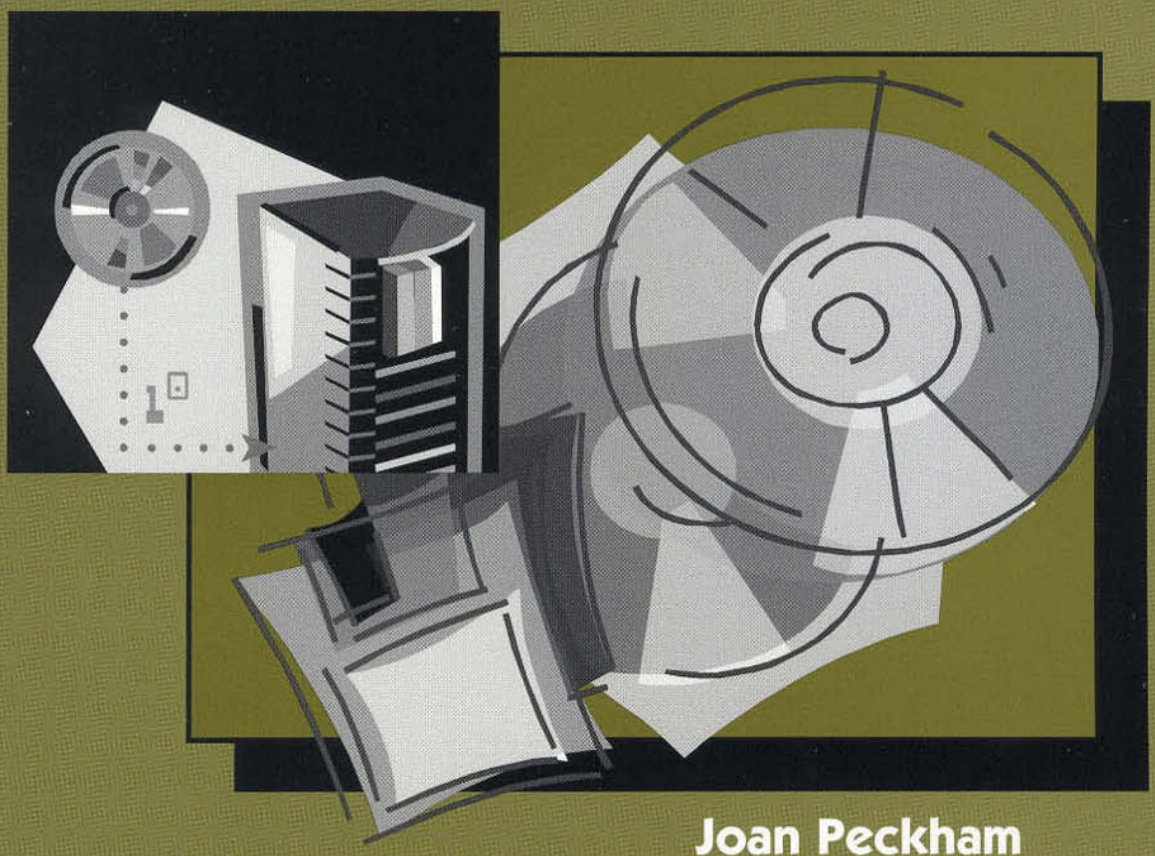
Practicing Software Engineering in the 21st Century



Joan Peckham
Scott J. Lloyd

IRM PRESS

Practicing Software Engineering in the 21st Century

edited by

Joan Peckham
University of Rhode Island, USA

and

Scott J. Lloyd
University of Rhode Island, USA

IRM Press
Publisher of innovative scholarly and professional information technology titles in the cyberage

Hershey • London • Melbourne • Singapore • Beijing

Acquisitions Editor: Mehdi Khosrow-Pour
Senior Managing Editor: Jan Travers
Managing Editor: Amanda Appicello
Typesetter: Jennifer Wetzel
Copy Editor: Heidi J. Hormel
Cover Design: Michelle Waters
Printed at: Integrated Book Technology

Published in the United States of America by
IRM Press (an imprint of Idea Group Inc.)
701 E. Chocolate Avenue, Suite 200
Hershey PA 17033-1240
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@idea-group.com
Web site: http://www.irm-press.com

and in the United Kingdom by
IRM Press
3 Henrietta Street
Covent Garden
London WC2E 8LU
Tel: 44 20 7240 0856
Fax: 44 20 7379 3313
Web site: http://www.eurospan.co.uk

Library of Congress Cataloging-in-Publication Data

Peckham, Joan, 1948-
Practicing software engineering in the 21st century / Joan Peckham and Scott J. Lloyd.
p. cm.
Includes bibliographical references and index.
ISBN 1-931777-50-0 -- ISBN 1-931777-66-7
1. Software engineering. I. Lloyd, Scott J. II. Title.
QA76.758.P42 2003
005.1--dc21

2002156236

ISBN: 1-931777-50-0
eISBN: 1-931777-66-7

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Other New Releases from IRM Press

- **Multimedia and Interactive Digital TV: Managing the Opportunities Created by Digital Convergence**/Margherita Pagani
  ISBN: 1-931777-38-1; eISBN: 1-931777-54-3 / US$59.95 / © 2003
- **Virtual Education: Cases in Learning & Teaching Technologies**/ Fawzi Albalooshi (Ed.), ISBN: 1-931777-39-X; eISBN: 1-931777-55-1 / US$59.95 / © 2003
- **Managing IT in Government, Business & Communities**/Gerry Gingrich (Ed.)
  ISBN: 1-931777-40-3; eISBN: 1-931777-56-X / US$59.95 / © 2003
- **Information Management: Support Systems & Multimedia Technology**/ George Ditsa (Ed.), ISBN: 1-931777-41-1; eISBN: 1-931777-57-8 / US$59.95 / © 2003
- **Managing Globally with Information Technology**/Sherif Kamel (Ed.)
  ISBN: 1-931777-42-X; eISBN: 1-931777-58-6 / US$59.95 / © 2003
- **Current Security Management & Ethical Issues of Information Technology**/Rasool Azari (Ed.), ISBN: 1-931777-43-8; eISBN: 1-931777-59-4 / US$59.95 / © 2003
- **UML and the Unified Process**/Liliana Favre (Ed.)
  ISBN: 1-931777-44-6; eISBN: 1-931777-60-8 / US$59.95 / © 2003
- **Business Strategies for Information Technology Management**/Kalle Kangas (Ed.)
  ISBN: 1-931777-45-4; eISBN: 1-931777-61-6 / US$59.95 / © 2003
- **Managing E-Commerce and Mobile Computing Technologies**/Julie Mariga (Ed.)
  ISBN: 1-931777-46-2; eISBN: 1-931777-62-4 / US$59.95 / © 2003
- **Effective Databases for Text & Document Management**/Shirley A. Becker (Ed.)
  ISBN: 1-931777-47-0; eISBN: 1-931777-63-2 / US$59.95 / © 2003
- **Technologies & Methodologies for Evaluating Information Technology in Business**/ Charles K. Davis (Ed.), ISBN: 1-931777-48-9; eISBN: 1-931777-64-0 / US$59.95 / © 2003
- **ERP & Data Warehousing in Organizations: Issues and Challenges**/Gerald Grant (Ed.), ISBN: 1-931777-49-7; eISBN: 1-931777-65-9 / US$59.95 / © 2003
- **Practicing Software Engineering in the 21st Century**/Joan Peckham & Scott J. Lloyd (Eds.),
  ISBN: 1-931777-50-0; eISBN: 1-931777-66-7 / US$59.95 / © 2003
- **Knowledge Management: Current Issues and Challenges**/Elayne Coakes (Ed.)
  ISBN: 1-931777-51-9; eISBN: 1-931777-67-5 / US$59.95 / © 2003
- **Computing Information Technology: The Human Side**/Steven Gordon (Ed.)
  ISBN: 1-931777-52-7; eISBN: 1-931777-68-3 / US$59.95 / © 2003
- **Current Issues in IT Education**/Tanya McGill (Ed.)
  ISBN: 1-931777-53-5; eISBN: 1-931777-69-1 / US$59.95 / © 2003

Excellent additions to your institution's library!
Recommend these titles to your Librarian!

To receive a copy of the IRM Press catalog, please contact (toll free) 1/800-345-4332, fax 1/717-533-8661, or visit the IRM Press Online Bookstore at: [http://www.irm-press.com]!

Note: All IRM Press books are also available as ebooks on netlibrary.com as well as other ebook sources. Contact Ms. Carrie Stull Skovrinskie at [cstull@idea-group.com] to receive a complete list of sources where you can obtain ebook information or IRM Press titles.

Practicing Software Engineering in the 21st Century

Table of Contents

Preface

Software engineering is a term that has a very broad definition. This process includes the logical design of a system; the development of prototypes, the automated generation of computer code for the system; the testing, validation and benchmarking of the code and the final implementation of the system. Once a new system is up and running, the software engineering process is used to maintain the system, evaluate its operation, keep track of new versions and refactor and/or reuse the code for other projects.

Over the past 30 years the discipline of software engineering has grown. In some cases, a specific programming paradigm, such as object-oriented, evolved into a broad discipline encompassing design and programming processes, tools and techniques. Several universities offer degrees as well as courses in software engineering. Standards for software engineering have been incorporated and formalized in England, Canada, Australia and the United States. Additionally, software engineering has received recognition from licensing and standards boards such as the Association of Computing Machinery (ACM) Institute of Electrical Engineering (IEEE), ISO 9000 and the Institute for Certification of Computing Professionals (ICCP).

Although many current design practices are focused on object-oriented techniques, this does not limit us to using object-oriented languages. It is quite possible to adopt the methods whether one writes in Fortran, C++ or writes scripts in Perl. In recent times the concept of software engineering has expanded to include not only code generation and system design, but a set of standards and methods that the software engineer should practice.

The practice of software engineering rightfully begins in the requirements phase of any system project, where the problem to be solved is well defined. Once this is captured, the design phase starts. In an effort to avoid the problem of "reinventing the wheel" a good designer decides what methods and patterns can be drawn from the existing software engineering "body of knowledge." Reusable generic design and code is only one advantage that has been realized today as the libraries of functions, patterns, and frameworks continue to grow.

Automated support for the application and integration of these reusable units with newly defined designs and modules using a Computer Aided Software Engineering (CASE) tool has created a new lexicon in this field. "Lower CASE" tools now refer to code generation while "higher CASE" tools are those tools used in the

construction and diagramming of proposed computer systems. There have recently been proposals to integrate these two capabilities into a single tool, so that once a system is proposed and analyzed using standard tools, such as Data Flow Diagrams (DFD), Entity Relationship Diagrams, and Unified Modeling Language (UML), this information is passed to another module of the tool to generate code consistent with these diagrams.

As previously mentioned in this preface, during the past 30 years a generalized body of knowledge about design as other aspects of software engineering processes has emerged with some generally accepted standards. The reader should refer to the "Guide to Software Engineering Body of Knowledge; SWEBOK" from IEEE Computer Society for an excellent description of this body of common knowledge and standards.

This book begins with a discussion of software patterns that are used to facilitate the reuse of object-oriented designs. While most CASE tools support the use of UML to extract the design from the software engineer and to assist in the development, most do not provide assistance in the integration and code generation of software patterns. In this chapter, the authors analyze the Iterator software pattern for the semantics that would be used in a CASE design tool to help the software engineer integrate this pattern into a design and then generate some of the code needed to implement the pattern. This work is based on semantic data modeling techniques that were previously proposed for the design of active databases.

The next chapter introduces a theoretical frame for processes definition validation in workflow processes with temporal restrictions. Workflow Interface 1 provides the process definition of the Work Flow Reference Model. This interface combines PNwC to provide the formalization and verification of systems based on the Petri Net theory with an extension. This extension allows the specification of temporal requirements via clock specification, using temporal invariants for the places and temporal conditions in the transitions. This chapter presents a technique to validate the process definition (PD) using Petri Nets with Clocks (PNwC). The algorithm for the analysis of a PNwC allows for the correction of errors in the modeling of the time variable. The algorithm generates information about temporal unreachable states and process deadlocks with temporal blocks. It also corrects activity invariants and transition conditions.

The third chapter identifies the key aspects of software engineering and systems engineering in an effort to highlight areas of consensus and conflict. The goal is to support current efforts by practitioners and academics in both disciplines to redefine their professions and bodies of knowledge. By using the Software Engineering Institute's Capability Maturity Model-Integrated (CMMISM) project, which combines best practices from the systems and software engineering disciplines, it can be shown that significant points of agreement and consensus are evident. Nevertheless, valid objections to such integration remain as areas of conflict. It is hoped that this chapter will provide an opportunity for these two communities to resolve unnecessary differences in terminology and methodologies that are reflected

in their differing perspectives and entrenched in their organizational cultures.

Historically the approach to software engineering has been based on a search for an optimal (ideal) methodology— the identification and application of a set of processes, methods and tools that can consistently and predictably lead to software development success. The fourth chapter presents the basis for pursuing a more flexible, adaptive approach. Less methodical techniques under a variety of names take what is described as a contingency-oriented approach. Because of the limitations in the nature of methodology, the high failure rate in software development, the need to develop methodology within an environmental context and the pressures of fast-paced "E" development, the authors argue that further exploration and definition of an adaptive, contingency-based approach to methodology is justified.

Chapter V challenges the established wisdom with respect to use cases. Use cases are classically elaborate to capture the functional requirements of the system by directly identifying objects, methods and data. Several authors of system analysis and design books advocate this approach. However the research reported in this paper indicates that there are better constructs for modeling use cases, at least initially. Further objects are not a particularly good medium for discussing requirements with users. This paper rehearses the arguments leading up to these conclusions and identifies some implications of these conclusions.

The theme of system development is continued in Chapter VI. Using the RAISE specification development process, a variety of components and infrastructures are built. These components are not independent and are related to each other, when the authors specify different systems into the same infrastructure. The RAISE method is based on the idea that software development is a stepwise, evolutionary process of applying semantics-preserving transitions. Reuse is crucially impacted in all the stages of the development, but there is no explicit reference to the specification of reusability in this development process. This chapter presents a rigorous process for reusability using RSL (RAISE Specification Language) components. The authors provide the mechanism to select a reusable component in order to guide RAISE developers in the software specification and construction process.

The section on system development concludes with Chapter VII. This chapter introduces the Functional and Object-Oriented Methodology (FOOM) . This is an integrated methodology for information systems analysis and design that combines two essential software-engineering paradigms: the functional/data approach (or process-oriented) and the object-oriented (OO) approach. FOOM has been applied to a variety of domains. This chapter presents the application of the methodology to the specification of the "IFIP Conference" system with focus on the analysis and design phases. The FOOM analysis phase includes data modeling and functional analysis activities, and produces an initial Class Diagram and a hierarchy of OO Data Flow Diagrams (OO-DFDs). The products of the design phase include: (a) a complete class diagram; (b) object classes for the menus, forms and reports and (c) a behavior schema, which consists of detailed descriptions of the methods

and the application transactions expressed in pseudocode and message diagrams.

Section II discusses methods to evaluate and manage the system development process. Chapter VIII presents a comprehensive quantitative management model for information technology. This methodology is assessment based and can be easily implemented without imposing an unacceptable organizational change. It supplies detailed information about the functioning of processes that allows managers to both effectively oversee operations and assess their prospective and ongoing execution risks. This offers a consistent risk reward evaluation.

Continuing with the theme of measurement and risk assessment Chapter IX describes the foundation and properties of specific object-oriented software measures. Many measures for object-oriented applications have been constructed and tested in development environments. However, the process of defining new measures is still alive. The reason for this lies in the difficulties associated with understanding and maintaining object-oriented applications. It is still difficult to relate the measures to the phenomena that need to be improved. Do current measurements indicate problems in reliability, maintenance or the unreasonable complexity of some portions of the application?

In order to reduce the complexity of software, new development methodologies and tools are being introduced. The authors talk about a new approach to development called separation of concern. Tools, such as Aspect/J or Hyper/J, facilitate the development process, but there does not seem to be a sound metrics suite to measure complexity and efficiency of applications developed and coded with Aspect/J or Hyper/J. In this chapter, the authors attempt to review the current research into object-oriented software metrics and suggest theoretical framework for complexity estimation and ranking of compositional units in object-oriented applications developed with Hyper/J.

Chapter X concludes the managing projects section by introducing a novel notion of temporal interaction diagrams that can be used for testing and evaluating distributed and parallel programming. An interaction diagram is a graphic view of computation processes and communication between different entities in distributed and parallel applications. It can be used for the specification, implementation and testing of interaction policies in distributed and parallel systems. Expressing interaction diagrams in a linear form, known as fragmentation, facilitates automation of design and testing of such systems. Existing interaction diagram formalisms lack the flexibility and capability of describing more general temporal order constraints. They only support rigid temporal order, hence have limited semantic expressiveness. The authors propose an improved interaction diagram formalism in which more general temporal constraints can be expressed. This enables the capture of multiple valid interaction sequences using a single interaction diagram.

Section III discusses specific applications and implementations that are best solved by the principles of software engineering. Chapter XI begins this section with relevant security issues that must be considered in any software implementation. While academicians and industry practitioners have long recognized the need

for securing information systems and computer architectures, there has recently been a heightened awareness of information technology (IT) management on computer-related security issues. IT managers are increasingly worried about possible attacks on computer facilities and software, especially for mission critical software. There are many dimensions to providing a secure computing environment for an organization, including computer viruses, Trojan horses, unauthorized accesses and intrusions and thefts to infrastructure. This complexity and multidimensional nature of establishing computer security requires that the problem be tackled on many fronts simultaneously. Research in the area of information systems security has traditionally focused on architecture, infrastructure and systems level security. Emerging literature on application-level security, while providing useful paradigms, remain isolated and disparate. The current study focuses on single, albeit an important, dimension of providing a safe and secure computing environment — application-software security.

The book progresses to a specific proposal for learning systems. Chapter XII presents a project proposal for future work utilizing software engineering concepts to produce learning processes in cognitive systems. This project outlines a number of directions in the fields of systems engineering, machine learning, knowledge engineering and profile theory that lead to the development of formal methods for the modeling and engineering of learning systems. This chapter describes a framework for formalization and engineering of the cognitive processes and is based on applications of computational methods. The work proposes the studies of cognitive processes in software development process, and considers a cognitive system as a multi-agent system of human cognitive agents. It is important to note that this framework can be applied to different types of learning systems. There are various techniques from different theories (e.g., system theory, quantum theory, neural networks) that can be used for the description of cognitive systems, which, in turn, can be represented by different types of cognitive agents.

Web-based applications are highlighted by Chapter XIII. Global competition among today's enterprises forces their business processes to evolve constantly, leading to changes in corresponding Web-based application systems. Most existing approaches that extend traditional software engineering to develop Web-based application systems are based on OO methods. Such methods emphasize modeling individual object behaviors rather than system behavior. This chapter proposes the Business Process-Based Methodology (BPBM) for developing such systems. It uses a business process as a unified conceptual framework for analyzing relationships between a business process and associated business objects, identifying business activities and designing OO components called business components. The authors propose measures for coupling and cohesion measurement in order to ensure that these business components enable the potential reusability. These business components can more clearly represent semantic system behaviors than linkages of individual object behaviors. A change made to one business process impacts some encapsulated atomic components within the respective business component without

affecting other parts of the system. A business component is divided into parts suitable for implementation of multi-tier Web-based application systems.

Geographic Information Systems (GIS) are presented in Chapter XIV. This chapter introduces an OO methodology for GIS development. It argues that a COTS-based development methodology combined with the UML can be extended to support the spatio-temporal peculiarities that characterize GIS applications. The authors suggest that by typifying both enterprises and developments, and, with a thorough knowledge of the software component granularity in the GIS domain, it will be possible to extend and adapt the proposed COTS-based methodologies to cover the full lifecycle. Moreover, some recommendations are outlined to translate the methodology to the commercial iCASE Rational Suite Enterprise and its relationships with tool kits proposed by some GIS COTS vendors.

Chapter XIV makes the claim of improved efficiency and reliability of networking technology, providing a framework for service discovery where clients connect to services over the network. It is based on a comparison of the client's requirements with the advertised capabilities of those services. Many service directory technologies exist to provide this middleware functionality, each with its own default set of service attributes that may be used for comparison and its own default search algorithms. Because the most expressive search ability might not be as important as robustness for directory services, the search algorithms provided are usually limited when compared to a service devoted entirely to intelligent service discovery.

To address the above problems, the authors propose a framework of intelligent service discovery running alongside a service directory that allows the search service to have a range of search algorithms available. The most appropriate algorithm may be chosen for a search according to the data types found in the search criteria. A specific implementation of this framework is presented as a Jini service, using a constraint satisfaction problem solving architecture that allows different algorithms to be used as library components.

Although component-based development (CBD) platforms and technologies, such as CORBA, COM+/.NET and enterprise Java Beans (EJB) are now de facto standards for implementation and deployment of complex enterprise distributed systems, according to the authors of Chapter XVI, the full benefit of the component way of thinking has not been gained. Current CBD approaches and methods treat components mainly as binary-code implementation packages or as larger grained business objects in system analysis and design. Little attention has been paid to the potential of the component way of thinking in filling the gap between business and IT issues. This chapter proposes a service-based approach to the component concept representing the point of convergence of business and technology concerns. The approach defines components as the main building blocks of business-driven service-based system architecture that provides effective business IT alignment.

The book now focuses its attention on specific issues of software engineering as applied to telecommunications networks. Chapter XVII describes the design of

a narrowband lowpass finite impulse response (FIR) filter using a small number of multipliers per output sample (MPS). The method is based on the use of a frequency-improved recursive running sum (RRS) called the sharpening RRS filter and the interpolated finite impulse response (IFIR) structure. The filter sharpening technique uses multiple copies of the same filter according to an amplitude change function (ACF), which maps a transfer function before sharpening to a desired form after sharpening. Three ACFs are used in the design as illustrated in examples contained in this chapter.

The book closes with Chapter XVIII, which describes another telecommunication application. This chapter presents the design of narrowband highpass linear-phase FIR filters using the sharpening RRS filter and the IFIR structure. The novelty of this technique is based on the use of a sharpening RRS filter as an image suppressor in the IFIR structure. In this way the total number of multiplications per output sample is considerably reduced.

The purpose of this book is to introduce new and original work from around the world that we believe expands the body of common knowledge in software engineering. The order of this book attempts to tell a story, beginning with the software process, including reusable code and specific design methodologies and the methods associated with this formalized structure. The book then proceeds to chapters that propose models to measure the system analysis and design process and to direct the successful development of computer systems. The chapters then progress to the next step in any system project — the implementation phase. This section includes various aspects of using and integrating the engineered software into a computer system. Its chapters address security and systems capable of learning. The book then concludes with specific examples of Web-based and telecommunication applications.

Acknowledgments

The editors would like to acknowledge the help of all involved in the collation and review process of the book without whose support the project could not have been satisfactorily completed. A further special note of thanks goes also to all the staff at Idea Group Inc., whose contributions throughout the whole process from inception of the initial idea to final publication have been invaluable. In particular, we thank Amanda Appicello who continuously prodded via e-mail to keep the project on schedule and Mehdi Khosrow-Pour whose enthusiasm motivated us to accept the invitation to take on this project.

Obviously in any project of this size it is impossible to remember, let alone mention, everyone who had a hand in this work becoming what it is today. Various graduate students and support staff from The University of Rhode Island were critical in creating this final product. Their support was vital in achieving what we hope is a well-edited publication. The authors deserve the greatest credit because their contributions were essential, giving us great material with which to work.

In closing, we again wish to thank all of the authors for their insights and excellent contributions to this book. We also want to thank all of the people who assisted us in the reviewing process. Of course our families deserve credit for simply putting up with us and supporting us. Our thanks to all these people!

Joan Peckham, Ph.D.
Scott J. Lloyd, Ph.D.
Kingston, RI, USA
September 2002

Section I

System Design

Chapter I

Integrating Patterns into CASE Tools

Joan Peckham
University of Rhode Island, USA

Scott J. Lloyd
University of Rhode Island, USA

ABSTRACT

Software patterns are used to facilitate the reuse of object-oriented designs. While most Computer Aided Software Engineering (CASE) tools support the use of Unified Modeling Language (UML) (Alhir & Oram, 1998) to extract the design from the software engineer and assist in development, most do not provide assistance in the integration and code generation of software patterns. In this chapter, we analyze the Iterator software pattern (Gamma et al., 1995) for the semantics that would be used in a CASE-design tool to help the software engineer to integrate this pattern into a design and then generate some of the code needed to implement the pattern. This work is based on semantic data modeling techniques that were previously proposed for the design of active databases (Brawner, MacKellar, Peckham & Vorbach, 1997; Peckham, MacKellar & Doherty, 1995).

INTRODUCTION

One of the intents of the object-oriented (OO) programming paradigm is to assist in the reuse of code through the use of classes that bundle data structures and procedures in such a way that they could more easily be moved from one implementation to another. When OO languages were first introduced, code libraries were developed to permit the sharing of objects and classes. At the same time, Object-Oriented Analysis and Design (OOAD) techniques were being developed (Booch, 1994; Coad & Yourdon, 1991; Jacobson, 1992; Rumbaugh et al., 1991; Wirfs-Brock, Wilkerson & Weiner, 1990). This gave us a set of notations for expressing the design of OO applications. The libraries also became a vehicle for the reuse of the OO designs and led to the capture of software patterns or designs that are frequently reused in software applications but are somewhat independent of particular application types. One of the most often cited book archives is *Design Patterns: Elements of Reusable Object-Oriented Software* (Gamma, Helm, Johnson, Vlissides & Booch, 1995). A combination of text, UML and code samples is used to communicate the patterns.

Early industrial experience indicates that patterns speed the development of systems but are hard to write (Beck et al., 1996). So a few CASE tools provide computer assistance to the programmer in choosing and integrating automatically generated code for the patterns in their applications. Tools that support the use of software patterns include those by Budinsky, Finnie, Vlissides and Yu (1996), Florijn, Meijers and van Winsen (1997) and Paulisch (1996). While these tools are just beginning to emerge, none have integrated code generation and general design in a generic way that permits seamless code specification with patterns. For example, the techniques are not generally language independent and are unable to generate code in more than one language. Some existing tools generate code but into a different workspace from the general software specification and coding environment, requiring the cutting and pasting of code from the pattern code space.

All software patterns have alternative implementations. These are typically explained using text and sample code. Software engineers are then expected to use this information to construct their own implementation of the pattern. Our goal here is to capture the semantics of patterns well enough that they can be presented to the software engineer via named choices in the CASE tool and then be used to generate the code. In Peckham and MacKellar (2002) we began to elaborate the choices in the Observer pattern of Gamma et al. (1995). In this paper we look at the Iterator pattern from the same source.

CASE TOOLS

CASE tools are used to assist software engineers in all aspects of the software lifecycle. These tools can help a team to manage, design, implement, test and

maintain a software project. They are believed to improve the documentation, correctness and consistency of software systems, especially large systems. They were developed to address the current "software crisis" (Sommerville, 2001) in which software projects are frequently over budget, error prone and delivered late (if at all). The CASE tools of interest here are those such as Rational Rose (Boggs & Boggs, 2002) that permit the design of software using UML.

While the design of software using a CASE tool and UML is helpful, we believe the software designer will get more help when higher level patterns can be used to compose the design (as opposed to the more fine-grained UML). Most case tools provide the designer with a set of UML constructs with which to design the software, but this design is on the level of software modules and the relationships between them. If patterns are used as the building blocks of the software design, then well understood collections of modules could be used to compose a design. This raises the level on which the software is being designed and can assist in more rapid design and development of the software.

Most CASE tools of Rational Rose's ilk also generate code from the UML design. Most existing pattern tools simply assist in a "cut and paste" process whereby the designer selects a pattern and is given a piece of code in the appropriate language to paste into the implementation. The programmer then needs to tailor the code to the implementation. There are several things wrong with this process:

1. The developer has to write the code. If we can extract the meaning and semantics of the patterns, and if we have an internal representation of these semantics, we can generate much of the code for the developer.
2. If the developer wishes to change the design, it can become time consuming and confusing to carry out and to keep track of all changes. If the CASE tool can provide the programmer with alternative pattern implementations, then much of the code can be generated once an alternative is selected. This is faster than having to modify the code "by hand."

As the reader can see from above, if the system can generate the code instead of providing a unit of code to cut and paste, the development and modification processes will be quicker and less error prone.

A PATTERNS PRIMER

As laid out in Gamma et al. (1995), patterns are described using diagrams, textual descriptions and, sometimes, with examples using a pseudolanguage or pseudocode. Included in each pattern description are:

Name - The name of the pattern;

Classification - The type of pattern based upon the author's classification system (there are three: creational, structural and behavioral);

Intent - The purpose of the pattern or what it does;
Also known as - Other used or known names for the pattern;
Motivation - A concrete example to help you understand the general pattern;
Applicability - Where to use the pattern;
Structure - A graphical representation of the pattern using a UML-like notation;
Participants - Description of the participating objects and classes;
Collaborations - How the participants interact;
Consequences - How the pattern carries out its goals and the results (positive and negative);
Implementation - Tips, possible pitfalls and language specific issues;
Sample code - Some examples in a specific language;
Known uses - Where the pattern has been used in widely known commercial or research software.

These pattern characteristics help the reader to understand the nature of the pattern. From these descriptions we can extract the alternative semantics for the pattern and use them to generate the code. Patterns can be thought of as providing notations for fairly sophisticated software implementations. These are language independent in that they can be implemented in any language, so the languages used for their implementation can be of any type. Although the conceptual models of patterns are usually OO in nature, this can just be thought of as employing the prevalent language type of the era. However the patterns can be mapped to any programming language if it supports the organization of code into modules and relationships among the modules. The relationships among the modules can be thought of as representing procedural activity along the relationship paths.

The above concepts are so fundamental to programming that they are present in most language types, including procedural and scripting, as well as object-oriented languages. So an application in which there are many language types used for implementation can be designed using patterns. It is not out of the question to include languages such as Perl, C, C++, Java, Python, etc.

To illustrate the meaning that a pattern can convey independent of the implementation language, consider "client-server." This term conjures up an arrangement between two software components that quickly conveys a structure and a communications approach. A software engineer choosing this architecture need only mention the term to a software team to convey a lot of meaning. It is this expressive power that makes patterns so attractive. However, as we will point out more clearly below, there are many implementation details that have yet to be chosen after a pattern, such as client-server, is decided upon. Here we will talk about such issues with the Iterator pattern.

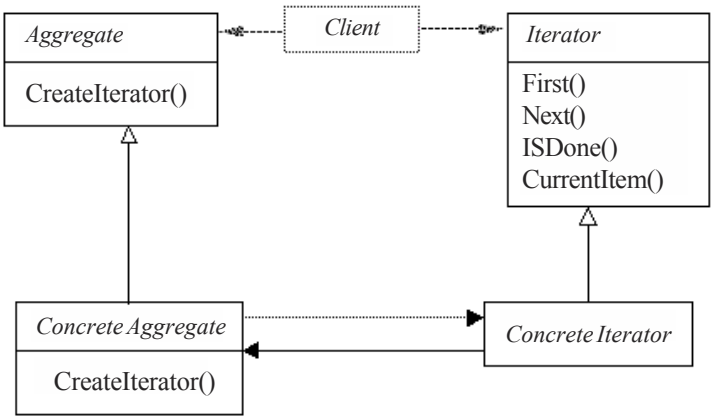
THE ITERATOR PATTERN

In Gamma et al. (1995), the Iterator pattern is described. The purpose of this pattern is to "provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation." The intent here is to permit transparency in the access and use of the aggregate object. This permits the user of the aggregate to traverse it without having to know the details of its structure and implementation. This is a primary OO design principle that permits the encapsulation of particular modules of code to reduce errors and permit easy code maintenance. This decouples the aggregate object from the rest of the code and makes it easier to export to other applications. At the same time, the aggregate is still usable by the application.

The basic structure of the Iterator pattern is shown in Figure 1. The client (the code that uses the Iterator) is shown in light print. There are two roles in this pattern known as Aggregate and Iterator. The client must know of the existence of both. It calls the CreateIterator method in Aggregate to create Iterator, which then assists it to traverse Aggregate. But since this method is a part of the Aggregate class, the details of how to iterate and, thus, create the Iterator are hidden from the client. The client, however, also knows that the methods First, Next, IsDone, and CurrentItem will be available once the Iterator is created.

Notice that the pattern makes use of abstract and concrete classes in the definition of the Iterator and Aggregate. An abstract class is used to describe the characteristics, structure and methods that subclasses in an implementation may have in common. Each of the subclasses are said to inherit these characteristics and may have additional structure and behavior. An abstract class cannot be instantiated in a given program. It is used only to convey these structures and behaviors. The more specialized concrete classes are instantiated in the implementation as objects.

Figure 1. Iterator Pattern, Basic Structure



The use of abstract and concrete classes here permits, for example, the creation of method signatures in the abstract class that are generic enough to be reused in a variety of applications and a concrete version with the elaborated code to be used in a specific application.

To implement the Iterator pattern the software engineer must normally read a textual description in a book, develop a refined design and implement "by hand" the features of the pattern that are important for the intended application. For example:

Delete iterators: To prevent "memory leaks" caused by objects that are created and then not deleted once they are no longer needed in the program, there needs to be a means for the client to delete the Iterator object. This is especially important in C++ where memory management is not supplied by the run-time program. Gamma et al. (1995) suggest that an IteratorPtr class is created to simplify the code needed to access the Aggregate object and to assure that a destructor is called to remove the object when it is no longer used. Most of the code can be generated when the Iterator pattern is chosen. Thus, the software engineer saves time and prevents errors by not having to code this feature each time the pattern is used.

The above is a feature that should always be used when coding the pattern in C++. In the next example, we look at pattern options that would need alternative implementations for the same feature (iteration). In some cases, different code can be generated; therefore, the CASE tool provides the software engineer with a means to choose the specifics of a particular implementation.

List Iteration Choices

There are two choices for iterating over the aggregate object — internal (II) and external (EI). If the client controls the iteration by explicitly requesting the next element from the iterator, then this is EI. In this mode the client requests an element, does some processing with the returned element and then contacts the aggregate object again for the next object. The other alternative is that the iterator carries out the whole iteration once the client begins the process. In this mode, the aggregate object will return a list (or other multi-valued) result. The intended application will of course determine this choice, and the software engineer should merely implement the appropriate method.

Once II or EI is chosen, there are additional alternatives for implementation of the Iterator pattern. Another challenge here is that we are planning a technique to generate code to iterate over an aggregate object without knowing in advance its type and thus the functions needed to iterate over it. For example, if generating code in a language that does not have adequate support for the parameterization of the functions used to iterate, then the internal iteration alternative can be clumsy to implement. There are two choices available to the software engineer: (1) Pass a pointer to the function needed to be applied iteratively, or (2) Use inheritance to

resolve the particular functions needed. Again the software engineer can select the best choice.

Some of the examples above bring up an important point: The choices provided to the software designer will sometimes be dependent on the language to which the pattern is mapped. So different strategies will need to be used when generating code in different languages. Similarly the software design interface will sometimes need to be asked different questions depending on the implementation language intent. The designer of the pattern interface will need to separate the language independent and dependent alternatives of the patterns.

The code for applying the internal Iterator is given in Gamma et al. (1995), although knowing the type of aggregate object on which the iteration is being carried out and the type of activity performed during iteration is needed. In the example given, the aggregate object was a list of employees and the operator applied upon iteration was PrintEmployees. The code generation facility in the CASE tool can generate the code and leave the specific aggregate and operators blank with instructions for the software engineer to fill in. For example, here is a passage of C++ code needed to implement an internal Iterator:

```
template <class Item>
class ListTraverser {
public:
      ListTraverser (List <Item>* aList);
      Bool Traverse ();
protected:
      Virtual bool ProcessItem (const Item&) = 0;
      private: ListIterator<Item> _iterator;
      };
```

This code can be generalized and given slots to be filled in for the particular application in question. For example the software engineer can be presented with the following that includes a legend that annotates the code with a guide of how to fill in the italicized code:

```
template <class AggItem>
class AggTraverser {
public:
      AggTraverser (Agg <AggItem>* AggType);
      Bool Traverse ();
protected:
      Virtual bool ProcessAggItem (const AggItem&) = 0;
      private: AggIterator<AggItem> _iterator;
      };
```

In some cases, the implementation choices are not very specific and very little code can be generated. But, even if only class and method headers are generated, this can assist the programmer in making sure that all of these necessary features are present.

FUTURE WORK

At this point in time, we have only analyzed a few patterns for their semantics. In the future we plan to continue analyzing other patterns and soon look for underlying structures that are similar or the same in collections of patterns. Then we will devise an internal language to represent these constructs. This will be used to store the software designer's software specification with patterns and then used to generate the code. Similarly, while there is a UML to specify the design of an application, we might augment the notation of UML with the ability to specify the patterns that are integrated within the design and to clearly and precisely indicate the alternative pattern implementations.

A means is needed to integrate various patterns within a design and to integrate the patterns with other software. For example, the pattern books outline how specific patterns can be used with each other in a system design. The CASE tool can use this information to offer assisting and cooperating patterns to the designer and begin to generate the code. It can also provide checking assistance in identifying problems or errors that are frequently made when specific patterns or combinations of patterns are selected.

Some preliminary ideas for a pattern wizard to be integrated into a CASE tool and for a pattern code generation system have been outlined in Qin (2002).

CONCLUSIONS

Software patterns can be very useful in preventing software designers and developers from "reinventing the wheel" for every new utilization of a specific pattern. This can lower the cost of software development and maintenance, providing more robust and error-free code. In addition, applications employing patterns are more likely to have a shorter development period. Currently software patterns have not been integrated into CASE tools in a robust manner. In this paper we have shown how we envision the integration of patterns into CASE design tools to assist in the selection of the proper patterns and to partially generate the code to implement the Iterator pattern. As is pointed out in the previous section, there are numerous other applications for this type of software tool and these will be explored in future research.

ACKNOWLEDGMENT

Thank you to Heng Chen, Chen Gu and Mingsong Zheng of Professor Peckham's CSC 509 (Software Engineering) class for beginning to outline the semantics of this pattern in a classroom exercise for us.

REFERENCES

Alhir, A., & Oram, A. (1998). *UML in a nutshell: A desktop quick reference (nutshell handbook)*. Sebastapol, CA: O'Reilly and Associates.

Beck, D., et al. (1996). Industrial experience with design patterns. *Proceedings of ICSE-18 (International Conference on Software Engineering)* (pp. 103-113) Technical University of Berlin, Germany.

Boggs, W., & Boggs, M. (2002). *Mastering UML with rational rose 2002*. Almeda, CA: Sybex.

Booch, G. (1994) *Object oriented analysis and design* (2nd ed.). San Francisco, CA: Benjamin Cummings.

Brawner, J., MacKellar, B., Peckham, J., & Vorbach, J. (1997). Automatic generation of update rules to enforce consistency constraints in design databases'. In Spaccapietra & Maryanski (Eds.), *Proceedings of the 7th IFIP 2.6 Working Conference on Database Semantics (DS-7) Searching for Semantics: Data mining, reverse engineering, etc*. Chapman and Hall.

Budinsky, F., Finnie, M., Vlissides, J., & Yu, P. (1996). Automatic code generation from design patterns. *IBM Systems Journal 35*(2), 151-171.

Coad, P., & Yourdon, E. (1991). *Object-oriented analysis* (2nd ed.). Upper Saddle River, NJ: Yourdon Press.

Florijn, G., Meijers, M., & van Winsen, P. (1997). Tool support for object-oriented patterns. *Proceedings of ECOOP (European Conference on Object-Oriented Programming) '97,* Jyväskylä, Finland.

Gamma, Helm, Johnson, Vlissides, & Booch. (1995). *Design patterns: Elements of reusable object-oriented software*. New York: Addison-Wesley.

Jacobson, I. (1992). *Object-oriented software engineering.* New York: Addison-Wesley.

Paulisch, F. (1996). Tool support for software architecture. SIGSOFT (Special Interest Group on Software Engineering) '96 Workshop. San Francisco, CA (pp. 98-100). Joint Proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints '96) on SIGSOFT '96 workshops. San Francisco, CA. New York: ACM Press.

Peckham, J., & MacKellar, B. (2001, April). Generating code for engineering design systems using software patterns. *Artificial Intelligence in Engineering,* 15(2), 219-200.

Peckham, J., MacKellar, B., & Doherty, M. (1995). A data model for the extensible support of explicit relationships in design databases. *VLDB (Very Large Databases) Journal, 4*(2), 157-159.

Qin, Y. (2002) Use of design patterns in case tool environment. Unpublished master's thesis, University of Rhode Island, Kingston.

Rumbaugh, J., et al. (1991). *Object oriented analysis and design.* Upper Saddle River, NJ: Prentice-Hall.

Sommerville, I. (2001). *Software engineering* (6th ed., p. 4). Addison Wesley.

Wirfs-Brock, Wilkerson, & Weiner, L. (1990). *Designing object-oriented software.* Upper Saddle River, NJ: Prentice-Hall.

Chapter II

Petri Nets with Clocks for the Analytical Validation of Business Process

Gabriel Vilallonga
Universidad Nacional de San Luis, Argentina

Daniel Riesco
Universidad Nacional de San Luis, Argentina

Germán Montejano
Universidad Nacional de San Luis, Argentina

Roberto Uzal
Universidad Nacional de San Luis, Argentina

ABSTRACT

This chapter introduces a theoretical frame for the Process Definition (PD) validation in Workflow or in those processes with temporal restrictions. The Interface 1 of Workflow makes the PD of the Work Flow Reference Model. This interface combined with Petri Nets with Clocks (PNwC) allows the formalization and verification of systems, based on the Petri Net theory and the extension. This extension allows the specification of temporal requirements via clocks specification, using temporal invariants for the places and temporal conditions in the transitions.

In this chapter we present a technique to validate the Process Definition (PD) by means of PNwC. The algorithm for the analysis of a PNwC allows correction of errors in the modeling of the time variable. The algorithm generates information about temporal unreachable states and process deadlocks with temporal blocks. Also, it corrects activities invariants and transitions conditions.

INTRODUCTION

Recently, Business Process Reengineering (BPR) has become one of the most popular topics at conferences on business management and information-systems design. BPR is the fundamental rethinking and radical redesign of business processes to achieve dramatic improvement in critical, contemporary measures of performance, such as cost, quality, service and speed. BPR implies taking a comprehensive view of the entire existing operation, analyzing and trying to redesign it in a way that uses new technology to better serve the customers (Jacobson, Ericsson & Jacobson, 1995).

Workflow is an important and valuable technology in BPR. It is a discipline, practice and concept. Most workflow engines can handle very complex series of processes with a workflow system. Workflow is normally comprised of a number of logical steps, each of which is known as an activity. An activity can involve manual interaction with a user or a workflow participant, or the activity might be executed using machine resources. Delivering work to users does increase efficiency. Automating the actual work provides huge increases in efficiency and provides managers with the facilities to create a virtual organization and participate effectively in the e-commerce revolution (Allen, 2000).

Petri Net (PN) is a tool for studying and modeling systems. PN theory allows system modeling and obtaining a mathematical representation of the system (Peterson, 1981). An important part of these system requirements are the temporal ones. The growing complexity and critical nature of these systems have motivated the search for verification methods (Alur, Courcoubetis & Dill, 1990; Ghezzi, Mandrioli, Morascas & Pezze, 1991). The PN models use an efficient method of analysis of network behavior. More networks are analyzed by simulations than by the generation of the state space. This does not guarantee that the states with very low probability happen in long runs. The analysis of these states with low probability can be the object of a serious analysis, for example, the loss of a message in a communication network. The systematic analysis of the state space takes all the events into account, even those that are improbable.

The PNwC proposed in Montejano, Riesco, Vilallonga, Dasso and Favre (1998) and Riesco, Montejano, Vilallonga, Dasso and Uzal (1999) has a high expressive power in the concurrent and asynchronous process modeling and allows modeling real-time systems. PNwC includes additional temporal elements, clocks, which are

not taken into consideration in the literature concerning the extensions of PN with time (Ghezzi, et al., 1991; Zuberek, 1991; Ghezzi, Mandrioli, Morascas & Pezze, 1989; Buy & Sloan, 1994). In addition to the power of structure analysis and system behavior that PN has, there has been an added structure of an algorithm that allows for the temporal analysis of the extended PN (Montejano, 1998). This algorithm detects deadlock and temporal blockings as well as the consistency of the restrictions.

We present here a technique for a temporal validation of the workflow PD using PNwC, allowing for the study of the models through a theory like PNwC.

WORKFLOW

Workflow is the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules (Allen, 2000). A Workflow Management System (WMS) is a system that defines, creates and manages the execution of workflow through the use of software, running on one or more workflow engines, which are able to interpret the process definition, interact with workflow participants and, where required, invoke the use of information technology (IT) tools and applications.

The definition of a WMS is that it can often interpret a workflow definition produced separately, manage triggers, alarms, and interoperate with external systems. It is important to note that while this definition covers the spectrum of workflow systems, it excludes simple relational database systems and e-mail with programmable delivery mechanisms.

All workflow systems are process oriented. A process definition, a representation of what should happen, is created, and it typically is comprised of some subprocess. This process would be split into a number of different sub processes. Each process and subprocess is comprised of some activities. An activity is a single logical step in the process. It is sometimes not practical to automate all activities during a single project. Therefore, workflow executes automated activities while process definition will describe all activities whether they are automatic or manual.

Interface 1: PD

The WRM that describes five interfaces represents the Workflow Engine Interoperability. We will focus on Interface 1 and PD tools to work on the definition of processes (Workflow Management Coalition, 1999b). The Interface 1 definition deals with passing PD from external tools to the workflow engine where it is enacted. The interface also defines a formal separation between the development and run-time environments, enabling a PD, generated by one modeling tool, to be used as input to a number of different workflow run-time products. This meets the often-

expressed user's requirement for the independence of modeling and workflow run-time products.

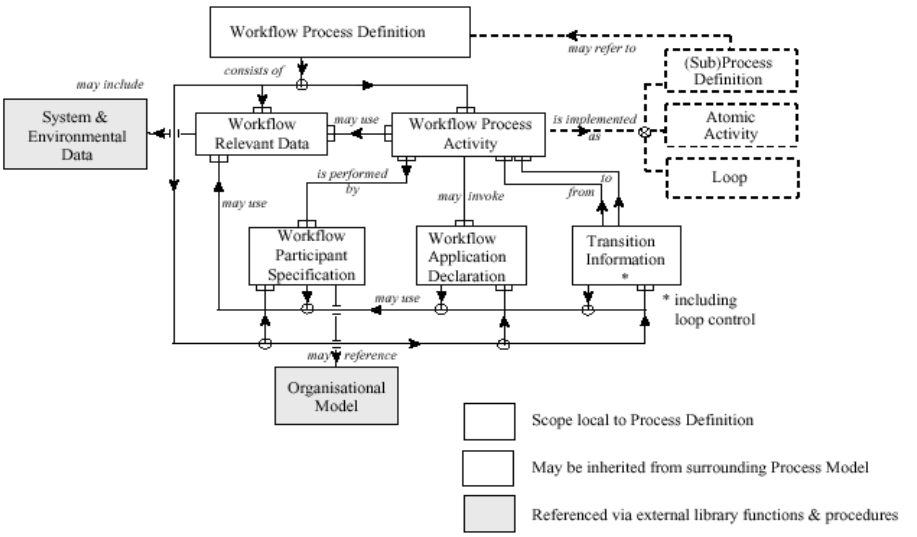
A PD is defined as "the representation of a business process in a form that support automated manipulation, such as modeling, or enactment by a workflow management system. The process definition consist of a network of activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual activities, such as participants, associated IT applications, and data, etc." (Workflow Management Coalition, 1999a).

The PD is interpreted by the workflow engine, acting as a template for the creation and control of that process enactment. The PD may contain references to sub-processes, separately defined, which make up part of the overall PD. An initial PD will contain the minimal set of objects and attributes necessary to initiate and support process execution.

The Interface 1 supports PD Import and Export. This interface includes a common Meta-Model for describing the PD (this specification), also a textual grammar for the interchange of PDs (Workflow Process Definition Language (WPDL)) and APIs for the manipulation of PD data.

The WPD describes the process itself. The Meta-Model describes the top-level entities contained within a WPD, their relationships and attributes (Figure 1). These entities provide header information for the PD and are therefore related to all other entities in that process. For each top-level entity there is an associated set of attributes, some mandatory and others optional, which describe the characteristics of the entity.

Figure 1. Metamodel Top-Level Entities



In this section we will concentrate on the relationship between different activities, Transition Information and their implementation.

Workflow Process Activity

A PD consists of one or more activities, each comprised of a logical, self-contained unit of work within the PD. An activity represents work, which will be processed by a combination of resource and/or computer applications. The scope of an activity is local to a specific PD.

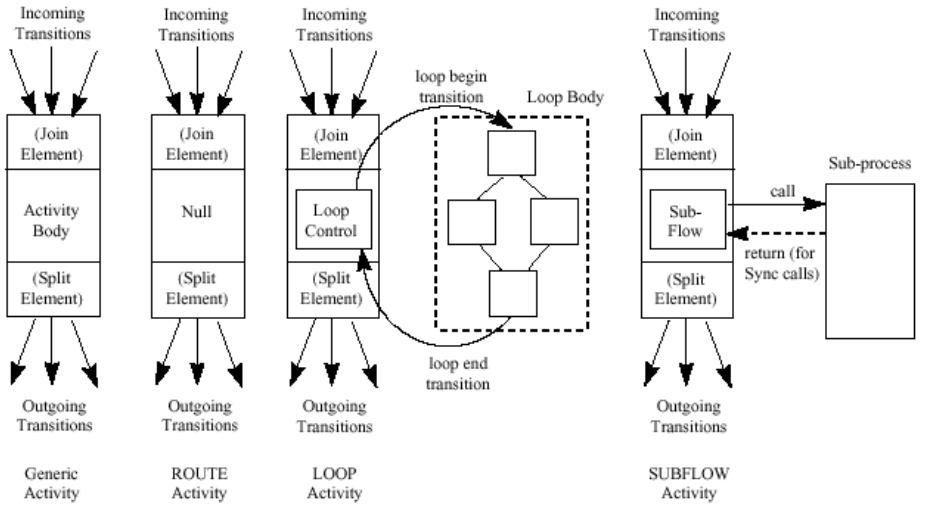
An activity may be atomic and, in this case, is the smallest unit of self-contained work, which is specified within the process. An activity may generate several individual work items for presentation to a user invoking, such as different IT tools.

The Workflow Activity Definition is used to define each elementary activity that makes up a workflow process. This workflow has attributes that may specify activity control information, implementation alternatives, performer assignment, run-time relevant information like priority, and data used in BPR and simulation situations.

Entity Type Relationship for Different Activity Types

The activity description is used to describe several different activity types. All these activities share the same general activity attributes, but the usage of other attributes, particularly participant and applications assignment and the use of workflow-relevant data, may be specific to the activity type.

Figure 2. Activity Structure and Transition Conditions



In general normal transition restrictions may be declared at the level of the activity boundary within the surrounding process, whereas specialized flow conditions (subflow, loops or internal parts of a route activity) operate "internal" to the activity (but may reference activities within the surrounding PD). Figure 2 is used to illustrate the generic structure of an activity and the above variants.

An activity may be implemented in one of four ways: No Implementation — this activity is implemented by manual procedures; Application — the implementation is supported by, one or more applications; Subflow — the activity is refined as a subprocess. This subprocess may be executed synchronously or asynchronously and a Loop — a loop control and the loop body. The loop body is connected with the Loop Control Activity by the corresponding Loop connecting Transitions.

The Transition Restriction has special attributes like JOIN and SPLIT description. A JOIN describes the semantic if multiple incoming Transitions exist for an activity. There are two possible ways to express a JOIN: AND JOIN and XOR JOIN. The AND JOIN can be treated as a "rendezvous precondition" of the activity. The activity is not initiated until the transition conditions on all incoming routes are TRUE. The XOR JOIN initiates the activity when the transition conditions of any of the incoming transitions are True.

A SPLIT describes the semantics of multiple outgoing transitions for an activity. There are two possible ways to express a SPLIT: AND SPLIT and XOR SPLIT. An AND SPLIT with transition conditions may be referred to as "conditional AND," "multiple-choice OR" or "nonexclusive OR.". The XOR-SPLIT has a list of identifiers of outgoing transition activity, representing alternatively executed transitions.

Activities are related to one another via flow control conditions (transition information). The Transition Information describes possible transition between activities and the conditions, which enables or disables during the workflow execution.

A PD is seen as a network of edges between the activity nodes and as a workflow process diagram. All edges are directed and given by a pair of activities (From <node>, To <node>). The edge of the activity net may be labelled by transition conditions**.** Transition from one activity to another may be conditional (involving expressions that are evaluated to permit or inhibit the transition) or unconditional. A transition condition for a specific edge enables that transition if the condition is TRUE. If no routing condition is specified, the transition behaves as if the condition with a value TRUE is present. If there are multiple incoming or outgoing transition activities, then further options, expressing control flow restrictions and condition evaluations semantics, are provided in the activity entity definition (AND/XOR variants of SPLIT/JOIN).

There are two possible transitions: "regular" and loop-connecting. For "regular" transitions, it is possible to define or synchronize multiple, concurrent or alternative, control threads, JOIN, SPLIT, and sequence transitions between activities, cascad-

ing transition/conditions, and blocking restrictions. The transitions within a process may result in the sequential or parallel operation of individual activities within the process. The information related to associated SPLIT or JOIN conditions is defined within the appropriate activity with split a form of "post-activity" processing in the to-activity. Loops-connecting transitions allow the expression of cycles in the transitions network. They connect the body of a Loop with the Loop Activity that is implemented by this body.

PETRI NETS WITH CLOCKS

PN is limited in its modeling and design power when used for systems where time is part of the system specification. Timed Graphs (TG) (Sifakis & Yovine, 1996; Olivero, 1994; Yovine, 1993; Daws, Olivero, Tripakis & Yovine, 1996), on the other hand, are a useful tool to specify system time constraints.

The power inherent of TG and PN has motivated us to extend PN theory with temporal requisites using TG. In addition to the power of structure analysis and system behavior that PN has, we have added an algorithm structure that allows the temporal analysis of the extended PN (Riesco, 1999).

With the extension of PN using TG, we get the advantage in modeling systems with asynchronous and concurrent behavior (PN) and the possibilities of formalizing systems with temporal requisites. The PNwC described in this paper is Timed Place PN, since the time is associated in the places. This time is specified by means of a place invariant using clocks. The invariants represent deadlines. When the time reaches the deadline, a transition that has this place, like an input place, must be executed.

A transition is enabled when the input places have the necessary number of tokens and, also, the values of the system clocks satisfy the invariants of each input place. The firing of a transition takes place if this is enabled and, using the system clocks, has to satisfy the condition associated with this one. This new firing causes the creation of tokens in outputs place and the update of the system clocks.

Definition 1.

A Clock x: Clock **x** is a positive real variable, i.e., $\mathbf{x} \in \mathbf{R}^{+0} = \{ \mathbf{z} / \mathbf{z} \in \mathbf{R}^{+} \vee \mathbf{z} = \mathbf{0} \}$; where $\mathbf{R}^{+}$ are all real positives.

Definition 2.

Set of all Clocks X: $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ is the set of all Clocks.

Definition 3.

Restricted predicates Ω_X: Ω_X is a set of restricted predicates on the places defined as a Boolean combination of atoms that take the form $\mathbf{x} \# \mathbf{c}$, where $\mathbf{x} \in \mathbf{X}$, # is a binary relation on the set $\{<, >, =, \geq, \leq\}$ and $\mathbf{c} \in \mathbf{R}^{+0}$.

Definition 4.

Restricted predicates Ψ_X: Ψ_X is a set of restricted predicates on the transitions defined as a Boolean combination of atoms that take the form **c #**

$x \# c'$ or the form $x \# c'$ where $x \in X$, # is a binary relation on the set $\{<, =, \leq\}$, $c, c' \in R^{+0}$. Every clock in Ψ_x must have an upper limit c'.

Definition 5.

Valuation VaL: VaL is a set of all vectors of dimension **k**, where **k** is the cardinality of set **X**, and each element of the vector belong to R^{+0},

$VaL = \{v / v = (v_1, \ldots, v_k) \wedge k = |X| \wedge \forall j, 1 \leq j \leq k \cdot v_j \in R^{+0}\}$.

Definition 6.

PNwC: A PNwC is a PN extended based on TG with a finite set of clocks whose values are incremented uniformly with time. The restrictions associated with the system are expressed by invariants on places and the association of an enabling condition with each transition. A clock can be reset in each transition. Also the firing of a transition shall be an instantaneous action that does not consume time. Time runs only at places and for no more than what is established by the associated invariant. Formally the structure of a PNwC is a t-uple:

$PNwC = < P, T, I, O, X, Inv, C, A >$, where:

- $P = \{ p_1, p_1, \ldots, p_n \}$ is a finite set of places, $n \geq 0$.
- $T = \{ t_1, t_1, \ldots, t_m \}$ is a finite set of *transitions*, $m \geq 0$, $P \cap T = \varnothing$.
- $I: T \rightarrow P^{\infty}$, is the input function and a mapping from transitions to bags of places.
- $O: T \rightarrow P^{\infty}$, is the output function and a mapping from transitions to bags of places.
- **X** is in Definition 2.
- $Inv: P \rightarrow \Omega$, associates to each place $p_i \in P$, a restricted predicate $\Omega \in \Omega_X$ (Definition 3) called place invariant.
- $C: T \rightarrow \Psi$, associates to each transition $t \in T$, a restricted predicate $\Psi \in \Psi_X$ (Definition 4) called transition condition.
- $A: T \rightarrow w$, set of clocks of the transition that are initialized to zero $w \subseteq X$.

Definition 7.

Affectation α: An affectation α is a function $\alpha: VaL \times T \rightarrow VaL$

$\alpha(v, t) = v'$ iff $\forall x_i \in X \cdot [x_i \in A(t) \Rightarrow v'(i) = 0 \wedge x_i \notin A(t) \Rightarrow v'(i) = v(i)]$.

Definition 8.

Marked Timed Petri Net (MPNwC): A MPNwC is defined as $MPNwC = < P, T, I, O, X, Inv, C, A, \mu >$ where **P, T, I, O, X, Inv, C** and **A** are in Definition 6, and $\mu \in M$.

The marking μ is as an n-vector $\mu = (\mu_1, \mu_2 \ldots \mu_n)$, with $n = |P|$ and $\mu_i \in Nat_0$ with $1 \leq i \leq n$.

The set of all marking **M** is the set of all vectors of dimension **n**, $M \subseteq Nat_0^{n.}$,

$M = \{\mu = (\mu_1, \ldots, \mu_n) \wedge n = |P| \wedge \forall i, 1 \leq i \leq n \cdot \mu_i \in Nat_0\}$

Definition 9.

Invariant of the Marking InvM(μ): Invariant of the Marking **InvM(μ)** is the conjunction of the invariants of the places where the number of tokens is greater than zero.

InvM(μ) iff Λ **Inv(p_i)** where $p_i \in P \wedge \mu(p_i) > 0$

Definition 10.

A Predicate applied to Valuation Φ[v]: Φ[v] iff **($(x_i \# c)$** is in **Φ)** $\Rightarrow$ **v(i) # c** holds, where $x_i \in X$, # is a binary relation on the set $\{<, >, =, \geq, \leq\}$, $c \in R^{+0}$.

Definition11.

A State q: A state of a **PNwC** is a pair q = (μ, v), μ ∈ M and v ∈ VaL, where the valuation v of the clocks satisfy the invariants of the net's places, i.e., InvM(μ)[v] holds.

Definition 12.

The set of all states Q: The set of all possible states of a PNwC is represented by $Q \subseteq M \times VaL$ such that $Q = \{(\mu, v) \mid \mu \in M \wedge$ **InvM(μ)[v]** holds $\}$

Execution of MPNwC

The execution of the MPNwC is done using the successor marking and the system state changes.

Definition 13.

Enabled Transition in MPNwC E(t, q): Let **q = (μ, v)** be a possible state of a PNwC, where **μ** is a marking and **v** the valuation of the clocks. In **q**, the valuation of the clocks satisfies the associated invariants to each place in the marking by Definition 11 A transition **t** ∈ **T** in a MPNwC is enabled **E(t, q)** in the state **q = (μ, v), E(t, q)** iff $\forall p_i \in I(t) \cdot \mu(p_i) \geq \#(p_i, I(p_i)) \wedge$ **C(t)[v]** holds (Definitions 6 and 10).

Definition 14.

System State Changing → : The System State Changing is represented by the following expression:

$SC = \langle \Theta, \rightarrow \rangle$,

where **Q** is the set of all states.

The changing relation $\rightarrow \subseteq Q \times (T \cup R^+) \times Q$ has two types of changing: temporal and instantaneous. The notation is $q \rightarrow^{time} q'$ for temporal changing and $q \rightarrow^{t} q'$ for instantaneous changing, where $q, q' \in Q$ and $time, t \in T \cup R^+$.

Definition 15.

Temporal State Changing $\rightarrow^{time}$: Temporal state changing represents the elapse of time by a changing labeled **time** from the state **(μ, v)** to **(μ, v+time)**, $(\mu, v) \in Q$, $time \in R^{+0}$.

$(\mu, v) \rightarrow^{time} (\mu, v+time)$ iff $\forall y, y \in R^{+0} \cdot 0 \leq y \leq time \Rightarrow$ **InvM(μ)[v+y]** holds (Definition 9).

Definition 16.

Instantaneous or discrete State Changing (by transition) $\rightarrow^t$: An instantaneous state changing is given by the execution of a transition $\mathbf{t \in T}$, where the changing is labeled **t,** from the state $\mathbf{(\mu, v)}$ to state $\mathbf{(\mu', v')}$ as Definition 14.

$\mathbf{(\mu, v) \rightarrow^t (\mu', v')}$ iff $\mathbf{E(t, (\mu, v)) \wedge \alpha(v, t) = v'}$ (Definitions 7 and 14).

Analysis Method

Analysis of a PNwC is based on the exploration of the symbolic execution of the system being analyzed. This execution is studied using the PN reachability graph (Definition 12). When the reachability graph is constructed, each graph node represents a symbolic state. The previous definitions are necessary for formalizing the process model that employs PNwC. The analysis algorithm allows model checking to detect errors in the structure as well as modeling of the time variable. In contrast to the algorithm described in (Henzinger, Nicollin, Sifakis & Yovine, 1994), this algorithm allows the analysis of concurrent and asynchronous processes as well as the time variable.

WPD AND THE PNwC

This section shows the relation between Meta-Model entities and the PNwC. As expressed previously, the activities are single logical steps. An activity represents work that will be processed by a combination of resource and/or computer applications. Also, an activity may be atomic and, in this case, is the smallest unit of self contained work that is specified within the process.

The motivations behind this work are the validation of the WPD about to the time variable. By means of the validation that is made with the PNwC, the simulation tests are avoided, keeping the usual benefits. In some cases, the networks are analyzed by means of simulations rather than by the generation of the state space. This does not guarantee that the states with very low probability happen in the long run. The analysis of these states with low probability can be an object of a serious analysis. The systematic analysis of the state space takes all the events into account, even those improbable ones.

Semantic Interpretation of Workflow Activities Using PNwC

This section shows the relation between the activities in WPD using PNwC. This allows that the activities definition may be translated to PNwC. This PNwC can be validated avoiding the simulation phase. In the following, each one of the structures of activities, with their attributes, is related to a PNwC.

The Workflow Activity Definition is used to define each elementary activity that make up a workflow process. The attributes are defined with the necessary information to correct the relationships with the other activities.

The place notation in a PNwC is the following, **Pi**, place name and $\mathbf{Inv}_i$ (Definitions 3, 6 and 9), invariant associated to **Pi**. A Transition, **Tj**, is the transition name, C_j (Definitions 4 and 6), the condition associated to **Cj,** and $\mathbf{A}_j$ (Definitions 6 and 7) is the affectation, reset clocks, associated to **Cj**.

The general attribute of an activity is LIMIT, an integer that expresses the expected duration for time management purposes in DURATION_UNIT. In a PNwC it is expressed by the condition associated to the activity shown in Figure 1. This attribute is local to the activity and may be expressed by a clock, but the potential of PNwC allows expression of more complex conditions and invariants. These expressions can check the activity limit and other temporal specifications by means of Boolean expressions (Definitions 3 and 4)**.**

The incoming and outgoing transitions are used in the same way for all the activities as shown in Figure 3. The incoming transitions — JOIN elements — are described by restriction transitions like AND JOIN or XOR JOIN. In contrast, the outgoing transitions — SPLIT elements — are described by means of AND SPLIT and XOR SPLIT restrictions. In PNwC both are represented by a place, which join all conditions, temporal restrictions, associated to the transitions by means of the invariant and Boolean combination.

The Generic Activity in Figure 3, I1 and I2 is represented by the transition T1, and the limit is expressed by the condition C_1. If there exists any new measurement, it can be initiated by the affectation A_1.

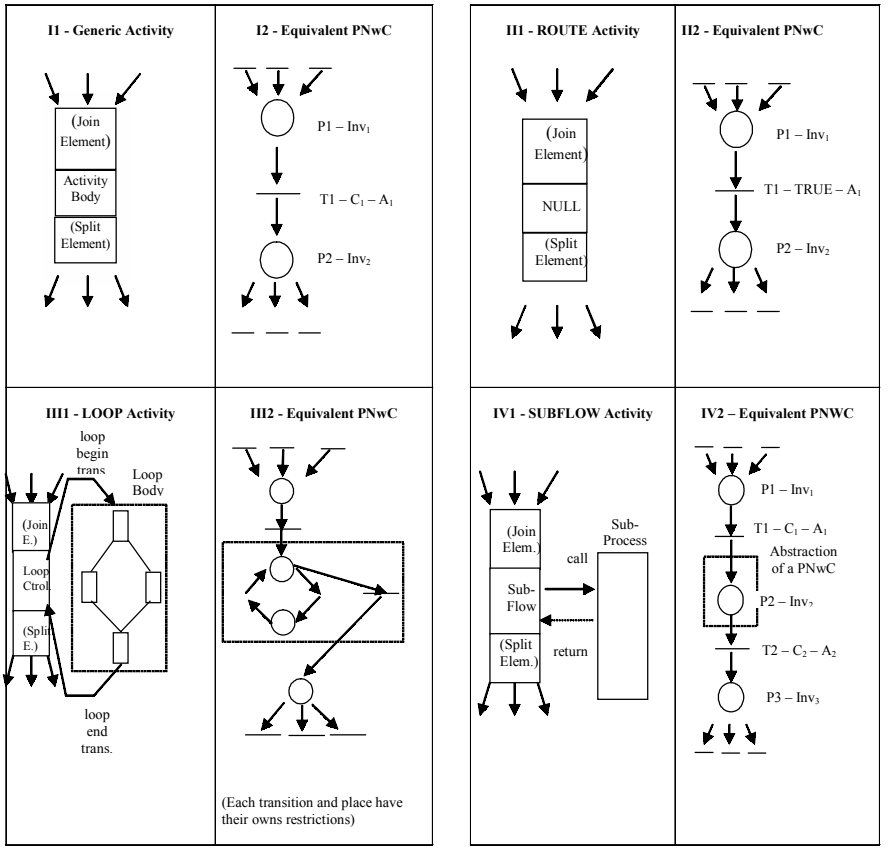
The ROUTE Activity in Figure 3, II1 and II2 is a "dummy" activity that permits the expression of cascading transition condition; this activity does not consume time. For this activity, the corresponding transition in the PNwC representation is TRUE. This transition always fires (Definitions 13).

The Loop Activity in Figure 3, III1 and III2 is refined as a loop and controls the execution of the loop repetition and loop body. In PNwC the transition T1 is the loop's beginning transition and, in P2, Inv_2 evaluates the loop expression and loop control. If it's true, execute the loop body. They are represented by P2, T2, P3, and T3. T4 represents the loop-end transition.

The SUBFLOW Activity in Figure 3, IV1 and IV2 is refined as a subprocess. This subprocess may be executed synchronously or asynchronously. The nature of PN is to model asynchronous and concurrent systems. The SUBFLOW Activity is represented perfectly by a subnet or by a place, P2, representing the abstraction of this process.

Using these equivalencies, once the model is translated to a PNwC, it may be validated by the algorithm presented in Riesco et al. (1999). The algorithm is oriented to the verification and correction of errors in the modeling of the time variable. The

Figure 3. Semantic of WF Activities Using PNwC



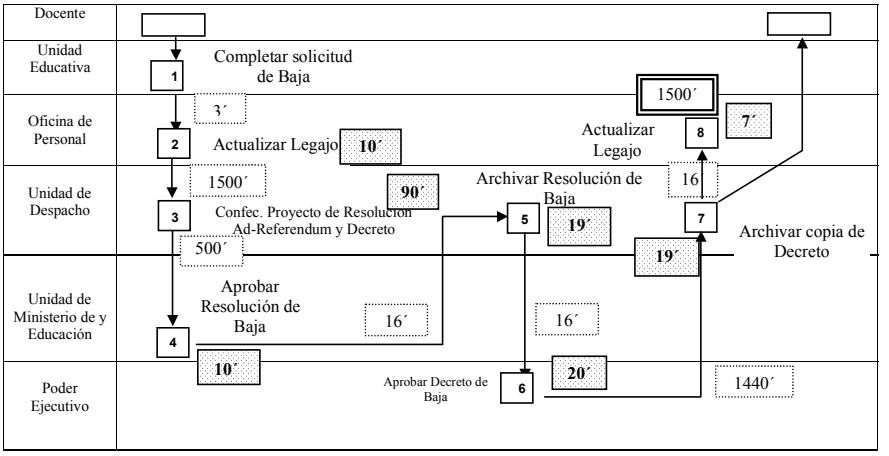
analysis is based on the exploration of the symbolic execution (Definitions 13, 15 and 16) of the system. This generates information about temporal unreachable states and processes with temporal blocks. Also it suggests corrections for place invariants and transition conditions.

EXAMPLE

Figure 4 shows a process to make the loss by resignation, death or retirement or hours chairs of an educationor (Process Manual, 1997). This process was implemented in the Ministry of Education of the San Juan province, Argentina. This process must be not exceed eight working days. In the process, different administrative units take part.

The tasks involved in the process are: Specific time of work, [i´]; and Time of transfer average [j´]. These numbers represent the time, expressed in minutes, for specific work.

Figure 4. Process for the Loss by Resignation, Death or Retirement or Hours Chairs of an Educational



In the following diagram, we express places with a circle. Associated to this, the text **Pi**, place name, simply means **P**, *P*lace, and **i** is the number of tasks in the original diagram. Next to the place name is expressed the **Invariant**, an expression, associated to this place /* def 3, 6 */.

The transitions have the names, the condition, (Definitions 4 and 6) and the affectation (Definitions 6 and 7).

Once the translation is made (see Figure 5), the validation algorithm is executed on the PNwC. This will give the information about the model.

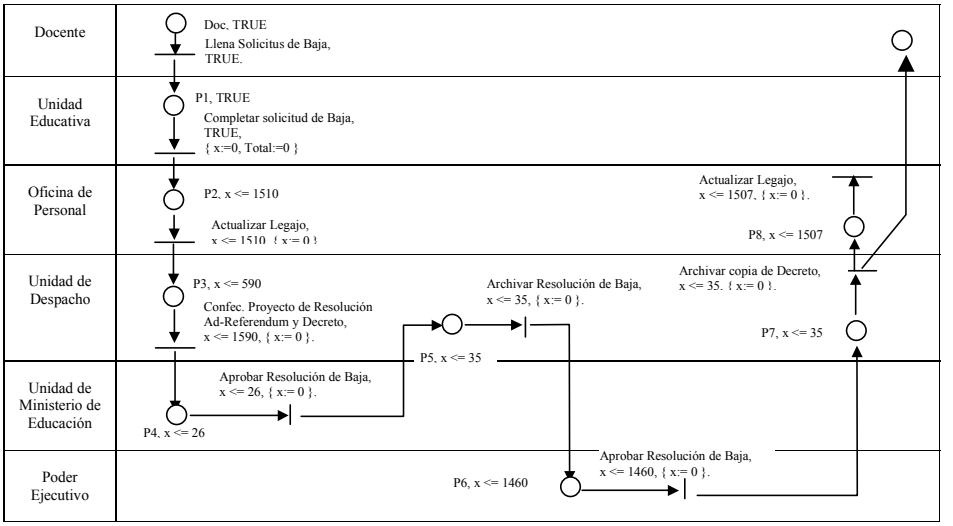
CONCLUSIONS

We have presented a technique to translate WPD to a PNwC. We believe this is very important for the verification of the system integrity, regarding its structure and its temporal specifications. Deadlock detection and temporal blockings, as well as the consistency of the restrictions, have been proposed in previous work. PNwC has highly expressive power in the concurrent and asynchronous process modeling and has the possibility to model the temporal facets of the WPD. PNwC includes additional temporal elements and clocks, which are not taken into consideration in the literature, concerning the extensions of PN with time.

Modeling and analysis of business organizations is required for the purpose of redesigning an enterprise's business process to make the organization more effective (Business Process Reengineering), as well as to establish advanced coordination technology. We presented an abstract frame of WPD translation to a PNwC.

Our work proposes the application of the PNwC for the WPD, covering the deficit of modeling and the validation of the time variable in the activities and

Figure 5. Process for the Loss by Resignation Translated to PNwC



transitions avoiding the simulation phase. This work allows modeling business with the variable time. This technique allows the qualitative analysis of WPD by means of PNwC. This technique was successfully applied in the process reengineering in the Ministry of Education of the San Juan province in Argentina.

REFERENCES

Allen, R. (2000). *The Workflow Handbook 2001*. Workflow Management Coalition (WfMC). Retrieved from http://www.wfmc.org.

Alur, R., & Courcoubetis, A. D., & Dill, D. L. (1990). Model checking for real-time systems. *Proceedings of the Fifth Symposium on Logic in Computer Science. IEEE C.S.P.* (pp. 414-425) Los Alamitos, CA.

Buy, U. & Sloan, D. (1994, August). Analysis of real-time programs with simple time. *In Proceedings of the 1994 International Symposium on Software Testing and Analysis* (pp. 228-239) Seattle, WA: ACM.

Daws, C., Olivero, A., Tripakis, S., & Yovine, S. (1996). The tool Kronos. *In Hybrid Systems III: Verification and Control.* (LCNS 1066, pp. 208-219) Springer Verlag.

Education Ministry of San Juan Province. (1997). *Diligenciar baja por renuncia, fallecmiento o jubilacion* (Process Manual Versión No. 2.1) San Juan, Argentina.

Ghezzi, C., Mandrioli, D., Morascas, S., & Pezze, M. (1989). A general way to put time in Petri Nets. *Proceeding of the 5th International Workshop on*

Software Specification and Design, (pp. 60-67) *Pittsburgh, PA:* IEEE Computer Society Press.

Ghezzi, C., Mandrioli, D., Morascas, S., & Pezze, M. (1991). A unified high-level Petri Net model for time critical system. *IEEE Transactions on Software Ingeniering 17*(2), 160-172.

Henzinger, T., Nicollin, X., Sifakis, J., & Yovine S. (1994). Symbolic model checking for real-time systems. *Information and Computation, 111*(2), 193-244.

Jacobson, I., Ericsson, M., & Jacobson A. (1995). *The object advantage. Business process reengineering with object technology.* Reading, MA: Addison Wesley.

Montejano, G., Riesco, D., Vilallonga, G., Dasso, A., & Favre, L. (1998). An analysis algorithm for timed Petri Nets. *Software Engineering* (SE'98) (pp. 271-274). IASTED. Las Vegas, ACTA Press.

Olivero, A. (1994). *Modélisation et analyse de systémes temporisés et hybrides.* Unpublished thesis Institut National Polytechnique de Grenoble, France.

Peterson, J. (1981). *Petri Net theory and the modelling of systems.* Englewood Cliffs, NJ: Prentice Hall.

Riesco, D., Montejano, G., Vilallonga, G., Dasso, A., & Uzal, R. (1999). Underlying formalism for a timed Petri Net algorithm. *Proceedings of the IASTED International Conference Software Engineering and Applications* (pp. 348-352) Scottsdale, AZ: ACTA Press.

Sifakis, J., & Yovine, S. (1996). Compositional specifications of timed systems. *In 13th Annual Symposium on Theoretical Aspects of Computer Science, STACS '96* (February, pp. 347-359) Grenoble, France. Lecture notes in Computer Science 1046, Springer-Verlag.

Workflow Management Coalition. (1999a, February). *Terminology & glossary* (Doc. No. WFMC-TC-1011, Version 3.0).

Workflow Management Coalition. (1999b, October). *Interface 1: Process definition interchange process model.* (Doc. No. WfMC TC-1016-P Version 1.1). Retrieved from http://www.wfmc.org.

Yovine, S. (1993). Méthodes et outils pour la vérification symbolique de systémes temporizes. Unpublished doctoral dissertation Institut National Polytechnique de Grenoble, France.

Zuberek, M. (1991). Timed Petri Nets: Definitions, properties, and applications. *Microelectronics and Reliability, 31*(4), 627-644.

Chapter III

Software and Systems Engineering: Conflict and Consensus

Rick Gibson
American University, USA

ABSTRACT

This chapter will identify the key aspects of software engineering and systems engineering in an effort to highlight areas of consensus and conflict to support current efforts by practitioners and academics in both disciplines in redefining their professions and bodies of knowledge.

By using the Software Engineering Institute's Capability Maturity Model – Integrated (CMMI[SM]) project, which combines best practices from the systems and software engineering disciplines, it can be shown that significant point of agreement and consensus are evident. Nevertheless, valid objections to such integration remain as areas of conflict. This chapter will provide an opportunity for these two communities to resolve unnecessary differences in terminology and methodologies that are reflected in their different perspectives and entrenched in their organizational cultures.

INTRODUCTION

With software an increasingly significant component of most products, it is vital that teams of software and systems engineers collaborate effectively to build cost-effective and reliable products. This chapter will identify the key aspects of software engineering and systems engineering in an effort to highlight areas of consensus and conflict to support current efforts by practitioners and academics in the both disciplines in redefining their professions and bodies of knowledge.

BACKGROUND

In response to increasing concerns about software development failures, the Software Engineering Institute (SEI) pioneered a software process improvement model in 1988, with the fully developed version of the Capability Maturity Model for Software (SW- CMMâ) appearing in 1993. The key processes were identified as (Paulk, Curtis, Chrissis & Weber, 1993): Requirements Management; Integrated Software Management; Software Project Planning; Software Product Engineering; Software Project Tracking and Oversight; Intergroup Coordination; Software Subcontract Management; Peer Reviews; Software Quality Assurance; Quantitative Process Management; Software Configuration Management; Software Quality Management; Organization Process Focus; Defect Prevention; Organization Process Definition; Technology Change Management; and Training Program.

Since the early 1990s, there have been comparable improvement models introduced in the system engineering community as well, some of which have been published and widely accepted: Systems Engineering Capability Maturity Model (SE-CMM) also known as the Electronic Industries Alliance Interim Standard (EIA/IS) 731, Systems Engineering Capability Model (SECM) and the Integrated Product Development Capability Maturity Model (IPD-CMM). In 1995, SEI recognized 18 process areas in systems engineering. The processes fell into three categories: engineering process, project process and organizational process (Bates Kuhn, Wells, Armitage & Clark, 1995). The resulting avalanche of models and standards has been described by Sarah Sheard (Software Productivity Consortium) as a "Framework Quagmire." In December of 2000, the SEI initiated CMMI[SM] project, which combines best practices from the systems and software engineering disciplines. (Note: CMM® and CMMI[SM] are copyrights and service marks of the Software Engineering Institute.)

ISSUES AND CONTROVERSIES

There is great hope that the SEI initiative will provide the impetus to overcome some long-standing discipline boundaries. The nature of the systems and software

engineering work has led to terminology differences rooted in the very descriptions of the disciplines.

Issues and concerns regarding such integration were articulated by Barry Boehm and Fred Brooks as early as 1975. Boehm suggested that the adoption of systems engineering reliability techniques by software engineers was counterproductive. Moreover, Brooks' Law suggests that a common system engineering solution to schedule slippage (add more people) only makes a late software project even later.

More recently, Boehm (1994) expressed concerns that, in spite of the central function of software in modern systems, the two engineering disciplines have not been well integrated. Boehm articulated the similarities and differences shown in Table 1.

Software engineering as defined by the Institute of Electrical and Electronics Engineers (IEEE Computer Society/Association for Computing Machinery, 2001) is: (1) the application of a systematic, disciplined and quantifiable approach to the development, operation and maintenance of software that is the application of engineering to software; (2) the study of approaches as in (1) and further identifies the body of knowledge for software engineering to be: software requirements, software design, software construction, software testing, software maintenance, software configuration management, software engineering management, software engineering process, software quality and software engineering tools and methods.

Table 1. Software and System Engineering Similarities and Differences

Similarities	Differences
Definition and analysis involves manipulation of symbols	Software is not subject to physical wear or fatigue
Highly complex aggregation of functions, requiring satisfying (though not optimizing) among multiple criteria	Copies of software are less subject to imperfections or variations
Decisions driven by need to satisfy quality attributes such as reliability, safety, security, and maintainability	Software is not constrained by the laws of physics
Easy and dangerous to suboptimize solutions around individual subsystem functions or quality attributes	Software interfaces are conceptual, rather than physical — making them more difficult to visualize
Increasing levels of complexity and interdependency	Relative to hardware, software testing involves a larger number of distinct logic paths and entities to check
	Unlike hardware, software errors arrive without notice or a period of graceful degradation
	Hardware repair restores a system to its previous condition; repair of a software fault generally does not
	Hardware engineering involves tooling, manufacturing, and longer lead times, while software involves rapid prototyping and fewer repeatable processes

A useful definition of systems engineering resides in an in process body-of-knowledge document by the International Council on Systems Engineers (Leibrandt, 2001) that defines systems engineering in terms of product and process: "…product oriented engineering discipline whose responsibility is to create and execute an interdisciplinary process to ensure that customer and stakeholder needs are satisfied in a high quality, trustworthy, cost effective and schedule compliant manner throughout a system's lifecycle." The process starts with customer needs and consists of stating the problem, investigating alternatives, modeling, integrating, launching the system and assessing performance. Moreover, the system engineer is responsible for pulling together all the disciplines to create a project team to meet customers' needs. The complete systems engineering process includes performance, testing, manufacturing, costing, scheduling, training and support, and disposal. The body of knowledge recognizes that systems engineering processes often appear to overlap software and hardware development processes and project management. Thus, systems engineering is a discipline that focuses on processes, developing structure and efficient approaches to analysis and design to solve complex engineering problems. In response to concerns about integrated development of products, the system engineer plans and organizes technical projects and analyzes requirements, problems, alternatives, solutions and risks. Systems engineering processes are not specific to a particular discipline; they can be applied in any technical or engineering environment.

In short, software engineering is defined by IEEE Standard 610.12 as the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software — that is, the application of engineering to software. The International Council on Systems Engineering (INCOSE) defines systems engineering as an interdisciplinary approach and means to enable the realization of successful systems. The following are definitions of the two disciplines from the CMMI[SM]:

Systems Engineering — The systems engineering discipline covers the development of total systems, which may or may not include software. Systems engineers focus on transforming customer needs, expectations and constraints into product solutions and supporting those product solutions throughout the product life cycle.

Software Engineering —The software engineering discipline covers the development of software systems. Software engineers focus on applying systematic, disciplined and quantifiable approaches to the development, operation and maintenance of software.

When different process models are in place within developer groups, such as for systems engineering and software engineering of an organization, the organizations will have communication problems, be unable to improve their processes and the combined performance another advances beyond the other in capability, then the problems are even more profound (Johnson & Dindo, 1998).

In 2002, the SEI released a single integrated capability model for systems engineering and software engineering, integrated product and process development, and supplier sourcing. The new model, CMMI[SM], is intended to improve organizations' development and maintenance of products. The CMMI[SM] will eventually replace the SEI's Software Capability Maturity Model (Phillips, 2002). In the integrated model (Software Engineering Institute, 2002), CMMI[SM], the categories and processes are:

Process Management

- Organizational Process Focus
- Organizational Process Definition
- Organizational Training
- Process Performance
- Organizational Innovation and Deployment

Project Management

- Project Planning
- Project Monitoring and Control
  Supplier Agreement Management Organizational
- Integrated Product Management
- Risk Management
- Qualitative Project Management
  Integrated Teaming

Engineering

- Requirements Management
- Requirements Development
- Technical Solution
- Product Integration
- Verification
- Validation

Support

- Configuration Management
- Process and Product Quality Assurance
- Measurement and Analysis
- Causal Analysis and Resolution
- Decision Analysis and Resolution
- Organizational Environment for Integration

One purpose of the CMMI[SM] was to evolve the software CMM®, while integrating the best features of the systems engineering capability models. The combination of the practices of the models into one single framework required more than just combining practices because of differences in interpretation, focus and terminology. Compromises and intentional inefficiencies were required in order to integrate these models. For example, the CMMI[SM] was released with two representations, continuous and staged (Table 2), to allow systems and software groups, respectively, to continue using the model representation with which they were already familiar.

With the CMMI[SM] there is significant coverage provided for the engineering dimension, more detailed coverage of risk management and measurement and enhanced analysis that was less specific in the Software CMM. Moreover, the CMMI[SM] (continuous representation) process areas are grouped by the categories: Process Management, Project Management, Support and Engineering. The Engineering category includes the process areas shown in Table 3, which are intended to integrate software and systems engineering by targeting product-oriented business objectives.

RECOMMENDATIONS BASED ON INTEGRATED PROCESS MODEL BENEFITS

Rassa (2001) summarizes the benefits of the CMMI[SM] project as follows:

- Common, integrated vision of improvement for all organizational elements;
- Means of representing new discipline-specific information in a standard, proven process improvement context;

Table 2. Continuous and Staged Representations

Systems Engineering — Continuous	Software Engineering — Staged
Migration path from EIA/IS 731	Migration path from SW-CMM
Encourages a focus on process areas to address business objectives — avoids the maturity plateau trap.	Encourages a proven sequence of improvements beginning with basic management practices and progressing through successive levels.
Encourages comparisons across organizations by process areas.	Encourages comparisons among organizations using maturity levels.
Provides increased visibility into capability achieved within a process area. Can measure below Level 2.	Cases studies and empirical data show return on investment for process improvement.
Provides a focus on risks specific to each individual process areas.	Summarizes process improvement results in a single maturity level number.
Encourages the generic practices from higher capability levels be more evenly and completely applied to all process areas.	

Table 3. Engineering Process Areas

Process Areas	Purpose
Requirements Management	Manage the requirements of the project's products and product components and to identify inconsistencies between those requirements and the project's plans and work products.
Requirements Development	Produce and analyze customer, product, and product components.
Technical Solution	Develop, design, and implement solutions to requirements.
Product Integration	Assemble the product from the product components, ensure that the product, as integrated, functions properly, and deliver the product.
Verification	Assure that selected work products meet their specified requirements.
Validation	Demonstrate that a product or a product component fulfills its intended use when placed in its intended environment.

- Efficient, effective assessment and improvement across an organization's multiple process disciplines;
- Reduced training and assessment costs.

According to the SEI, "The CMMI[SM] effort is intended to support process and product improvement and to reduce redundancy and eliminate inconsistency when using separate stand-alone models. The goal is to improve efficiency, return on investment and effectiveness by using models that integrate disciplines such as systems engineering and software engineering that are inseparable in a systems development endeavor" (Software Engineering Institute, 2001). With the arrival of the CMMI[SM], a wider continuum of the product life cycle has been targeted for possible enhancement, no longer limiting process improvement only to the development of software. This integrated approach provides a reduction in the redundancy and intricacy resulting from the use of multiple, separate process improvement models. For organizations that wish to assess their process improvement efforts against multiple disciplines, the CMMI[SM] provides some economies of scale in model training and assessment training. This one evaluation method can provide separate or combined results for the fields of software and system engineering. Furthermore, software organizations can also focus on the amplifications for software engineering within the engineering shared process areas and take advantage of any systems engineering amplifications that are helpful. Although still subject to debate, a distinction is made between base and advanced engineering practices as model constructs. The CMMI[SM] provides the groundwork for enterprise-wide process

improvement with a new emphasis on products and services as well as process. This emphasis is on both organizational maturity and process capability because the CMMI[SM] directs increased attention to measurement and analysis.

EVIDENCE OF SIMILARITIES: CMMI[SM] GENERIC PRACTICES

As explained by Ahren, Turner and Clouse. (2001), the CMMI[SM] draws a distinction between model components that are **required** for process improvement (i.e., satisfied goals) and components that are **expected** to play an essential role as indicators that the required components are in place and institutionalized as common features of the organization's culture. A **practice** is a statement of an expected component; and it may be unique to a single process area (specific practice) or may apply across all process areas (generic practice). In short, generic practices (see Table 4) imply a bridge across the disciplines of software and systems engineering.

THE NEED TO AVOID THE RATING GAME

In many ways, the philosophy of process maturity levels is much like Maslow's hierarchy, which suggests that before one can address higher-level needs like self-actualization, the needs from lower in the hierarchy, such as food and shelter, need

Table 4. Generic Practices

Generic Practice	Maturity Level	Common Feature
Establish an Organizational Policy	2	Commitment
Establish Requirements and Plan the Process	2	Ability
Provide Resources	2	Ability
Assign Responsibility	2	Ability
Train People	2	Ability
Manage Configurations	2	Directing Implementation
Identify and Involve Relevant Stakeholders	2	Directing Implementation
Monitor and Control the Process	2	Directing Implementation
Objectively Evaluate Adherence	2	Verifying Implementation
Review Status with Higher-Level Management	2	Verifying Implementation
Establish a Defined Process	3	Ability
Collect Improvement Information	3	Directing Implementation
Establish Quality Objectives	4	
Stabilize Subprocess Performance	4	
Ensure Continuous Process Performance	5	
Correct Common Cause of Problems	5	

to be met. The CMMI[SM]-staged approach provides organizations with a very structured approach to becoming a more mature institution. The definition of these maturity levels provides organizations with milestones of achievement. It also allows organizations to establish where they are in the software process improvement continuum. For the past decade, the Software CMM® levels provided a structure and gave organizations milestones in the process of becoming more mature, but also had the unintended effect of creating a competitive rating scale between software organizations. An assessed process may be adequate in one environment but may not suffice for a new project in a different environment (Paulk, Weber & Chrissis, 1999). Organizations that use maturity levels to assess contractors run the risk of neglecting many of other factors that would help determine the most appropriate contractor to work on a specific project.

The existence of maturity levels also introduces the risk of organizations setting maturity level goals instead of focusing on improving software/systems to address business goals. With all of the claims of return on investment surrounding process improvement efforts, it is easy to understand why management may strive to reach specific maturity levels for all the wrong reasons. A communication of this concept must be made to all stakeholders who would be affected by the implementation of process improvement. The implementation of a process improvement program needs to be a part of the means to achieve the business goals. Businesses can easily become preoccupied in reaching a specific maturity level and forget the ends that they are trying to accomplish. As a result, organizations may end up taking shortcuts in order to be assessed sooner at a certain level even though more attention to a specific process would have been beneficial in reaching a specific business goal.

In contrast, the systems engineering community adopted an alternative approach to visualizing process improvement: A continuous representation based on individual process areas. The continuous representation evaluates organizations based on capability levels instead of maturity levels. The main difference is that capability levels apply to an organization's process-improvement achievement in individual process areas. These capability levels are 0 (not performed) to 5 (optimizing). Maturity levels apply to an organization's overall process-improvement achievement using the staged model. Using the continuous representation, an organization would have a capability profile, consisting of a list of process areas and their corresponding capability levels. One of the clear benefits of continuous representation is that it provides organizations with the ability to select the order and grouping of improvement areas that best complement their business objectives (Shrum, 1999).

Adopting the continuous representation of CMMI[SM] not only forces software organizations to define business goals and choose process areas that should be implemented first to focus on these goals, but it also forces companies who are choosing a new subcontractor to do the same. One of the claimed benefits of a staged representation is that it facilitates comparisons among organizations (Shrum, 1999).

While it may simplify comparisons, it does so at the loss of additional details. By using a continuous representation, the comparison can be done based on the process areas that are judged important by the organization rather than simply comparing the organization's maturity score. When using a continuous representation, there is less likelihood that organizations will try to attain a specific maturity level without reasons to do so. It provides an incentive to address processes that would have the greatest impact on their business goals.

Since the continuous representation of the CMMI[SM] implicitly encourages organizations to base process improvements on defined business goals, these organizations are less likely to ignore possible improvements outside the scope of the CMMI[SM] model. If the CMMI[SM] model does not provide them with the means to work toward succeeding on a specific business model, organizations will need to find alternative process-improvement methods. Importantly, there are several areas that are completely ignored within the CMMI[SM]. Examples include strategic-level processes, such as business strategic planning, architecture definition and strategic planning and control. This group of processes focuses on the adjustments needed over time to meet the changing conditions and requirements of the environment (Purvis, Santiago & Sambamurthy, 1999). Since strategic level processes may have a greater impact on the core business goals than improved quality, it may be more beneficial for an organization to focus of these areas before moving forward with CMMI[SM] processes.

THE NEED TO OVERCOME THE DIFFERENCES

Despite anticipated problems bringing systems engineering best practices into the established software process, improvement models are expected to be very beneficial. Boehm (1994) reminds us that an important reason to overcome or bridge these differences is to establish an adequate supply of people who can deal with complex systems problems. The Bureau of Labor Statistics' (1997) estimates of

Table 5. Anticipated Employment Growth 1996-2006

Type of Job	1996 Employment	2006 Employment	% Change
Database Administrators and Computer Support	212,000	461,000	118%
Computer Engineers	216,000	451,000	109%
Systems Analysts	506,000	1,025,000	103%
Data Processing Equipment Repair	80,000	121,000	52%
Engineering, Science, and Computer Systems Managers	343,000	498,000	45%

anticipated growth in information technology jobs, shown in Table 5, provides further support for this concern.

The final job type, managers, is a significant concern addressed by Jerry Weinberg in an interview (Layman, 2001). Weinberg explains that the software development problems are growing faster than individuals' levels of competence. Moreover, he asserts that the current state of practice is one where we need to apply a few fundamentals (e.g., requirements, reviews, configuration management), that is things known to be useful, but not adopted in the sense of consistent application.

It has been suggested that the systems engineering — hardware engineering interfaces have matured nicely over many years, but that the systems engineering — software engineering interface is not as mature.

Meanwhile, the dependency on the systems engineering-software engineering interface has increased faster than it has matured. Concerns on the state of systems engineering include:

- Most successful projects rely on expertise established with similar systems.
- Lack of documented processes makes repeatability difficult.
- Development efforts for unprecedented or significantly different systems often encounter problems.

Concerns on the state of systems engineering include:

- The brief history of software development has been filled with problems of cost overruns, schedule slippage and failure to achieve performance goals.
- Systems are increasingly dependent on software, yet hardware typically gets the most visibility.

Although, many software-only organizations remain adamant that they do not do systems engineering, all software must run on a computer system and interface with others. This perceived separation of concerns exacerbates the difficulties associated with hardware/software/system tradeoff decisions, which are further complicated by terminology differences and disparate mental models.

However the integration potential of the CMMI[SM] can allow the system and software engineering communities to get the most out of their similarities. The CMMI[SM] allows organizations to tailor the model to mesh with their own mission and goal statements as well as their business objectives. Each individual project can use CMMI[SM] models for individual disciplines and discipline combinations because the architecture of the CMMI[SM] does not force the employment of every discipline. Before the CMMI[SM], the systems engineering models shared many of the same principles as the software version of CMM®, but were written to address the needs and terminology of the systems engineering community. Because the CMMI[SM] includes the common and shared elements and best features of both software and system engineering together with discipline specific elements, an organization can generate integrated capability maturity models or discipline specific capability

models. With CMMI[SM] an organization can still capitalize on these similarities and improve the efficiency of and the return on investment for process improvement. The resulting integrated capability models will adapt to an organization's business purposes.

FUTURE TRENDS IN ENGINEERING EDUCATION

Software engineering education is receiving increased scrutiny. Parnas (1999) asserts that the current educational system for software professionals fails to satisfy employers. A recent survey (Lethbridge, 2000) found that some of the topics that surveyed professionals thought were most important were not taught at most universities. He found knowledge-education gaps in vital software engineering process areas such as configuration management, release management and human-computer interfaces. The survey also found that respondents needed to acquire software engineering skills such as testing, quality assurance, verification and project management in the workplace.

The ranked list of knowledge gaps from the survey were:

1. Negotiation
2. Human-computer interface
3. Leadership
4. Real-time systems design
5. Management
6. Software cost estimation
7. Software metrics
8. Software reliability and fault tolerance
9. Ethics and professionalism
10. Requirements gathering and analysis.

Parnas (1999) advocated that software engineering be placed in the engineering departments of universities so that students remain focused on the fact that the goal of software engineers, like other engineering disciplines, is to apply knowledge to build *products*. A different solution comes from Meyer (2001), who recommends a software curriculum that covers principles, practices, applications, tools and mathematics. By practices, he refers to configuration management, project management, metrics, ergonomics and user interfaces, documentation, user interaction, and high level systems analysis and debugging. Denning (2001) developed a framework for IT schools designed to meet a professional performance level for an entry-level programmer. He claims that few undergraduate programs prepare students at this level. His curriculum calls for team projects every spring semester. Entry-level professional skills include ability to design systems of hundreds of modules, programming, testing and documentation.

Carnegie Mellon's Working Group on Software Engineering Education and Training has developed guidelines for software engineering education (Bagert, Hilburn, Hilsop, Lutz, McCracken & Mengel, 1999). The guidelines describe the body of knowledge for software engineering education and contain model curricula. Core topics are requirements, design, construction, project management and software evolution (maintenance, extensibility and reengineering). Recurring topics in each course are professionalism, process, quality, modeling, metrics, tools and environment, and documentation.

A joint endeavor of Association for Computing Machinery (ACM) and IEEE-CS published their accreditation and model curricula for software engineering education in 1998. According to an IEEE-ACM task force on computer science curriculum (IEEE/ACM, 2001), an undergraduate degree in computer science should require 31 hours in the core knowledge areas of software engineering: software design, process, requirements and specifications, validation and software evolution. Per the task force (IEEE/ACM, 2001), advanced courses are: Advanced Software Development, Software Engineering, Software Engineering and Formal Specification, Empirical Software Engineering, Software Process Improvement, Component Based Computing, Programming Environments and Safety Critical Systems.

The Software Engineering Institute (Bagert et al., 1999) recommends that a software engineering curriculum have five content areas: computer science fundamentals, mathematics, natural sciences, software engineering and general engineering. In addition they identify modules of software engineering that should be presented, in separate courses or included in others: introduction to computer science for software engineers (i and ii), professionalism and ethics, software requirements, software design, software quality, software construction and evolution (issues, methods and techniques), and software project design.

Systems Engineering Education

Similarly, systems engineering education is being evaluated. Asbjornsen and Hamann (2000) explored four concepts in systems engineering education, including creating a new department or discipline, offering graduate education to degreed engineers, practical on-the-job training and integrated engineering. They recommend that systems engineering be integrated into all educational programs for all types of engineering. This matches the recommended solution for software engineering educational reform.

A review of systems engineering curricula suggests that systems engineering is a management process; systems engineers unite and manage technical people from many different disciplines to create systems, thus courses relating to project management are required.

On the other hand, software engineering is a specific engineering domain. All of the software engineering programs, whether undergraduate or advanced degrees,

limited their required coursework to software engineering and computer science as specified by organizations such as SEI, IEEE and ACM.

The two domains are different and the curricula reflect the differences. Software engineering is about the systematic approach to development, operation, and maintenance of software. Systems engineering is about integrating engineering disciplines. It appears from the curriculum comparisons that a software engineering major would be less prepared to be a part of multi-disciplinary team than the graduate from a systems engineering program.

CONCLUSION

The ongoing process improvement efforts centered on integration of software and systems engineering, as initiated by the SEI's CMMI[SM] project, has highlighted two key issues for researchers and practitioners:

1. A renewed focus on products and business objectives as drivers of process improvement;
2. Opportunities for high-leverage process improvements.

The concept of architecture continues to serve as a theoretical link for both the software-system tradeoffs and the integration of process improvement efforts. While respecting the legitimate differences in areas, such as reliability testing, it is important to sustain the hope that overlapping or underlying theories will emerge regarding areas of common concern such as: requirements, security, safety and performance.

In order to achieve true integration of software and system engineering practices into one process improvement model, the remaining differences of terminology and model construction have to be addressed. These two communities have well-developed disparate languages and methodologies that are reflected in their different origins, models and perspectives, differences that have become entrenched in their organizational cultures. With the adoption of an integrated process improvement model, an organization can assess both software and systems engineering functions, reduce conflict and increase consensus.

REFERENCES

Ahern, D., Turner, R. & Clouse, A. (2001). *CMMI(SM) distilled: A practical introduction to integrated process improvement.* Boston, MA: Addison-Wesley.

Asbjornsen, O.A., & Hamann, R. (2000). Towards a unified systems engineering education. *IEEE transactions on systems, man, and cybernetics — Part C: Applications and reviews,* 30(2), 223-243.

Bagert, D., Hilburn, T., Hilsop, G., Lutz, M., McCracken, M. & Mengel, S. (1999, October). *Guidelines for software engineering education, Version 1*. (Tech. Rep. CMU/SEI99-TR-032). Retrieved October 12, 2002 from the World Wide Web: http://www.sei.cmu.edu/pub/documents/99.reports/pdf/99tr032.pdf.

Bates, R., Kuhn, D., Wells, C., Armitage, J., & Clark, G. (1995). *A systems engineering capability maturity model, Version 1*. [Handbook SECMM-95-01]. Carnegie Mellon University.

Boehm, B. (1994). Integrating software engineering and system engineering. *The Journal of INCOSE,* I(I).

Davies, J.K. (2001). Systems engineering: the ubiquitous unicorn. *Proceedings of the INCOSE* UK Symposium.

Denning, P. (2001). The IT schools movement. *Communications of the ACM, 44*(8), 19-22.

IEEE Computer Society/Association for Computing Machinery (IEEE/ACM). (2001). Computing curricula 2001. (Computer Science Final Report). Joint Task Force on Computing Curricula.

International Council on Systems Engineering (INCOSE). Retrieved February 2002 from the World Wide Web: http://www.incose.org/.

Johnson, K.A., & Dindo, J. (1998, October). Expanding the focus of software process improvement to include systems engineering. *CROSSTALK The Journal of Defense Software Engineering,* 13-19.

Layman, B. (2001). An Interview With Jerry Weinberg. *Software Quality Professional, 3*(4), 6-11.

Leibrandt, R. (2001, April 22). *A guide to the systems engineering body of knowledge (SEBoK).* Retrieved February 2002 from the International Council on Systems Engineering (INCOSE) Web site: http://www.incose.org/orlando/sebok/attach/sebok_text.doc.

Lethbridge, T.C. (2000). What knowledge is important to a software professional? *IEEE Computer, 33*(5), 44-50.

Meyer, B. (2001, May). Software engineering in the academy. *IEEE Computer,* 28-35.

Parnas, D. (1999, May). Software engineering programs are not computer science programs. *IEEE Software,* 19-30.

Paulk, M., Curtis, B., Chrissis, M., & Weber, C. (1993, February). The capability maturity model for software, Version 1.1. (CMM/SEI-93-TR-24 DTIC ADA 263403). Pittsburgh, PA: Software Engineering Institute.

Paulk, M.C., Weber, C.V., & Chrissis, M.B. (1999). The capability maturity model for software. In K. El Emam & N. H. Madhavji (Eds.), *Elements of software process assessment and improvement* (pp. 3-22). Los Alamitos, CA: IEEE Computer Society.

Phillips, M. (2002, February) CMMI, Version 1.1: What has changed. *CROSSTALK The Journal of Defense Software Engineering,* 4-6.

Purvis, R.L., Santiago, J., & Sambamurthy, V. (1999). *An analysis of excluding IS processes in the Capability Maturity Model and their potential impact* (pp. 31-46). Hershey, PA: Idea Group Inc..

Rassa, B. (2002). *Beyond CMMI-SE/SW v1.0.* Retrieved March 13, 2001, from the Software Engineering Institute Web site: http://www.sei.cmu.edu/cmmi/publications/sepg01.presentations/beyond.pdf.

Shrum, S. (1999, December). *Choosing a CMMI model representation*. Retrieved July 17, 2001, from the SEI Interactive Web site: http://www.stsc.hill.af.mil/crosstalk/2000/jul/shrum.asp.

Software Engineering Institute (SEI) (2001). *CMMI frequently asked questions*. Retrieved March 2001 from the Software Engineering Institute Web site: http://www.sei.cmu.edu/cmmi/comm/cmmi-faq.html.

Software Engineering Institute (SEI) (2001, March). Capability maturity model integrated (CMMI), Version 1.1. (CMM/SEI-2002-TR-012). Pittsburgh, PA: Software Engineering Institute.

US Department of Labor, Bureau of Labor Statistics (1997). Employment and Earnings.

Chapter IV

Lean, Light, Adaptive, Agile and Appropriate Software Development: The Case for a Less Methodical Methodology

John Mendonca
Purdue University, USA

Jeff Brewer
Purdue University, USA

ABSTRACT

Historically, the approach to software engineering has been based on a search for an optimal (ideal) methodology — that is, the identification and application of a set of processes, methods and tools that can consistently and predictably lead to software development success. This chapter presents the basis for pursuing a more flexible and adaptive approach to methodology. Less methodical methodologies, under a variety of names, take a contingency-oriented approach. Because of the limitations in the nature of methodology, the high failure rate in software development, the need to develop methodology within an environmental context and the pressures of fast-paced "e-development," the authors argue that further exploration and definition of an adaptive, contingency-based approach to methodology is justified.

INTRODUCTION

Despite the high rate of failure in software development, the fundamental strategy for achieving quality in software engineering continues to be methodology — that is, discovery and application of that ideal set of processes and practices that lead to software products that are accurate, effective, delivered on time and within budget. The path to an optimal methodology leads theorists and practitioners toward increasingly refined sets of concepts, models, rules, project management strategies, descriptions of deliverables, tools, testing standards, test-case constructs and the many other components of a well-defined methodology. Perhaps because of its close identity with the "engineering" paradigm, ubiquitous failure seems not to have shaken faith in the methodical approach to software development. In fact, the response to failure seems often to be more methodology.

In recent years, due to the increasing complexity of the information technology (IT) arena and the furious pace of e-commerce and e-business development, a less methodical approach to software development management has gained attention. This approach has often been linked with Extreme Programming (XP) and has been called by a variety of names, including "lean" and "light" methodology (Yourdon, 2000b). Highsmith (2000) used the term "adaptive" in his book describing the basic concepts, but he and others prominent in XP theory and practice seem to have settled on "agile" as the preferred term. Earlier this year, with the support of XP proponents and others, the "Manifesto for Agile Software Development" (2001) was developed and published.

Regardless of the name, the approach embodies two characteristics. The first characteristic is that it is less methodical. It is not fixated on the search for an optimal methodology but is contingency oriented, allowing for adaptation and flexibility depending on environmental issues. The second characteristic is that it incorporates a concept of appropriateness. A methodology must not only adapt to its environment, it must also reflect an appropriate level of rigidity, the "just-right" level between no methodology and a heavily restrictive one that suffocates rather than informs.

This paper argues that because of the inherent limits to methodology, unrealized expectations and the fast-paced, complex and unpredictable environment, a less methodical contingency approach to software engineering is justified.

METHODOLOGY: EXPECTATIONS AND LIMITATIONS

As noted above, a software development methodology is a set of processes and techniques for the management of software development. The numerous formal documented methodologies and many more informal ones vary based on the many paradigms and variables that are part of the software development landscape. Ivaria, Hirschheim and Klein (2000/2001) suggest there are more than 1,000 information systems development methodologies and offers a schema for their characterization

and evaluation. All methodologies have the common characteristics of being a defined set of activities and are based on the concepts of quality and engineering in software development. In addition, there are two very significant aspects of the nature of methodology itself: First, it is defined, developed and verified only through experience, *after* development has occurred; and second, methodology is itself a *system*.

Pressman (1997) suggests that a methodology is composed of three parts: processes, methods and tools. Processes provide the framework for activities. At the highest level they prescribe guiding principles, the use of resources, a definition and hierarchy of sub-processes, the sequential order of activities and other constraints. Methods provide the implementation techniques within the framework of processes. Examples of methods include requirements analysis procedures, design paradigms, testing strategies and program construction procedures. Tools, such as computer-aided software engineering (CASE) and project management software, support processes and methods.

Methodology is at the core of the concept of "software engineering" (SE). IEEE (1993) defines SE as the application of a "systematic, disciplined, quantifiable approach to the development, operation and maintenance of software." The engineering paradigm is thus predicated on well-defined processes within a generally predictable environment with well-defined outcomes and roles, sequential project (phased) development and historical information for a "best practices" approach. SE is tightly coupled to quality software development via methodology, the means by which quality is achieved. Anyone who has worked with a methodology is acutely aware that it is no *guarantee* of quality. It does provide a framework for proceeding, for organizing and understanding the tasks ahead, so it is a good base from which development can proceed.

However, every methodology is limited in several significant ways. First, a useful methodology is developed only after systems have been implemented, both successfully and unsuccessfully, and successful processes and methods ("best practices") are identified. It took a large body of experimentation, knowledge and practice, for example, before the parameters of the Systems Development Life Cycle (SDLC) were well defined and stabilized. The object paradigm embodied in object-oriented (OO) analysis, design and programming had a profound impact on systems development methodology (Capper & Colgate, 1994). OO methodologies continue to undergo rigorous experimentation, testing and proof (see the CETUS links website ["18,452 Links"] for information and links to over 50 documented OO methodologies). This "lagging" characteristic of methodology is especially significant because developers working in the latest paradigms, such as Internet and middleware development, cannot rely on well-defined methodologies for structure and guidance.

Another significant way in which methodologies are limited is that a methodology is itself a system. Nicholas (2001) defines a system as "an organized or complex

whole; an assemblage of things or parts interacting in a coordinated way." All systems are affected by events in their environment either internal and under the organization's control or external and not controllable by the organization. A "one-size-fits-all" approach to software development cannot work. Certain processes are appropriate for some conditions and inappropriate for others. Processes differ because environmental factors differ — factors such as the experience of people involved, goals, project scope, time to market, technologies used, government decisions and many others. Grady Booch, one of the principle developers of the Unified Modeling Language and now a chief scientist at Rational Software Corp., argues that the rise of new adaptive methodologies is due in large part to its ability to address the social dynamics of small teams (Betts, 2002). He goes on to say that the heavier more traditional methodologies like his company's Rational Unified Method have a difficult time meeting these needs. The Rational methodology is not sufficiently adaptive to allow for these small development teams. This concept is significant because it recognizes the responsibility of developers for selecting an appropriate methodology, or even appropriate sub-components of many different methodologies, in order to support and promote development success. It also highlights the flexible and adaptive nature of appropriate methodology, especially in dynamic, emerging environments where change is a dominant characteristic.

McConnell (1996) supports this idea by stating that organizations must master the ability to select the most appropriate and effective processes and practices that have been identified from other successful or not so successful projects. He goes on to point out that organizations must choose practices that are specific to achieving their unique project goals and schedule objectives.

The basic philosophy of how people are viewed and valued by many organizations is another significant issue for a "heavy" methdology. Ambler (2002) points out that these methodologies attempt to lay out a prescriptive set of processes — follow each step exactly as predetermined and a successful project will be the result. The idea that the process steps are more important than the individuals carrying them out may be a recipe for disaster. The theory is that if you follow the rules, it does not matter who carries them out. If key people leave the project or the organization, the project should go on as scheduled because everything is documented. By experience this does not appear to be the case. Regardless of how well the system is documented, if key people leave, the project suffers. Organizations need to use methodologies that appropriately factor in people issues in addition to process ones. Adaptive methodologies consider not only the technology involved and the type of application but also the number and experience of the people involved.

METHODOLOGY APPLIED

The case for a less methodical methodology is based on three assertions: (1) the failure of methodology to provide consistent success in SE; (2) the lack of well-

developed, well-defined methodologies applicable to a fast-paced, innovative, emergent systems development environment; and (3) the need for using an appropriate approach to SE that balances the demands of effectiveness with efficiency — a level of methodology that is "just right."

On Failure and Methodology

In the introduction to his widely used text on software engineering, software quality guru Roger Pressman (1997) refers to what he calls a three-decades long "chronic affliction," that affliction being the ongoing problems associated with software development and the continuing high rate of failure. Despite investments in software engineering, management frameworks and development methodologies, failures still litter the IT landscape. The reasons for failure are numerous (see Lientz & Rea [2001] for an excellent annotated list) and the application of methodology *per se* certainly should not be universally offered as the *only* critical factor in success or failure. In fact, defining failure is not a simple task. For example, is a six-month project that comes in two weeks late a failure; or one that returns its investment in 36 months instead of the expected 32?

Popular estimates that claim a 50 percent or greater failure rate are not easily verified but anecdotal evidence is abundant. The Standish Group's CHAOS survey (Johnson, 1999), for example, claims that only 26 percent of 1998 projects were deemed "successful" by survey respondents. In a review of a Standish Group survey, conducted in 2000, Berinato (2001) points out that although some numbers have improved we have a long way to go. Outright failures of software projects declined from 40 percent to 23 percent but "challenged" projects rose from 33 percent to 49 percent. "Challenged" in this context represents projects that had one or more of the following issues: cost overruns, time overruns or delivery of a system with fewer features than were promised. When you look at the total picture, 75 percent of all software development projects could be classified as "challenged." Whatever the true failure rate is, most executives and information technologists would probably agree that failure is significantly more common than one should expect considering the great investment in software engineering concepts and discipline over the past decades. The point is that methodology, as a quality assurance construct that is expected to ameliorate development problems, if not eliminate them, has regularly failed to do so.

Methodology's role in failure may be the result of several kinds of errors. One explanation for failure might be an inappropriate selection of methodology among the variations available — a mismatch between the methodology and the characteristics of the development environment. In addition to this selection failure, another possibility is the misapplication of an appropriate methodology. A third possibility, one that is increasingly common in an emergent technical environment, is the situation in which there is no appropriate methodology on which developers can rely. This

lagging characteristic of methodology may contribute to failure, or at least does not assist in the avoidance of failure, because in an innovative technical environment proven methodologies are not defined and documented. The beginning points of large software development paradigm shifts are especially susceptible to this kind of failure.

Software development for enterprise integration is arguably a current example of this problem. In the past five years the pressure to integrate existing processes and systems (rather than building new ones) or to implement packaged software (rather than developing in-house versions) has grown tremendously. A survey by Morgan Stanley states that 35 percent of Fortune 500 companies list integration as their top objective (Sullivan, 2001). Other listed objectives, such as e-business and Customer Relationship Management, also have a critical integration component and when included would tend to increase that percentage significantly. The Boston Consulting Group (Dickel & Sirkin, 2000), in a survey of more than 100 CEOs and CIOs involved in enterprise-wide systems implementations, reported that only 33 percent of integration projects were viewed as "positive" in terms of value creation, cost effectiveness and financial impact. Based on investment and expectations, this is an arena in which successful methodologies are needed but are sorely lacking.

Methodology in a Fast-Paced, Emergent Environment

IT is arguably the most disruptive force to organizations in the past century, being both a driver and an object of organizational change. In its role as chief enabler of better-faster-cheaper for the organization, IT is a critical factor in delivering competitive advantage within the rapidly changing business environment. E-commerce and e-business place a particular strain on methodology, requiring "trade-offs between schedule, functionality, resources and quality" (Yourdon, 2000a) in an extremely demanding environment. A contemporary software development methodology, therefore, will be required to operate in an environment with the following characteristics (adapted from Mendonca, 2000; Yourdon, 2002):

- A rapid pace in introduction of new technologies (software and hardware);
- A demand for expeditious development and implementation, leading to new rapid development and implementation techniques;
- Telecommunications integrated into, and inseparable from, the computing environment;
- Modularization of hardware and software, emphasizing object assembly and processing;
- Integration of seemingly incompatible and diverse technologies.
- More demanding users and managers, who have high expectations for the business value and investment return of Internet-based commerce;
- Exacerbated peopleware issues, for example finding or developing appropriately prepared information technology workers;

- High risk projects, due to innovation and expectations of quick delivery and high investment return, require a methodology that incorporates rigorous risk assessment and control.

In this kind of emergent, unformed, somewhat chaotic environment, a traditional approach to using an optimal methodology — predictable, tested and proven — has no application. There is no doubt that, even in environments where the pace is not so rapid and change not so prevalent, methodology is viewed by business proponents, users and some IT non-managerial staff to be overly rigorous, bureaucratic and burdensome to the point of being a hindrance rather than an enabler. It is viewed as being even more so in a fast-paced, emergent environment.

Appropriate Methodology: Just Enough

There are numerous ways to judge the success or failure of any one particular software development effort. Measurements commonly identified include: effectiveness (Does the product function as expected?); value (Does the product meet its financial and other value-enhancing goals?); cost (Is the cost justified by the benefits accrued?; Is the project within budget?); and timeliness (Was the project delivered on time?). As noted, these parameters play against each other — that is, as we increase resources to ensure factors such as effectiveness and timeliness, we increase costs associated with the project and, therefore, erode efficiency valuations. Recognition of this tension in the business environment is important to IT and non-IT staff alike.

Having no methodology at all risks chaos and possible delivery of a product that does not adequately respond to user needs, is over budget or has taken too long to implement. On the other hand, a highly structured, rigid methodology, what James Highsmith very descriptively calls a "monumental" methodology (Highsmith, 2000), may be too costly because of the use of unnecessary resources and time delays. Organizations faced with time-sensitive e-business projects that demand quick implementation for competitive advantage must carefully consider the options. The challenge is, of course, to identify how much is "just right." Under some circumstances, a less rigid methodology may be more appropriate, while under other circumstances, for example in a well-defined environment with known deliverables, a more rigid methodology is a good choice.

The SDLC model, as embodied in formal commercially available methodologies developed by consulting companies, is certainly an example of the optimized, monumental approach to methodology. Another example, arguably, is the Software Engineering Institute's Capability Maturity Model for Software (CMM), which has become widely recognized within the IT industry. It includes a detailed description of processes in the context of software development process improvement (Paulk, 1995). Although it is presented primarily as a framework for improvement, rather

than a methodology per se, CMM defines multiple "key process areas" (methods) that organizations adopt to develop full capability in quality software development. However its formal procedures, detailed documentation and heavy resource requirements have come under for inapplicability to small and medium-sized projects and development in fluid environments. The result has been that some proponents are now advocating using "CMM with good judgment" to soften its inflexibility (Paulk, 1999).

A systems approach to methodology considers environmental factors that determine not only the appropriate selection of processes, methods and tools but also the appropriate level of formality to be applied. These factors include:

1. Technology factors: What technologies are being used and by whom? Are they new or mature? Have successful appropriate methodologies been developed that provide a good fit for the technology?
2. People factors: What methodologies are developers experienced with? Is there a good expectation of strong developer/user collaboration? Are users available, and committed to a high level of involvement?
3. Project definition factors: Is the project well defined in scope and objectives or will it require definition as it unfolds? What is the expected development timeline? Can the project be subdivided in mini-projects with smaller deliverables? What are the customer expectations with respect to performance, memory use, robustness and reliability?
4. Processes: Are business processes stable or does the project require process re-engineering?

Yourdon (2000a) calls this approach a "risk/reward" approach to defining an appropriate level of resource investment in methodology. A chaotic nonmethodology development environment oriented to "code and fix" is not an acceptable alternative but neither is an inflexible monumental methodology in many circumstances. Fowler (2000) suggests that a flexible, less methodical approach to methodology attempts a "useful compromise between no process and too much process, providing just enough to gain a reasonable payoff."

TOWARD A LESS-METHODICAL METHODOLOGY

What does a less methodical methodology look like and how does it respond to the requirements described above? Indeed, is a light, lean, extreme, adaptive, agile, appropriate methodology a methodology at all? A contingency approach to methodology recognizes the limitations of methodology, the experience of failure and the inherent difficulty in creating optimized methodologies in a rapidly changing emergent environment. Its approach is to choose the processes and techniques appropriate to

a given environment, to take advantage of the capabilities of any and all formalized methodologies and to apply guidelines for developing processes that may not have been fully defined.

The definition of this approach to software development continues to evolve, with many participants from the project management, quality assurance and software engineering disciplines joining together to contribute. Developers of the XP concepts, notably Kent Beck and Martin Fowler, have played an important role by way of adopting similar principles for flexible, adaptive programming techniques. Other development frameworks, such as Crystal methods, Scrum, Adaptive Software Development and Dynamic Systems Development Methodology, have also contributed (Highsmith, Cockburn & Boehm, 2001). Highsmith's book on adaptive software development, recent writing on "agile" development and the Manifesto form the core of these principles and guidelines.

Fowler (2000) very accurately describes these related methodologies as fundamentally "adaptive rather than predictive" and "people-oriented rather than process-oriented." While it is not the objective here to fully define the nature of these methodologies, basic principles include:

- Agile/simpler processes that continuously respond to changes in the environment;
- Appropriate selection of process components that reflect efficiency in addition to effectiveness;
- An adaptive approach (frameworks) rather than adherence to pre-defined process rules;
- Frequent, rapid delivery of smaller software components to achieve faster more accurate feedback;
- A collaborative approach to development;
- An expectation of change during the development process;
- Outcomes are emergent, rather than fixed;
- Creativity in problem solving;
- Dynamic re-prioritization — at appropriate intervals, the team should reevaluate the current effectiveness of processes and adjust as necessary.

Berinato (2001) makes these suggestions on ways to become "agile":

- Slash the budget, enforcing smaller and simpler projects that are better defined and more easily achievable.
- If it does not work, kill it. Project teams need to build in critical reviews of the project at predetermined short-time intervals. These review need to access two things — does the system work and does it fulfill a needed purpose.
- Keep requirements to a minimum. Each implementation iteration of a system should deliver a subset of the requirements and be budgeted separately.

- Build on demonstrated and documented success, not hope.
- Keep development teams small, so effective communication between team members is more easily accomplished.

The principles and guidelines of a flexible, less methodical methodology continue to be described and defined. It is clear that this is a genuinely different approach than the optimized one inherent in a fixed and fully defined methodology. Practices such as continuous feedback and incremental product delivery will form the core of techniques for implementation of this approach. However, there clearly needs to be further work in identifying those frameworks and practices that support the concept and can support successful delivery of quality software. Ironically, like any other methodology, it will need to be tested and proven. Considering the limitations of methodology and the increasing complex environment in which software is engineered; however, it has good potential and is worthy of our attention.

REFERENCES

Ambler, S.W. (2002). *Agile modeling: Effective practices for eXtreme programming and the unified process.* New York: John Wiley & Sons, Inc.

Berinato, S. (2001, July). The secret to software success. *CIO Magazine*.

Betts, M. (2002, May 20). The future of software development. *Computerworld, 36*(21), 271.

Capper, N.P., & Colgate, R.J. (1994). The impact of object-oriented technology on software quality. *IBM Systems Journal, 33*(2), 284-286.

Dickel, K., & Sirkin, H. (2000). Getting value from enterprise initiatives: A survey of executives. Boston Consulting Group, Inc. Retrieved from the World Wide Web: http://www.bcg.com/publications/files/Enterprise_computing_report.pdf. 18,452 Links on Objects and Components. Retrieved from http://www.cetus-links.org.

Fowler, M. (2000). Put your process on a diet. *Software Development*.

Highsmith, J.A. (2000). *Adaptive software development: A collaborative approach to managing complex systems*. Dorset House.

Highsmith, J.A., Cockburn, A., & Boehm, B. (2001, September). Agile software development: The business of innovation. *Computer*.

IEEE. (1993). Standards collection: software engineering. *IEEE Standard,* 610, 12-1990.

Ivari, J., Hirschheim, R., & Klein, H.K. (2000/2001, Winter). A dynamic framework for classifying information systems development methodologies and approaches. *Journal of Management Information Systems*.

Johnson, J. (1999, December). Turning chaos into success. *Software Magazine*.

Lientz, B.P., & Rea, K.P. (2001). *On time technology implementation*. London: Academic Press.

Manifesto for agile software development (2001). Retrieved from the World Wide Web: http://www.agilealliance.org.

McConnell, S. (1996). *Rapid development: Taming wild software schedules*. Redmond, WA: Microsoft Press.

Mendonca, J. (2000). Educating the business information technologist: Developing a strategic IT perspective. In M. Khosrow-Pour (Ed.), *Information Technology Education in the New Millenium.* Hershey, PA: Idea Group Inc.

Nicholas, J. M. (2001). *Project Management for Business and Technology*. Upper Saddle River, NJ: Prentice Hall, Inc.

Paulk, M., et al. (1995). *The capability maturity model: Guidelines for improving the software process.* Pittsburgh, PA: Carnegie Mellon University, Software Engineering Institute.

Paulk, M.C. (1999, June). Using the CMM with good judgment. *Software Quality Professional*.

Pressman, R. S. (1997). *Software engineering: A practitioner's approach*. New York: McGraw-Hill Companies, Inc.

Sullivan, T. (2001, August). Take your medicine. *Infoworld*, August 10.

Yourdon, E. (2000a, August). Success in e-projects. *Computerworld*, August 21.

Yourdon, E. (2000b, September). The 'light' touch. *Computerworld*, September 18.

Yourdon, E. (2002). *Managing high-intensity Internet projects*. NJ: Prentice Hall, Inc.

Chapter V

How to Elaborate a Use Case

D.C. McDermid
Edith Cowan University, West Australia

ABSTRACT

This paper challenges established wisdom with respect to use cases. Use cases are classically elaborated by directly identifying objects, methods and data. The research reported in this paper indicates that there are other better constructs for modeling use cases, at least initially, and that objects are not a particularly good medium for discussing requirements with users. This paper rehearses the arguments leading up to these conclusions and identifies some implications of these conclusions.

INTRODUCTION

This paper describes conclusions from two action research studies concerned with modeling use cases. Because an alternative structure of a use case is proposed, the term **business rule** will be used throughout to signal that a different structure for a use case (though not purpose) is implied. At this point, however, the reader may regard the terms business rule and use case as synonymous. Initially in these action research studies, the classical process of elaborating use cases was followed (Jacobson, Christerson, Jonsson & Overgaard, 1992). That is, business objects were

used as a means of capturing and documenting the requirements of the information system, so the ability of business objects to describe requirements in the early stages of developing a system was tested. It was found that business objects did not contain constructs that were **directly** conducive to requirements gathering from users and neither did they facilitate presentation and discussion of requirements at an appropriate level of abstraction. It was found that a more appropriate vehicle for analyzing requirements at this stage was brought about by (customizing) the notion of a **business rule**.

This paper attempts to explain the relationship between business objects and business rules. In doing so, it highlights some inadequacies of business objects as a medium for gathering and expressing requirements at any early stage of development, and, most importantly, it demonstrates how all analysts who work with use cases can improve requirements specification**.** Some of this discussion relates to how users prefer to conceptualize their world; other parts of this discussion relate to the pragmatics of conceptual modeling.

The paper proceeds as follows. The remainder of this section introduces the idea of a business rule and some of its history in the literature. It then details the action research studies so the reader can appreciate the ensuing discussion. This is followed by a discussion of certain inadequacies of business objects that arose from the research.

There has been a continued and active discussion of business rules in both the practitioner and academic literature. With regard to the practitioner literature, especially in North America, there has been a concern of insufficient treatment of and insufficient focus on business rules generally (Chikofsky, 1990; Sandifer & Von Halle, 1991a, 1991b; Jones, 1991; Lucas, 1993; Moriarty, 1993a, 1993b, 1993c, 1993d; Von Halle, 1994; Baum, 1995). The discussion in the practitioner literature is wide ranging. For example, it covers questions of how a DBMS environment can deal with business-rule implementation options (Von Halle, 1994) through to Joint Application Design (Lucas, 1993). Much of the discussion, however, quite naturally focuses on immediate solutions to existing problems, such as illustrating how business rules could be integrated into data dictionaries (Sandifer & Von Halle, 1991a) or how CASE tools could improve the documentation of business rules (Baum, 1995), rather than exploring the conceptual basis for modeling business artifacts.

As regards the academic literature, there are a number of examples of authors actively using the term business rule in their work (Feuerlicht & Blair, 1990; Ross, 1994; Sandy, 1994; Herbst, 1996). However there are difficulties in identifying work in this area because there are a wide variety of names given to diagrams, which model business rules or, at least, some aspects of a business rule. While there is, as yet, no standardization of terminology in this area, the work of Kung and Solvberg (1986), Kappel and Schrefl (1989) and Tsalgatidou and Loucopoulos (1991a, 1991b) could be regarded as synonymous with business rules modeling. The Business Rule

Diagram discussed here, though it would broadly be considered in the same category as the above, has a number of unique aspects.

The definition of a business rule provided here suggests something of the context and nature of a business rule as well as identifying its constructs. It is defined as "…an explicit state change context in an organization which describes the states, conditions and signals associated with events that either change the state of a human activity system so that subsequently it will respond differently to external stimuli or reinforce the constraints which govern a human activity system' (McDermid, 1998, p. 20).

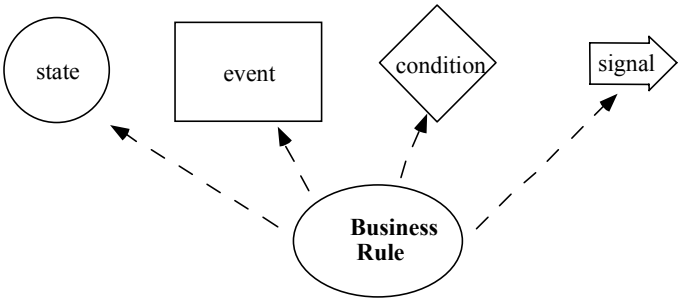
The definition contains four explicit constructs: states, events, conditions and signals. **States** reflect the status of an object of interest at any given time, so, for example, a manufacturing work order might occupy the states planned, in progress or completed. **Events** are actions carried out internally by the organization that change the state of one object. They are considered to be instantaneous occurrences that reflect the organization's policy on what should happen in a particular circumstance, e.g., cancel work order. One important role of the event is to avoid specifying processing detail. This was seen as a pitfall at this early stage. Events differ from methods in that methods may not change states (for example, a method may simply reveal data in response to a service request). **Conditions** define the criteria by which objects of interest in the business move from one state to the next as events take place. Sometimes many conditions must be met in order for an event to take place, increasing complexity. It is argued that modeling conditions without the context of states and events (and vice versa) is far less powerful or useful. Lastly **signals** either enter or leave the human activity system. Signals that enter the system will typically initiate activity within the system and so these are called **triggers**. Triggers may be external such as a customer sending an order or internal such as one department sending a document to another department, which then triggers off some activity. Further a trigger may be a time trigger — an activity beginning at the start of the day or the end of the month. Those signals which leave the system serve the purpose of informing those outside the system of what has occurred inside the system and are referred to as **messages**. Though some might argue that the idea of a condition is at the heart of a business rule (Loosley, 1992), the related constructs of state, event and signal provide a context for the business rule. So, as an aid to memory we might say:

Business Rules = States + Events + Conditions + Signals

The abstraction of these constructs is shown diagrammatically in Figure 1.

The action research studies were performed in two organizations (McDermid, 1998). In the first organization, the researcher worked with a systems analyst on an existing system. The outcome of this study was the production of an initial definition of a business rule and a means of diagramming it. It was in this study that most of

Figure 1. Abstraction of Key Constructs of a Business Rule



the technical limitations of a business object was observed. In the second organization, an analyst and users were studied using this Business Rules Diagram (BRD). In this study, although to large degree a confirmation of many of the technical aspects, aspects of how users prefer to conceptualize their world were observed.

Over the course of the action research studies, six different versions of a (structured) BRD were developed. Earlier versions contained fewer constructs, and, as each version was evaluated, it was concluded that there was a need for additional constructs to ensure completeness in the description of a business rule. The four major constructs identified were seen as the minimum for holding a reasoned discussion with users at this stage. Observe that data was not one of the four constructs and neither was method, although there is some overlap between the event construct and method. By the end of the studies, a methodology for constructing the BRD had been developed. The main steps in producing a BRD are:

- Identify candidate business rules;
- Identify candidate business events and signals;
- Identify candidate business objects in problem situation;
- Construct object life history for each candidate object identified;
- Construct User Business Rules Diagrams;
- Construct Business Rules Diagram.

Candidate business rules at this stage are brainstormed with users as simple narrative statements (Table 1). Later the business rule will be expressed more rigorously in a diagram. Similarly candidate business events and signals are brainstormed at this point (Table 2). Notice that rows are categorised into events, triggers and messages.

The purpose of these two steps is to establish enough context about a problem situation to begin to identify business objects and their associated life histories. While there is necessarily considerable organization required at this point in sorting out the objects and life histories, the actual process is well-known and familiar to object and data modellers alike. At this stage the User Business Rules Diagram (UBRD) can

Table 1. Example of Candidate Business Rules

- Orders sent by mail or telephone
- Omission on order line leads to deletion of that order line
- Credit balance >= order value to accept order, otherwise reject
- Stock qty >= order qty for normal order, otherwise outstanding
- One invoice for one order
- Sum of payments = order value - sum of credit notes
- One order may have many credit notes
- Many payments per invoice possible

be constructed. As indicated earlier, for the purposes of this paper, an UBRD can be considered a use case, i.e., a discrete meaningful episode of work in the mind of a user.

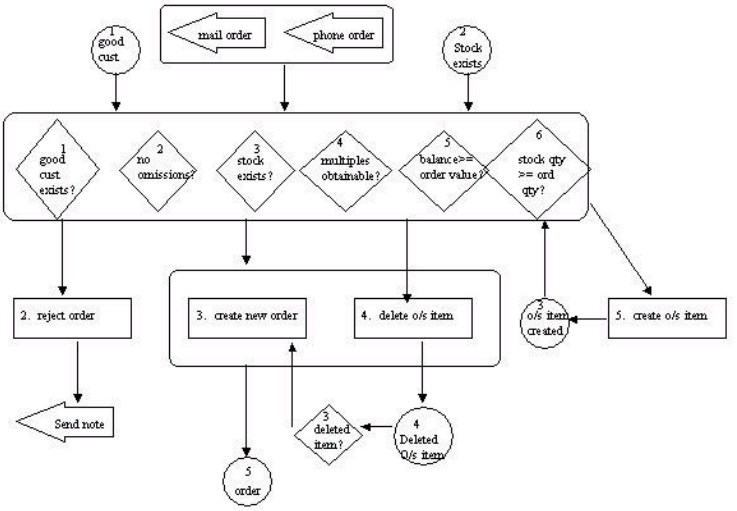
Figure 2 shows an example of an UBRD representing one business rule which would become part of a complete BRD. States are represented by circles, events by rectangles, conditions by diamonds and signals by thick arrows. The softbox is a Harel blob (Harel, 1988), which acts as an encapsulator of constructs. The example in Figure 2 is the most complex state change context so far modelled; the vast majority of business rules are much simpler involving typically no more than five or six constructs. While a full description of the BRD is outside the scope of this paper, Figure 2 illustrates the potential complexity of a state change context.

In this particular example, a single business rule may result in different events taking place (since events are tied to a single object). For example, if the order ends up being rejected, only event 2 is executed. On the other hand, it is possible for an order to be accepted but some of its items held as outstanding items until sufficient stock is available. In this case, event 3 (create new order) takes place but also event 5 (create outstanding item) is executed. Also two state changes occur in this scenario. So it can be seen that a state-change context has to be able to describe all potential events that may occur, and it requires a sufficiently rich notation to support this.

Table 2. Candidate List of Events and Signals

Receive customer order	T
Delete line	E
Reject order	E
Create new order	E
Send invoice	M
Generate credit note	E

Figure 2. User Business Rules Diagram



THE IMPORTANCE OF STATE IN DESCRIBING A USE CASE

One unique aspect in the definition of a business rule (use case) is the idea of being a state-change context. Most other definitions of a business rule focus on conditions and do not involve the state as a construct (e.g., Appleton, 1988). In the definition used here, however, states are those anchors upon which the business rule is modelled. Without states it is much more difficult to put a boundary on a business rule so discussion with stakeholders is harder. The state-change context makes business rules modeling more tractable and potentially reduces the number of states that have to be modelled at this level, i.e., working with users to establish the requirements of a system. For example, within the process of accepting an order, there may be a number of "sub-states," such as order received and copied to standard order form, credit status checked and acceptable, stock availability checked, etc. However a customer does not necessarily get involved with these internal matters. From a customer perspective, the order is either accepted or rejected, yet an order processing clerk is likely to perceive the same process from a lower level of abstraction. Unless there is a standard for this, two different analysts could well model related problems at different levels of abstraction resulting in confusion. Worse still is the possibility that one level of abstraction is not simply an elaboration of the other. This may happen if there is overlap between abstraction by function and abstraction by geography, for example. By insisting that state changes must be detectable by an observer external to the human activity system, the unit of abstraction is defined in a way that can be externally validated, reducing inconsistency.

This has profound implications for accepted wisdom in use cases generally and in business objects in particular. It is suggested that for business objects the first formal construct identified should be the state (as opposed to method or data). Further identifying states can be seen as a precursor to identifying methods and data as follows. Suppose a business object (called order) has the states ordered, invoiced and paid. Data typically associated with the object order is more strongly coupled to its states than to the object itself. For example, data such as order-no and order-date might be associated with the object when it is put into the ordered state, while invoice-no and invoice-date might be associated with the object when it is moved to the invoiced state and so on. Similarly the identification of methods follows from having identified the states, although it is a moot point. Nevertheless this discussion does raise the question as to whether the traditional view of immediately identifying data and methods is appropriate for business objects, especially when it is argued that states provide a better basis for establishing data.

THE RELATIONSHIP BETWEEN USES CASES AND OBJECTS

The research conducted also highlighted a more general problem with object modeling. There is an incongruence between what may be termed "system" events and "object" events. In Figure 2 a number of events may fire depending on the result of executing conditions. For instance there were events concerned with accepting an order outright, rejecting an order and even a scenario in which most of the order is accepted but some items are kept as outstanding order items until stock is available. Now each event relates to one object and one object only. Yet in a use case (or business rule), many objects may be changed in some way. In other words, a business rule is a better and more convenient unit of abstraction to review business policy than an event (or method) in a business object. While on balance it is still appropriate and overall desirable to decompose a system into business objects in order to "divide and conquer," business objects do not appear to be a useful vehicle for exploring business policy. This is because some business rules affect many business objects.

It can also be observed in Figure 2 that conditions and signals are more directly related to the business rule than the business object (though it may often be the case that individual conditions and signals relate only to one business object). In Figure 2 there are a number of conditions that are tested. The first condition is whether a customer is in good status. If not the order request will be rejected at that point; if so further conditions will be tested to establish whether or not the whole order will be accepted or whether part of it will be made outstanding. The point of this illustration is to demonstrate that conditions are not directly related to one event; rather they contribute to the business rule as a whole and through that to perhaps

several events (or methods). Indeed it would be possible for the same event to be executed within different business rules, for example, where there were different procedures (i.e., business rules) for removing a customer object. This same logic also holds for signals. For instance, in Figure 2, the trigger of either a mail or phone order applies to the whole business rule and not any one single event or state change within the rule.

In summary, it can be concluded that there is a many-to-many relationship between business objects and business rules; or one business object may be affected by many business rules and one business rule may affect many business objects. What is new in this paper is that the constructs of state, event, condition and signal are asserted as intermediate linchpins connecting business objects with business rules.

BUSINESS OBJECTS AND COMPUTER OBJECTS

The implications of the assertions in this paper are far reaching. The two main assertions are that, at the early stages of systems development, the more primitive construct of state is a more immediate and appropriate way of conceptualizing objects and, secondly, that the role of the business object as the central unit of abstraction is somewhat diminished by the business rule (or use case, if you will). While it is true that some have argued for the dominance of the use case (Jacobson et al., 1994), there has been relatively little structural discussion on how or whether constructs, such as trigger, conditions, messages, states and events, play a role as better constructs for modeling purposes.

The tenets of object modeling are built on seamless integration and reduction of the semantic gap between models (Jacobson et al., 1994), yet, if business rules displace (to some extent) the role of the business object as the prime unit of abstraction in the early stages of requirements engineering, then this calls into question the validity of using business objects to generate so-called "computer" object models (which may contain many additional types of objects such as interface and control objects). Here the needs of business modelers and computer-based object modelers (seeking, for example, to optimize reusability) are more different than previously imagined. Given the above discussion, an alternative would be to consider using business rules as the prime medium for seamless integration throughout the development life cycle.

REFERENCES

Appleton, D. (1988, February). Second generation applications. *Database Programming and Design,* 54.

Baum, D. (1995, March). The right tools for coding business rules. *Datamation,* 36-38.

Chikofsky, J. (1990, May). The database as a business road map. *Database Programming and Design,* 62-67.

Feuerlicht, G., & Blair, A. (1990). Development of a prototype system for the management of business rules and integrity constraints in database applications. *Expert Systems for Information Management,* 5(2), 79-93.

Harel, D. (1988). On visual formalisms. *Communications of the ACM,* 31(5), 514-530.

Herbst, H. (1996). Business rules in systems analysis: a meta-model and repository system. *Information Systems,* 21(2), 147-166.

Jacobson, I., Christerson, M., Jonsson, P., & Overgaard, G. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach*. Wokingham, England: Addison-Wesley.

Jones, B. (1991, November). Letter to editor. *Database Programming and Design,* 9.

Kappel, G., & Schrefl, M. (1989). A behavior integrated entity-relationship approach for the design of object-oriented databases. In C. Batini (Ed.), *Entity-relationship approach* (pp. 311-328). North Holland: Elsevier Science.

Kung, C. H., & Solvberg, A. (1986). Structural and behavioural modeling. In T. Steel & R. Meersman (Eds.), *Database semantics* (pp. 205-221). North Holland: Elsevier Science.

Loosley, C. (1992). Separation and integration in the Zachman Framework. *Data Base Newsletter, 20*(1).

Lucas, M. (1993, September). Getting down to business. *Informatic*s, 41-45.

McDermid, D. C. (1998). The development of the business rules diagram. Unpublished doctoral dissertation, Curtin University of Technology.

Moriarty, T. (1993a, April). Business rules analysis. *Database Programming and Design,* 66-69.

Moriarty, T. (1993b, June). Losing the business. *Database Programming and Design*, 66-69.

Moriarty, T. (1993c, February). The next paradigm. *Database Programming and Design*, 66-68.

Moriarty, T. (1993d, May). Where's the business? *Database Programming and Design*, 68-69.

Ross, R. (1994). *The business rules book: Classifying, defining and modeling rules*. Database Research Group.

Sandifer, A., & Von Halle, B. (1991a, January). Designing by the rules. *Database Programming and Design*, 11-14.

Sandifer, A., & Von Halle, B. (1991b, February). A rule by any other name. *Database Programming and Design*, 11-13.

Sandy, G. A. (1994). Nice models but where's the business. *Proceedings of the 5th Australian Conference on Information Systems*, Melbourne, Australia.

Tsalgatidou, A., & Loucopoulos, P. (1991). Rule-base behaviour modelling: Specification and validation of information systems dynamics. *Information and Software Technology, 33*(6), 425-431.

Von Halle, B. (1994, January). Making business rules real. *Informatics*, 48-49.

Chapter VI

A Rigorous Model for RAISE Specifications Reusability

Laura Felice
Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina

Daniel Riesco
Universidad Nacional de San Luis, Argentina

ABSTRACT

During the RAISE specification development process, a variety of components and infrastructures are built. All of these components are not independent, but they are related to each other, especially when we specify different systems in the same infrastructure. The RAISE method is based on the idea that software development is a stepwise, evolutionary process of applying semantics-preserving transitions. So, the reuse process is crucial in all stages of the development, but there is not explicit reference to the specification reusability in this development process.

This chapter presents a rigorous process for reusability for RAISE Specification Language (RSL) components. We provide the mechanism to select a reusable component in order to guide RAISE developers in software specification and construction.

INTRODUCTION

Software components are typically very rich in information, making the task to characterize them and capture their relevant properties difficult. However this is not the only reason that makes software reuse difficult.

Information retrieval methods based on analyses of natural-language documentation have been proposed for constructing software libraries (Helm & Maarek, 1991; Maarek, Berry & Kaiser, 1991). Software components represented by natural language can make the retrieval process a task with ambiguity, incompleteness and inconsistency. All of these problems can be minimized by using a rigorous method in the retrieval of a component.

The RAISE method (D. Bjorner, lecture notes, Technical University of Denmark, 2000) is based on the idea that software development is a stepwise, evolutionary process of applying semantics-preserving transitions.

Based on this observation, we propose to introduce a Reusable Component (RC) model for the definition of the reusable component structure into RAISE.

In this work we propose the RC model for the definition of the structure of a reusable component that integrates specifications in RSL (George, Haff, Havelund, Haxthausen, Milne, Nielson, et al., 1992) and object-oriented code.

The RC model describes object-oriented classes at different levels of abstraction:

- Specialization — hierarchies of RSL implicit specifications related by formal specialization relation;
- Realization — hierarchies of RSL complete algebraic specifications related by realization relations;
- Code — hierarchies of imperative RSL schemes related by implementation relations and linked to object-oriented code.

We define a rigorous process for reusability of RC components. Its manipulation, by means of specification building operators (Rename, Extend, Combine, Hide), is the basis for the reusability.

Our approach allows that the properties of components formally specified can be characterized by giving a functional (RSL specification) description. Therefore they may be useful to someone searching for a particular component.

Different possible classes of existing RC components may be retrieved using a formal reasoning technique: an exact match to the query specification, a component more general than the query, or a component more specific than the query. An illustrative example presents a reusable component classification and a specification matching.

RELATED WORKS

Different approaches to specifying reusable components functionalities have

been proposed. The way in which the components can be used with others can play a critical role in the reuse implementation.

Related to the RAISE method, we emphasize the work of Beltaifa and Moore (2001) where they propose an infrastructure to support reuse which improves both the ease and efficiency of reusing software components. The main difference with our work is the integrated process defined for all stages of the development method.

As a typical related work, we can mention Hennicker and Wirsing (Lecture notes, 1992) who presented a model for reusable component definition. A reusable component is defined as an unordered tree of specifications where any two consecutive nodes are related by the implementation relation, and the leaves are different implementations of the root. The work of Chen and Cheng (1997) is another approach that provides a formalism to register components properties to reuse them based on the architecture and integration of the system. They are related to LOTOS tools that facilitate the retrieval of the reusable component.

On the other hand, the work of Zaremski and Wing (1997) is related to specification matching. It is very important to emphasize this proposal has been referenced by a lot of authors.

The survey paper by Krueger (1992) discussed a different approach for software reuse. Eight categories, such as high-level languages, source code components, application generators, etc., were discussed.

Carma-McClure (1995) gives a detailed explanation of how to make reuse work in practice. The purpose of this survey is to fill in the missing details about how to practice software reuse. It is a step-by-step guide that empowers the reader to infuse reuse into the software development process (both object-oriented and traditional structured/nonobject-oriented processes) and to attain the maximum benefits it can offer.

Penix (1998) proposes automated component retrieval and adaptation using a heuristic based on specification semantics for approximating specification matches that indicate component reusability. He gives a formal model of architectures using algebraic theories to specify relationships between the system and component specifications.

THE RAISE METHOD

There are two main activities in the method: writing an initial specification and developing into something that can be implemented in a programming language (George, 2002). Writing the initial specification is the most critical task in software development. If it is wrong, i.e., if it fails to meet the requirements, the following work will be largely wasted. It is well known that mistakes made in the life cycle are considerably more expensive to fix than those made later.

What kinds of errors are made at the beginning? The main problem is that we may not understand the requirements. The requirements are written in a natural

language and, as a result, are likely to be ambiguous. The aim of the initial specification is to capture the requirements in a formal, precise manner. Formality means that our specification has just one meaning. It should:

- Be abstract and leave out as much detail as possible. The requirements may demand that identifiers have a certain format, but we try to extract the essential information.
- Use users' concepts. For instance, if the requirements say that each customer has an account, and an account is a record of all the customers' transactions, then that is what the specification should say.
- Make it readable. Specifications are intended to be read by others: by those who check that they correspond to requirements, by those who implement them, by those who write test plans, by those who later maintain the system, etc.
- Look for problems. What we want to do is avoid mistakes or find them quickly. So we concentrate on the things that appear difficult, strange or novel, and we ignore or defer things that are straightforward.
- Minimize the state. This means in particular that we try hard not to include in the state dependent information — information that can be calculated from other information in the state.
- Identify consistency conditions. Consistency conditions are needed if some possible state values cannot correspond to reality — two users of a library borrowing the same copy of a book simultaneously, perhaps.

Kinds of Modules

RAISE identifies two kinds of modules: global objects and state components. Global objects are objects declared at the top level in a separate file. In general, they are not advised because they have too wide a scope. But there are typically collections of types that we need in many places, such as identifiers for various kinds of entity, and it is convenient to collect these in one global object. A guide to when types should be in a global object is that types visible to users, i.e., types that occur as parameters to user functions or in the results of user functions, should generally be defined in one.

Most modules will contain a type modeling (a part of) the state, together with functions to observe it and generate values of it, and we name these state components. Generators usually include functions to change state values and perhaps also to create them. The type is often called the type of interest of the module. Such modules are usually defined as schemes and typically instantiated within others. Modules should have only one type of interest. We write separate modules for each state component because we can then enforce a discipline that the part of the state within the module is only accessed through the functions defined for it. This enables us to change the way that part is modeled without affecting anything else, as long as we maintain the original properties. Such a technique is known as

encapsulation through information hiding. Object-oriented approaches to program design follow the same ideas: They are typically called the observers and generators methods.

There are various ways of writing modules according to the way in which the type of interest is defined.

THE RSL

The aim of the project RAISE was to develop a language, techniques and tools that would enable industrial use of formal methods. The results of this project include RSL, which allows us to write formal specifications. In addition to this, a method to carry out developments based on such specifications and a set of tools to assist in edition, checking, transforming and reasoning about specifications are provided.

RSL is a "wide spectrum" language that can be applied at different levels of abstraction as well as stages of development. It includes several definition styles, such us model-based, property-based, applicative, imperative, sequential or concurrent.

A specification in RSL is a collection of modules. Modules allow the decomposition of a specification into more understandable and reusable units. A module is basically a named collection of declarations, and it can be a scheme or object. Each module should have only one type of interest. However, the kernel module concept is that of a class expression. A basic class expression is a collection of declarations enclosed by the keywords **class** and **end** and represents a class of models. Objects and schemes are defined using class expressions. An object is a named model chosen from the class of models represented by some class expression. Objects can be global, embedded or parameters for sharing. They allow one to express the dependency of a module on other modules. In some situations it is convenient to be able to manipulate a class expression before defining objects. So a name should be given to the class expression. The named class expression is called a scheme.

An applicative class expression contains type, value and some axiom definitions. Axioms may be used to constrain the values.

A type is a collection of logically related values, and it may be specified by an abstract or concrete definition. An abstract type, also referred to as a sort, has only a name. A concrete type can be defined as being equal to some other type or, using a type expression, formed from other types.

There are some type constructors that allow the definition of composite types:

- products (X),
- total functions (->) and partial functions (~->),
- sets (-set for finite sets) and -infset for infinite ones,
- lists (* for finite lists and ^{w} for infinite ones),
- maps (*m*-> for finite maps and *m*~-> for infinite ones).

Sets, lists and maps define collections of values of the same type. A set is an unordered collection of distinct values, while a list is a sequence of values of the same type. A set is an unordered collection of distinct values, while a list is a sequence of values, possibly including duplicates. A map is a table-like structure that maps values of one type into values of another type.

RC Model Description

RC describes object classes at three different conceptual levels: specialization, realization and code. These names refer to the relations used to integrate specifications in the three levels. A more detailed description can be found in Felice, Leonardi, Favre and Mauco (2001).

RC Components

The **specialization level** describes a hierarchy of incomplete RSL specifications as an acyclic graph. The nodes are related by specialization relations. In this context, it must be verified that if P(x) is a property provable about objects x of type T, then P(y) must be verified for every object y of type S, where S is a specialization of T.

Specialization level reconciles the need for precision and completeness in abstract specifications with the desire to avoid over specification.

Every leaf in the specialization level is associated with a subcomponent at the realization level. A realization subcomponent is a tree of complete specifications in RSL:

- The root is the most abstract definition.
- The internal nodes correspond to different realizations of the root.
- Leaves correspond to subcomponents at the implementation level.

If E1 and E2 are specifications, then E1 can be realized by E2 if E1 and E2 have the same signature and every model of E2 is a model of E1 (Hennicker & Wirsing, 1992).

Adaptation of reusable components, which consumes a large portion of software cost, is penalized by over dependency of components on the physical structure of data.

The **realization** level allows us to distinguish these decisions linked with the choice of data structure. In RAISE there are four main specification style options. They are applicative sequential, imperative sequential, applicative concurrent and imperative concurrent (George, Haxthausen, Hughes, Milne, Prehn & Pedersen, 1995). Associated with them, we can also distinguish between abstract and concrete styles. Imperative and concrete styles use variables, assignments, loops, channels (in concurrent specifications), etc. that are related to design decisions about data structures. Every specification at the realization level is linked to subcomponents at the code level.

The **code** level groups a set of schemes in RSL associated with code. RAISE method provides translation processes, which start with a final RSL specification, and produce a program in some executable language, for example C++ using the translation tool component of the RAISE toolset (George, 2002).

RC Relationships

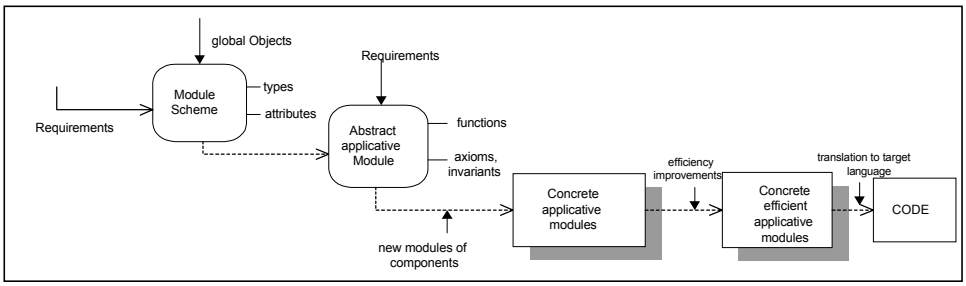
It is worth considering that the three relations (specialization, realization and code) form the "RAISE implementation relation" (George, Haff, Havelund, Haxthausen, Milene, Nielsen, et al., 1992). Any formal system that aims to provide a means of specification and development must provide a notion of implementation. That is, if specification E1 is related to specification E2 in the model, then we need to know if E1 and E2 are in the "RAISE implementation relation." The following properties must be satisfied:

- Properties preservation — all properties that can be proved about E1 can also be proved for E2 (but not vice versa in general).
- Substitutivity — an instance of E1 in a specification can be replaced by an instance E2, and the resulting new specification should implement the earlier specification.

RAISE DEVELOPMENT PLAN: APPLYING A REUSE MODEL

Engineers usually proceed from applicative to imperative specifications. We propose to introduce the RC model for the definition of the reusable component structure into RAISE method. Where to introduce this model? RAISE developers start with the module scheme specification, define an abstract applicative module and develop a sequence of concrete applicative modules and corresponding set of imperative modules from the final applicative modules. Summarizing, we can picture them as in Figure 1.

Figure1. Overview of the RAISE Method



During the first stages of development, when the interaction with the stakeholders is crucial, the use of client-oriented engineering techniques seems to be necessary in order to enhance the communication between the stakeholders and the software engineers. It has been proposed a systematic reuse approach that integrates natural language requirement specifications with formal specifications in RSL. Some heuristics develop a formal specification in RSL starting from models that belong to the Requirements Baseline (Mauco & George, 2000).

The objective is that engineers can make reuse in all development stages. We propose to introduce an RC model in all the development steps in order to include abstraction, selection, specialization and integration of software artifacts in the RAISE method.

Suppose that as part of a system implementation we need a component that we have specified with an abstract applicative module specification $S_{Q\ (query).}$ Further suppose that there is an abstract module in our library with specification $S_{L(library)}$ and its implementation has been verified to be correct with respect to the specification of S_Q. If we can show that S_Q is matched by S_L under generalized module match, then it is known that we can use the library module, and the behavior will be consistent with that specified by $S_{Q.}$ We are using specification match to check that using a library component will not "break" our system.

Thus, when we apply the mechanism to a concrete applicative module, we are selecting an abstract applicative specification adapted to the system requirements having a translation to the concrete specification in the library.

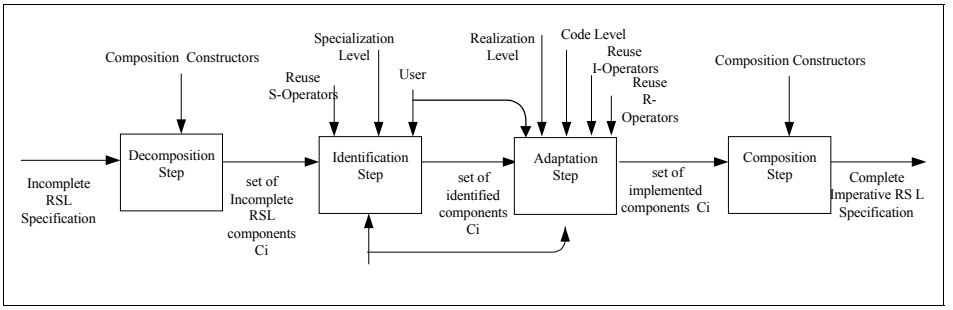
THE REUSE PROCESS

Formal specifications are used to model the problem requirements and the function of the library components. The specifications are written in RSL language. The classification scheme consists of a collection of formal definitions representing possible component features in the domain. The formalization of the scheme permits automated classification of the specifications. The retrieval mechanism is based on syntactic comparison of feature sets. The components returned by the retrieval mechanism are passed on to a more detailed evaluation that uses specification matching to determine reusability.

The results of specification matching determine the relationship that exists between the retrieved components and the requirements specification. The adaptation phase allows for the determination of whether a mechanism exists to adapt or combine the retrieved components to solve the problem.

There is evidence that specification matching to determine component reusability can be carried out using automated theorem proving (Zaremski & Wing, 1997). Attempting specification matching over a large library of components is not a practical retrieval mechanism. The idea is to work by classifying components in a way so that components likely to match for reusability will be assigned similar

Figure 2. The Method



features. By formally defining the classification features and the feature assignment process, classification could be automated.

The main idea is to transform the incomplete RSL specification into complete imperative specification by reusing existing components. The method has the following steps: decomposition, identification, adaptation and composition depicted in Figure 2.

In the decomposition step, the decomposition of a goal specification Eg into subspecifications E1, E2, ... En is formalized.

In the identification step, for each specification Ei, a component Ci (in the specialization level) and a sequence s_1, s_2,..., s_n of RSL specifications must be identified, verifying the implementation relation. A node in Ci must be selected as a candidate to be transformed. The identification of a component is correct if it can be modified by rename, hide and extend operators to match the query Ei.

In the adaptation step, a leaf in the subcomponent associated in the realization level and a sequence of operators used in the previous steps are applied. Then a scheme in the code level is selected, and the same operators in the selected leaf are applied. Finally, in the composition step, the subspecifications Ei and their implementations are composed. Here we are concentrating on the identification step.

RC Identification

In this section the use of specification matching to identify RC components is described. In the identification process, we search for all RC components that satisfy a given query.

It must be able to find the component faster than the user could build it. To address this problem, it is necessary to provide a classification scheme and then manual or automated retrieval techniques.

Our classification components are described by a set of features. Each feature represents a property or an attribute of the component. Features can be refined to give them a more precise meaning. Our objective is to support effective retrieval of the component. In order to be useful in practice, the component specifications should

be independent of the kind of component, allowing the storage of different kinds of them, e.g., requirements definitions, designs, documentation, etc.

Being an RSL specification's collection of modules and basically a named collection of declarations either as a scheme or an object — objects and schemes defined using class expressions — we propose a classification based on feature orientation.

Each feature describes a property of the component and is characterized by:

a. kind of component — describing the function of different kinds of components like requirements specifications, designs specifications, systems specifications, documentation, etc.;
b. operations performed;
c. component structure — modules involved (schemes and objects);
   - granularity of each module — list of objects related with the class;
d. relationships to another component — "implements" relations and composition of components);
e. component specification style — applicative sequential, imperative sequential, imperative sequential, applicative concurrent or imperative concurrent and abstract and concrete styles.

When the possible components are localized, the objective is to compare a component with the query. This process has two essential steps: signature matching and semantic matching. The signature matching enables a syntactic comparison of a query specification with specifications existing in RC reusable components. The semantic matching compares the specifications dynamic behavior. The bases of the signature matching come from Zaremski and Wing (1997), even though they were adapted to the identification of RC components.

The signature of a specification consists of a set of sorts and a set of operations, each operation being equipped with a particular functionality.

Let L=<SL, FL> be the signature of a library specification and Q=<SQ, FQ> the signature of a query specification where SL and SQ are sets of sorts and FL and FQ are sets of operation symbols. The signature matching is defined in Table 1.

This means that given a query specification Q, an RC library C and a predicate P gives back the RC components that satisfy P. The signature matching is based on operations matching. Different kinds of operations matching can be applied. They are the exact, generalized and specialized matchings of operations. This matching requires a specifications signature matching (sorts and operations) and the axioms proofs between pairs of operations.

Table 1.

Signature–Matching:
Query-Signature X RC-Library X Match-Predicate $\rightarrow$Set-of RC-Components
Signature-Matching(Q,C,P) = { c $\in$ C/ P(Q,C)}

Searching for a Reusable Candidate

RC library is a fundamental piece to the engineering because it contains existing components that will be reused later in the large systems constructions. The appropriate use of it requires:

- the organization of the library in order to facilitate the search of the components and to improve the efficiency retrieval;
- the comparison between the descriptions of the library component with the description of the new description.

The identification consists of: a syntactic comparison between a library component in RSL language and the user query one, and a semantic comparison of the library component behavior with the query specification behavior both expressed by axioms. The former description corresponds to the syntactic matching, and the latter corresponds to the semantic matching.

A very important factor to consider is the granularity levels of components. The granularity levels of components are the list of objects that belong to a RSL module. The component can modify its size from construct operations of the language to modules for big software systems. To describe and to reuse components they must be encapsulated.

It is very common in RAISE specifications to decompose the systems in levels. For example, functions in a first level are as well decomposed in subfunctions that will be calculated in second and following level modules. Thus, these two granularity levels allow users to reuse components in both levels.

On the other hand, axioms allow more precise expression and detailed relationships between two components. For example, functions like strcpy: String x String → String and strcat: String x String → String have the same functionality but it would be inadequate to substitute one for the other. If a query specification of String could be a candidate, then it would be necessary to compare the axioms of strcpy and strcat functions to conclude which of the functions has the desired behavior.

Types of Matching

A RAISE component is a set of "module structure." Furthermore a component can be related with others by means of the implement, refine and compose relationships.

Different matchings can be applied to both the component modules and the values of a scheme. We consider signature specification matching and axioms proof of the values, being the specification matching.

The signature operations matching expresses whether transformations can be applied to a pair of operations signatures.

The axiom's proof is defined by means of a logical-mathematics relationship among axioms (e.g., the imply).

The specification signature matching implies a matching between sorts and the matching of operations signature. Different kinds of operations matching are described:

Be $O_{L(\text{Library component})}$: T_{L1} x T_{L2} x .....→ T_{Lm} and
$O_{Q(\text{query})}$: T_{Q1} x T_{Q2} x .....→ T_{Qm},
two operations.

The exact matching is defined:

Exact Operation Matching $_E(O_L, O_Q)$ = ∃ a sequence of T_{LI} renaming R/R $O_L = O_Q$

The exact matching is a good starting point but very restrictive. There must be useful functions in the library so located users can adapt them by means of the reuse operators. The relaxed matchings, specialized and generalized, are defined:

Generalized Matching: $\text{Match}_G (O_L, O_Q) = O_L \geq O_Q$

where $O_L \geq O_Q$ expresses that O_L is "more general " than O_Q. It means that the argument types in O_Q are specializations of associated types in O_L.

Specialized Matching: $\text{Matchs} (O_L, O_Q) = O_L \leq O_Q$

where $O_L \leq O_Q$ expresses that O_L is "more specific" than O_Q. It means that the argument types in O_L are specializations of associated types in O_Q.

The exact, generalized and specialized operations matchings can be extended to RSL specifications signature:

S-Exact Matching
S-Exact Match $_E$ (L, Q) = ∃ a mapping A_F: $F_Q \rightarrow F_L/A_F$ is one-to-one and ∀ $O_Q \in F_Q$: $\text{Match}_f (A_F(O_Q), O_L)$

The exact matching requires that the specifications match exactly the rename.

It would be necessary for the user to construct a partial query specification that contains a subset of operations reflecting the main behavior and matching a more general specification. This kind of matching is known as generalized matching:

S-Generalized Matching
S-Match G (L, Q) = ∃ a mapping AF: FQ → FL/ ∀ OQ ∈ F_Q:
$\text{Match}_f (A_F(O_Q), O_L)$

The specialized matching matches the query specification with an RC specification more specific:

S-Specialized Matching
S-Match $_S$ (L, Q) = S-Match $_G$ (Q, L)

The signature acts like a filter that excludes obvious "no" matchings before the semantic matching. Thus, to the correct component reuse, it is necessary to check if the semantic requirements of the objective specification expressed by axioms are satisfied by a component.

ILLUSTRATIVE EXAMPLE

There are several suggested principles in creating a collection of modules to model a system:

- Each module should have only one type of interest and define functions to create, modify and observe values of the type.
- The modules should as far as possible form a hierarchy. Each module below the top one should be instantiated in only one other — its parent — as an embedded object, and its functions should only be called from its parent.

This leads naturally to a top-down style of specification and development. As we decide on the concrete type for a module, perhaps involving several components, then as long as these component types are nontrivial, we define new modules for them as children of the original.

In this section we present an example of a reusable software component classification (Mauco & George, 2000) and a specification query matching it. The Milk Production System is a reusable component belonging to the Agricultural System Infrastructure and its complete definition can be found in Mauco and George (2000).

Figure 3 shows the partial classification of the reusable component.

According to the classification, we can classify this component by the following shown in Table 2.

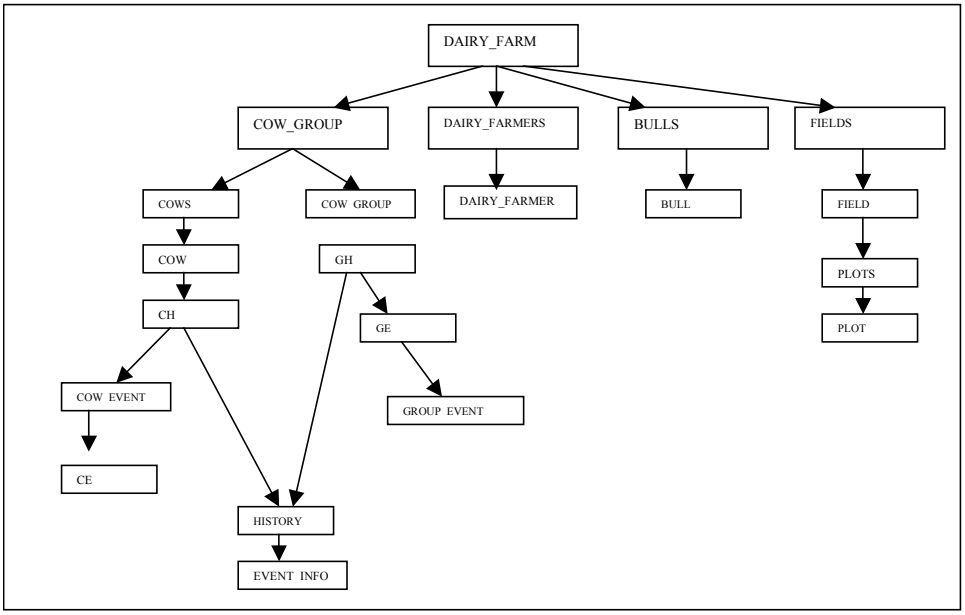
Outline of the System

Basically the aim of an Agriculture Production component is to produce farming and to obtain a rent by its use in the Agricultural System Infrastructure.

This component is divided into three main components: Farmings_Groups, Animal_Groups and Fields.

Figure 4 shows part of the Agriculture System Module structure named Agriculture.

Figure 3. Milk Production System Module Structure



Looking at the different component classifications, we can determine that Dairy_Farm is the appropriate reusable component for our requirement. When the reusable component has been identified, it is passed on to a more detailed evaluation that uses specification matching to determine reusability.

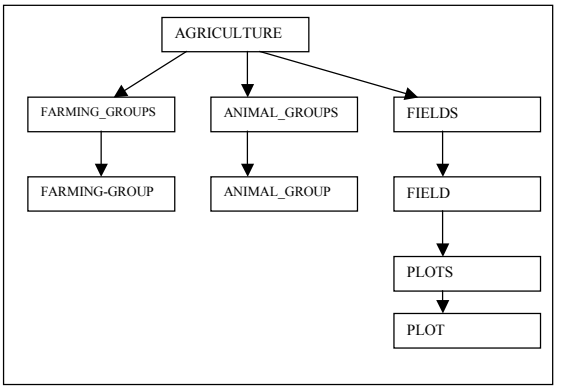
The results of specification matching determine the relationship that exists between each of the retrieved components and the requirements specification.

The process goes on with each module using the same classification for the component. It is clear that in this example there are analogous modules like Fields and Field modules, having the same name in both structures. Also, we can consider that this structure branch is the same branch in the Milk Production component. Thus,

Table 2.

(a) *kind of component:* System specification describing a Milk Production System
(b) *operations:* records the events where animals participate. E.g.: births, milking, feedings,etc.
(c) *component structure:*
schemes: {Dairy_Farm Cow_Groups, Fields, Bulls, Dairy_Farmers, Cow_Groups, Cows, Field, Bull, Dairy_Farmer ,Cow, Plots, Plot, History, Event_Info, Gropu_Event, Cow_Event....}
objects: {GH, GE, CH, CE, K, GT, D}
granularity: Dairy_Farm: {Bulls, Fields, Cow_Groups, Dairy_Farmers}
.....
(d) *relationship:* related with Meat Production System
(e) *specification style:* concrete applicative sequential

Figure 4. Agriculture System Module Structure



the process of reuse is obvious and we can talk about the exact matching of signatures. In this case, it is not necessary to apply the rename operator.

As explained previously, a search query can be based on the structural properties of the software component.

Animals_Groups in our structure will match with Cow_Groups as in Cow_Groups specification (Figure 5) and the Animals_Groups specification (Figure 6). Like the component classification, we can classify these specifications analogously:

Cow_Groups Classification

a. kind of component — Cow_Groups specification. Classify the cow groups into different types;
b. operations: list of values
   add-cow-group,
   is calf-group,
   define-range,

Figure 5. Reusable Scheme

```
scheme COW_GROUPS(CS:COWS)=
  class
   object CG: COW_GROUP
type Cow_group=GT.Group-type→ CG.Cow_group
                                    m
  value
     add-cow-group:
        GT.Group-type x Cow-groups → Cow-groups
     add-cow-group (gt,cgs) ≡
           cgs † [gt ? CG. make-cow-group()]
      pre gt ∉ cgs ∧ ~ is-calf-group(gt),
     is calf-group: GT.Group-type → Bool
     define-range: GT.Cow-id x Cow-group → Nat x Nat,
     define-calf-group: Nat x Nat x Cow-groups
                          ~→ Cow-groups
      has –produced-milk:
      GT.Group-type x D.Period x Cow-groups x
                 CS.Cows → Bool
        .....
```

Figure 6. Query Scheme

```
scheme ANIMAL_GROUPS(AS:ANIMALS)=
  class
   object AG: ANIMAL_GROUP
   typeAnimal_group=GT.Group-type→ AG.Animal_group
                                    m

  value
     add-animal-group:
        GT.Group-type x Animal-groups →
                Animal-groups
     add-animal-group (gt,ags) ≡
           ags † [gt ? AG. make-animal-group()]
     pre gt ∉ ags ∧ ~ is-calf-group(gt),
     is calf-group: GT.Group-type → Bool
     define-range: GT.Animal-id x Animal-group →
                          Nat x Nat,
     define-calf-group: Nat x Nat x Animal-groups
                          ~→ Animal-groups
     #-resources: Animal-group → Nat,

     associated-field: GT. Animal-id → Field
```

has-produced-milk;
...
c. component structure
scheme: Cow-group
object: CG
- *granularity*: {CG}
scheme: Cows
object:CS;
d. relationship: -;
e. specification style: concrete applicative sequential.

Animal-Groups Classification

a. kind of component — Animal_Groups specification. Classify the animal groups in different types and define some features of interest in Agriculture System;
b. Operations — list of values
add-animal-group,
is calf-group,
define-range,
#-resources,
associated-field, ...;
c. component structure
scheme: Animal-group
object: AG, F (Field)
- granularity: {AG,F};
d. relationship: -;
e. specification style — concrete applicative sequential.

There are several features to take into account when a reusable component has been selected to be reused. One of the most important points to analyze is whether the component has been designed to be generic or abstract or for a specific software system. Thus, in the first case, it is necessary to apply the specialization and instantiations mechanisms of the component; and, in the second case, the role of pre- and postconditions and properties expressed by axioms is crucial.

In the next paragraph, we will analyze the list of values of two specifications and the corresponding matching among them.

Both components have been designed for a specific system belonging to the same infrastructure. The first step in the domain matching consists of verifying the arity of functions. The domain of a query operation must coincide in the number of arguments with the number of argument of the operation domain of the library specification.

When we obtain the operation matchings in image and in domain argument numbers, different types of matching are applied. For example:

- Exact Matching: We verify that the sorts and the order of domain arguments of the query operation coincide with the sorts of the domain of the component operation.
  - is calf-group: GT.Group-type → Bool is a particular case because it is the same function used in both specifications and it derived from the analysis of the requirements. In this case it is not necessary to apply the rename operator.
- Specialized Matching: for query operations more general than library operations in a domain.
  - add-animal-group: GT.Group-type x Animal-groups → Animal-groups
    add-animal-group (gt,ags) ≡ ags † [gt → AG. make-animal-group()]
    pre gt ∉ ags ∧ ~ is-calf-group(gt);
  - add-cow-group: GT.Group-type x Cow-groups → Cow-groups
    add-cow-group (gt,cgs) ≡ cgs † [gt → CG. make-cow-group()]
    pre gt ∉ cgs ∧ ~ is-calf-group(gt),
    Model(Animal-groups)⊇Model (Cow-groups);
  - the same reasoning occurs with
    - define-range:GT.Animal-id x Animal-group→ Nat x Nat,
    - define-calf-group:Nat x Nat x Animal-groups~→ Animal-groups
    Model(Animal-groups)⊇Model (Cow-groups).

When the operation matching is made, we proceed with the matching of specification signatures. Also, it is possible to make the semantic verification by means of axiom proofs. This proof uses not only the axioms of the user specification but also the preconditions, giving implications among the preconditions of operations in which we make the syntactic matching.

From the example given above, which is a description of a real system situation, we observe that the signature query is more general than the signature library. Thus, the library is an implementation of the query because its signature match and Mod((Animal-groups)⊇Mod(Cow-groups) and Mod(Cow-groups) is the class of models in which the axioms of Animal-groups are satisfied.

In particular, we can observe both in query scheme specification and in library scheme specification specific functions do not have meaning in the other specification. The manipulation of these operations will be by means of specification building operator Hide.

CONCLUSIONS AND FUTURE WORK

In this chapter a strategy to classify and select a reusable component has been presented. We defined the RC model for the description of reusable components and a transformational process with reuse from RSL specifications to code, which is the main direction for our current work.

Here, we presented only one variety of matches for signatures and specifications of functions and models, but we have developed various matches to show how they are used for a variety of applications. Also, the component classification provides the starting point for the analysis of the component type.

Our goal was to solve a problem that is a weakness of RAISE formal method. Therefore, the proposal was to apply not only the software component reuse but also a domain specification reuse, i.e., in the confines of the domain engineering.

REFERENCES

Beltaifa, R., & Moore, R. (2001, May). *A software reuse infrastructure for an efficient reuse practice* (Tech. Rep. No. 230). Retrieved from http://www.iist.unu.edu.

Bjorner, D. (2000). *Software Engineering: A New Approach*. Lecture Notes, Technical University of Denmark.

CarmaMcClure. (1995). Software reuse — recent papers. Retrieved 2002 from http://www.reusability.com/papers.

Chen, Y., & Cheng, B. (1997, March). Formally specifying and analyzing architectural and functional properties of components for reuse. *Proceedings of 8th Annual Workshop on Software Reuse,* Columbus, OH.

Felice, L., Leonardi, C., Favre, L., & Mauco, V. (2001, May). Enhancing a rigorous reuse process with natural language requirement specifications. *Proceedings of 2001 Information Resources Management Association International Conference* (pp. 271-275) Toronto, Canada.

George, C. (2001). *RAISE tools user guide*. (Tech. Rep. No. 227). Retrieved from http://www.iist.unu.edu.

George, C. (2002). *Introduction to RAISE*. (Tech. Rep. No. 223). Retrieved April 2002 from http://www.iist.unu.edu.

George, C., Haff, P., Havelund, K., Haxthausen, A., Milne, R., Nielsen, C., et al. (1992). *The RAISE specification language*. Prentice Hall.

George, C., Haxthausen, A., Hughes, S., Milne, R., Prehn, S., & Pedersen, J. (1995). *The RAISE Development Method*. BCS Practitioner Series. Denmark: Prentice Hall.

Helm, R. & Maarek, Y. S. (1991). Integrating information retrieval and domain specific approach for browsing and retrieval in object-oriented class libraries. *Proceedings of OOSPLA91,*. 47-61.

Hennicker, R., & Wirsing, M. (1992). *A Formal Method for the Systematic Reuse of Specifications Components*. Lecture Notes in Computer Science 544, Springer-Verlag.

Krueger, C. (1992, June). Software Reuse. *ACM Computing Surveys*, *24*(2).

Maarek, Y. S., Berry, D. M., & Kaiser, G. E. (1991). An information retrieval

approach for automatic constructing software libraries. *IEEE Trans. Software Engineering, 17*(8), 800-813.

Mauco, V., & George, C. (2000). Using requirements engineering to derive a formal specification. (Tech. Rep. No. 223). Retrieved December 2000 from http://www.iist.unu.edu.

Penix, J. (1998). Unpublished doctoral dissertation, University of Cincinnati, Cincinnati, OH.

Zaremski, A., & Wing, J. (1997). Specification matching of software components. *ACM Transactions on Software Engineering and Methodology (TOSEM), 6*(4), 333-369.

Chapter VII

The Application of FOOM Methodology to IFIP Conference Case Study

Judith Kabeli
Ben-Gurion University, Israel

Peretz Shoval
Ben-Gurion University, Israel

ABSTRACT

FOOM (Functional and Object-Oriented Methodology) is an integrated methodology for information systems' analysis and design, which combines two essential software-engineering paradigms: the functional/data approach (or process-oriented) and the object-oriented (OO) approach. Having applied FOOM in a variety of domains, this chapter presents the application of the methodology to the specification of the IFIP Conference system. We focus on the analysis and design phases. FOOM-analysis phase includes data modeling and functional analysis activities and produces an initial Class Diagram and a hierarchy of OO data flow diagrams (OO-DFDs). The products of the design phase include: (a) a complete class diagram; (b) object classes for the menus,

forms and reports and (c) a behavior schema, which consists of detailed descriptions of the methods and the application transactions, expressed in pseudocode and message diagrams.

INTRODUCTION

This chapter provides a brief description of FOOM methodology, along with its application to IFIP Conference case study. A more detailed description of FOOM can be found in Shoval and Kabeli (2001). The description of the IFIP Conference case study is provided in Mathiasssen, Munk-Madsen, Axel Nielsen and Stage (2000), who demonstrate the application of their OOA&D to the IFIP case. The objective is to show how the FOOM methodology, which combines the process and object-oriented paradigms, is suitable for analyzing and designing business-oriented information systems.

BACKGROUND

Many paradigms for system analysis and design have been proposed over the years. Early approaches have advocated the functional approach (DeMarco, 1978; Yourdon & Constantine, 1979). The development of OO programming languages gave rise to a new approach that maintains that in order to develop information systems in such languages, it is recommended to perform OO analysis and design. Many OO methodologies were developed (e.g., Booch, 1991; Coad & Yourdon, 1990, 1991; Jacobson, 1992; Martin & Odell, 1992; Rumbaugh, Blaha, Premerlani, Eddy & Lorensen, 1991; Shlaer & Mellor, 1988, 1992; Wirfs-Brock, Wilkerson & Wiener, 1990), and the area is still evolving. The multiplicity of diagram types in the OO approach has been a major motivation for developing the Unified Modeling Language (UML) (see Booch, Rumbaugh & Jacobson, 1999; Clee & Tepfenhart, 1997; Larman, 1998; Maciaszek, 2001; UML Rose, 1998). UML was developed in order to produce a standard (unified) modeling language. It consists of several types of diagrams with well-defined semantics and syntax, which enables the presentation of a system from different point of views.

Information systems development is a multiphase process in which the analysis and design are of primary importance. Therefore it is vital to examine which approaches and methods are appropriate to perform each of these phases. On the one hand, those who adopt the OO approach claim that using data abstraction at the analysis phase, producing a model of reality by means of classes, is preferable to producing a functional model, because the real world consists of objects. However, as far as we know, no such study has shown that the OO approach is more effective than the functional/data approach in the development of business-oriented information systems.

OO methodologies tend to neglect the functionality aspect of system analysis and do not clearly show how to integrate the system functions, or transactions, with the object schema. One sometimes gets the impression that the functionality of the system is expressed solely by means of methods that are encapsulated within objects, thus disregarding functional requirements that cannot be met by simple methods. In contrast, based on vast experience in performing functional analyses with DFDs, we have encountered no problems with them as a means to express the functionality of the system; the only problem was how to continue from them to the next phases of development.

In our opinion, since process and object are both fundamental building blocks of reality, the analysis phase must cover both the functional and the data aspects. The functional approach, using DFDs, is suitable for describing the functionality of the system, while Class Diagrams are suitable for modeling the data structure. Since the OO approach is the one most appropriate for performing the design phase, we suggest performing data modeling by creating an initial Class Diagram. It seems more effective to produce an initial Class Diagram at the analysis phase and then to use it as input in the design phase. (The term initial Class Diagram will be clarified later on.)

For the design phase it is crucial to provide a smooth and seamless transition to system implementation. Since there is an agreement on the advantages of OO programming, it is also desirable to design the system with methods and tools that belong to the OO family. Therefore, we conclude that, in order to perform each of those development phases with its most appropriate method, there is a need to integrate the OO and functional-oriented approaches.

Dori's Object-Process Methodology (OPM) (Dori, 1996, 2001) integrates the two approaches. It utilizes a single graphic tool Object-Process Diagram (OPD) at all development phases. Since OPD defines a new notation that combines DFD and OO diagrams, it includes a great number of symbols and rules. It seems to us that such diagrams are not easy to construct and comprehend for large-scale systems and that reality has to be modeled by means of simple notations, which are easy to learn, comprehend and utilize. A single hybrid notation, like OPM, must be very rich in order to elicit all these points of view, thus leading to a complex, perhaps distorted, model of reality. On the other hand, multiplicity of models and corresponding diagramming tools, as found in UML, may be too complicated. Too many diagram types (even standard ones) can hamper coherent understanding and lead to the production of erroneous models and systems.

We are looking for an optimal way to integrate the process and object approaches. Since users express their information needs in a functional and data manner, and not by means of an object structure and behavior, an appropriate (natural) method to carry out the analysis task is by functional/data analysis. On the other hand, the design should be made through the OO approach to facilitate the transition of the design to OO programming, which has proved to be a better approach to implementing software. The integration of the two approaches is made possible

because it applies principles and techniques taken from the ADISSA methodology, especially transaction designs. A **transaction** is a process that supports a user who performs a business function and is triggered as a result of an event. A transaction in a DFD consists of elementary functions that are chained through data-flows and data-stores and external-entities that are connected to those functions. The transaction designs enable the transition from functional analysis DFDs to an OO model that consists of object and behavior schemas. [More details can be found in the ADISSA references Shoval (1988, 1990, 1991).]

FOOM METHODOLOGY

The Analysis Phase

The analysis phase consists of two main activities: data modeling and functional modeling. The products of this stage are a data model, in the form of an initial Class Diagram, and a functional model, in the form of hierarchical OO-DFDs (supported by a data-dictionary).

The initial Class Diagram consists of "data" classes (also termed "entity" classes), namely classes that are derived from the application requirements and contain "real-world" data (Other classes will be added at the design stage.). Each class includes attributes of various types (e.g., atomic, multi-valued and tupels of attributes, keys, sets, and reference attributes). Association types between classes include "regular" (namely 1:1, 1:N and M:N) relationships, with proper cardinalities, generalization-specialization (is-a or inheritance) links between super- and sub-classes, and aggregation-participation (is-part-of) links. Note that in our model, relationships are signified not only by links between respective classes but also by reference attributes to those classes. However, the initial Class Diagram does not include methods; these will be added at the design phase. The initial class diagram of the IFIP Conference case study is shown in Figure 1.

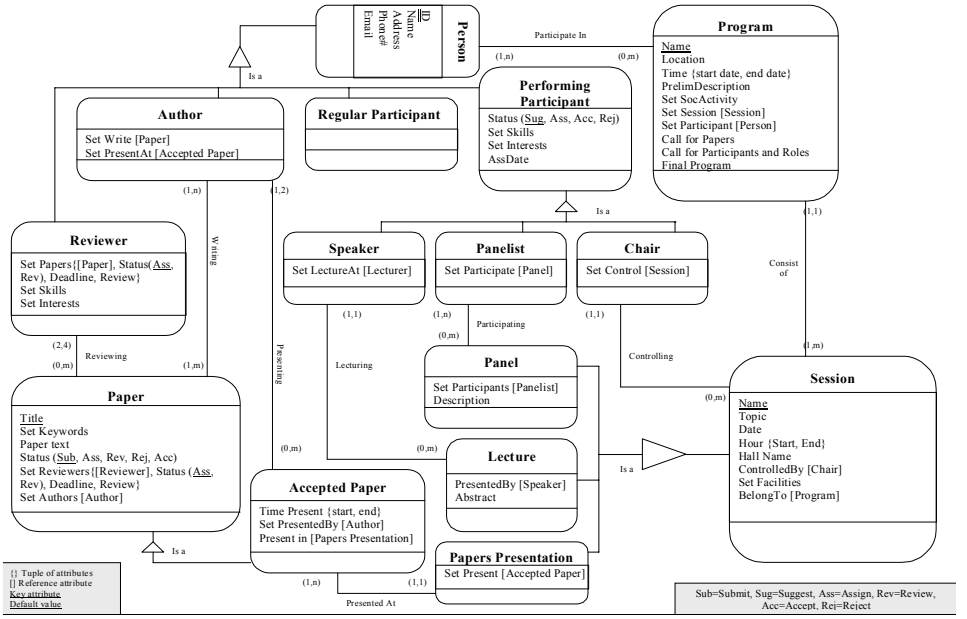
The OO-DFDs specify the functional requirements of the system. Each OO-DFD consists of general or elementary functions, external entities (mostly user-entities, but also time and real-time entities), object-classes (instead of the "traditional" data-stores) and the data flows among them. Figures 2 through 4 show three OO-DFD examples. Note that a general function is represented as a double circle, meaning that its subfunctions are described in a separate sub-OO-DFD. Classes within the OO-DFDs correspond to classes that exist in the initial Class Diagram.

The Design Phase

Defining Basic Methods

Basic methods of classes are defined according to the initial Class Diagram. We distinguish between two types of basic methods: elementary methods and relationship/integrity methods.

Figure 1. Initial Class Diagram of IFIP Conference



Elementary methods include: (a) construct (add) object; (b) delete (drop) object; (c) get (find) object and (d) set (change) attributes of object. Elementary methods actually belong to a general (basic) class from which all the classes inherit.

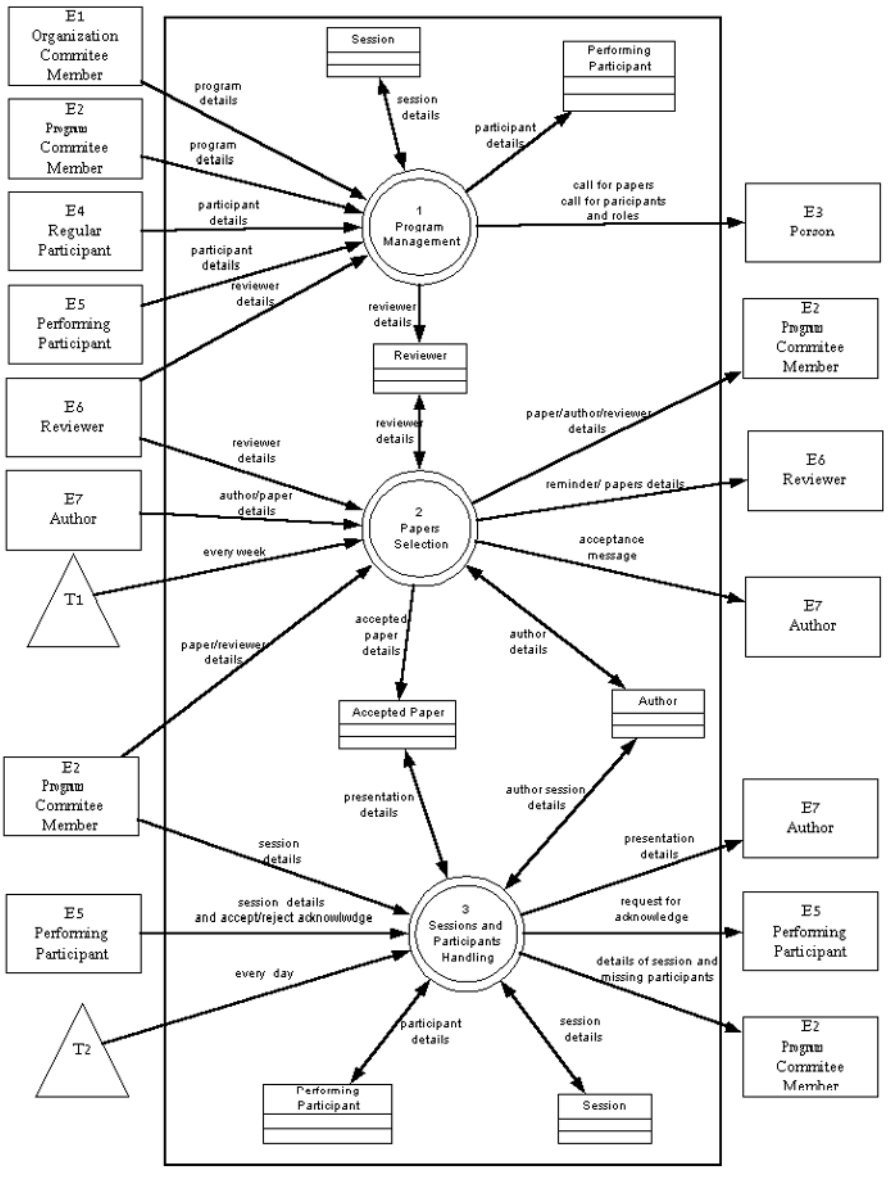
Relationship/integrity methods are derived from structural relationships between classes. They are intended to perform referential integrity checks, depending on the relationship types between the classes and on cardinality constraints on those relationships. For each relationship, which is also expressed in terms of reference attributes, the involved classes include appropriate integrity methods, which will fire whenever an object of a respective class is added, deleted or changed. [Note that additional, application-specific methods will be added in the stage of behavior design (see Design of the Behavior System).]

Top-Level Design of the Application Transactions

This stage is performed according to ADISSA methodology, where the application transactions are derived from OO-DFDs. [For more details, see Shoval (1988).] Note that here the transactions include **classes** rather than data-stores.

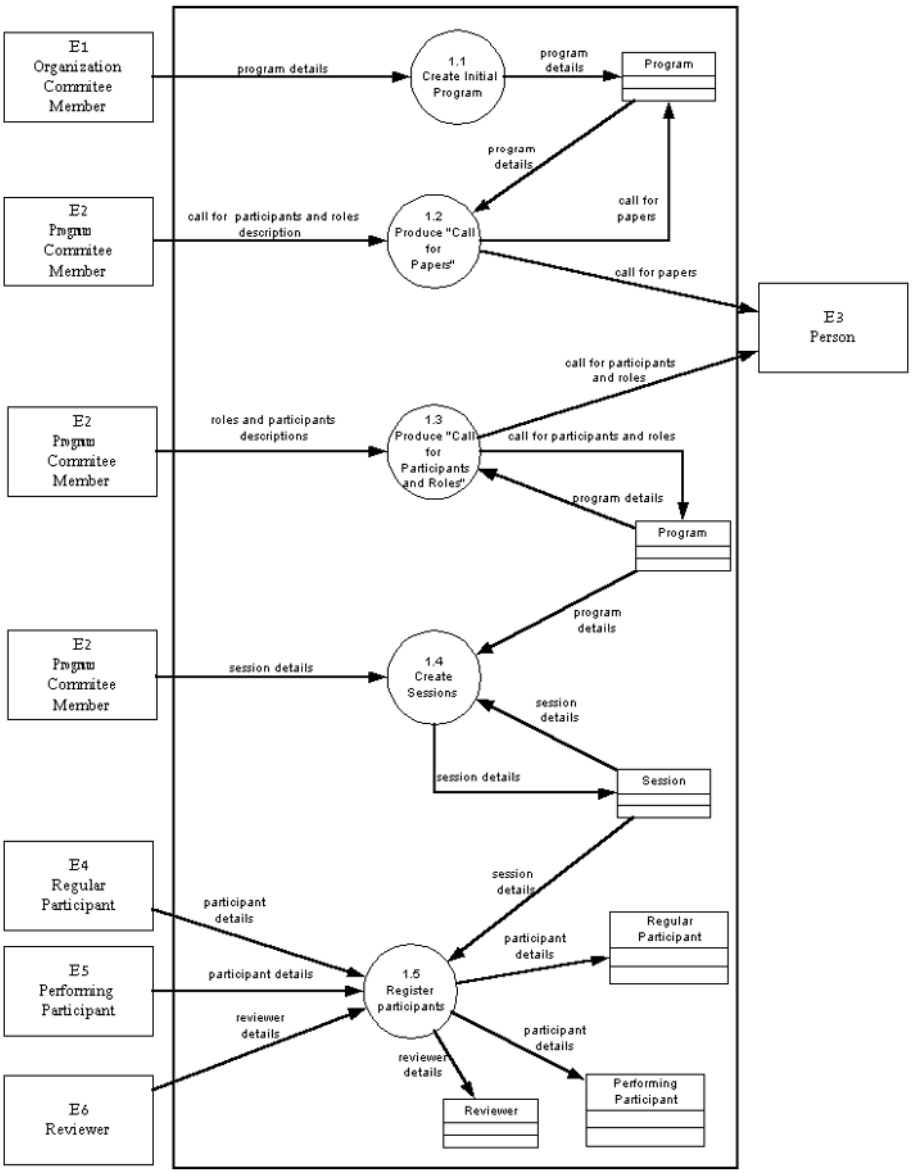
The products of this stage include transaction diagrams, which are extracted from the OO-DFDs, top-level descriptions of the transactions and a new class — "transactions class." This virtual class will not contain objects — only the transaction methods (as will be elaborated in Design of the Behavior System).

Figure 2. OO-DFD-0: The IFIP Conference



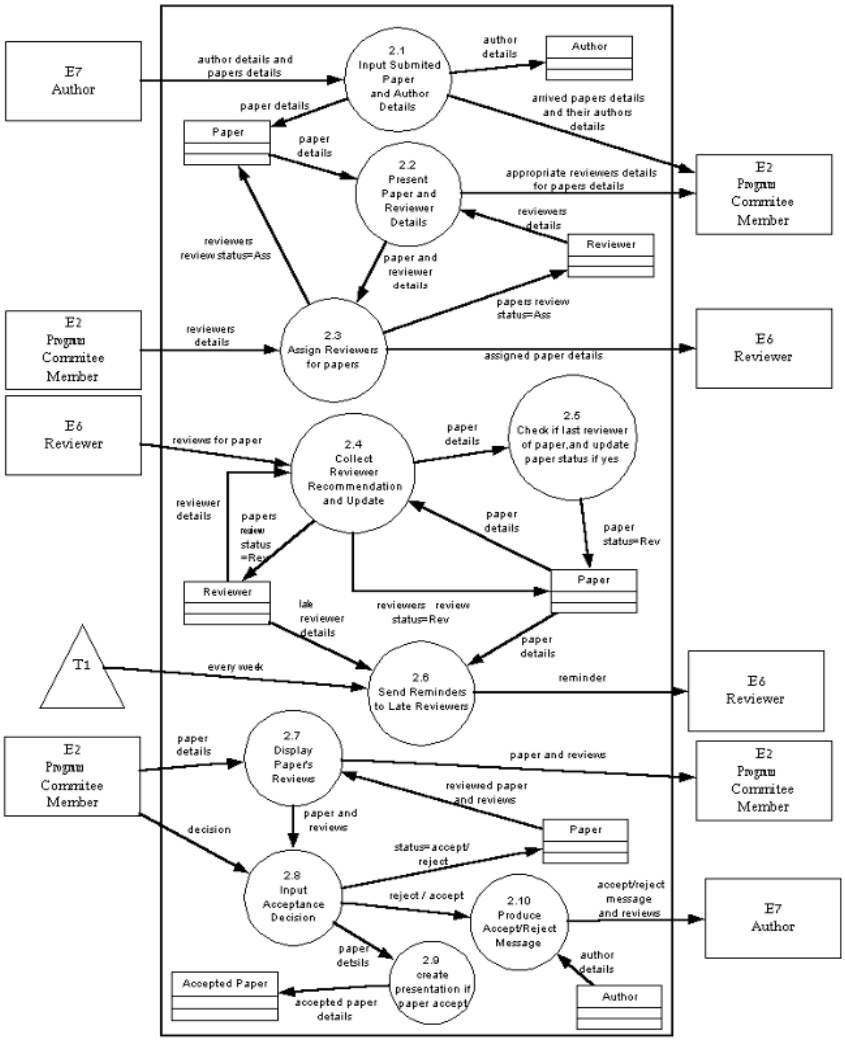
A top-level transaction description is provided in a structured language (e.g., pseudo-code or flowchart), and it refers to all components of the transaction: every data-flow from or to an external entity is translated to an "Input from ..." or "Output to ..." line; every data-flow from or to a class is translated to a "Read from ..." or "Write to ..." line; every data flow between two functions translates to a "Move from

FIgure 3. OO-DFD-1: Program Management



... to ..." line and every function in the transaction translates into an "Execute function..." line. The process logic of the transaction is expressed by standard structured programming constructs (e.g., if... then ... else ...; do-while ...). The analyst and the user, who presents the application requirements, determine the process logic of each transaction. This cannot be deducted "automatically" from the transaction diagrams alone, because a given diagram can be interpreted in different ways, and it is up to the user to determine the proper interpretation.

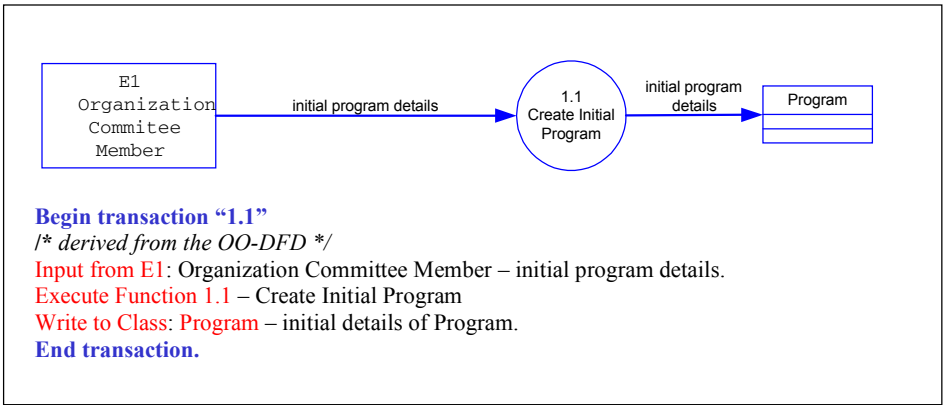
Figure 4. OO-DFD-2: Papers Selection



The top-level transaction descriptions will be used in further stages of design, namely input/output design and behavior design, to provide detailed descriptions of the application-specific class methods (in addition to the basic methods), as well as the application programs.

Figures 5 and 6 show examples of two transaction diagrams. One (Figure 5) is a "simple" transaction consisting of one elementary function, one class and one user entity; this transaction is derived from OO-DFD-1 (Figure 3). The other (Figure 6) is a more "complex" transaction consisting of several elementary functions, classes and user-entities; this transaction is derived from OO-DFD-2 (Figure 4).

Figure 5. Top-Level Description of Transaction "1.1"



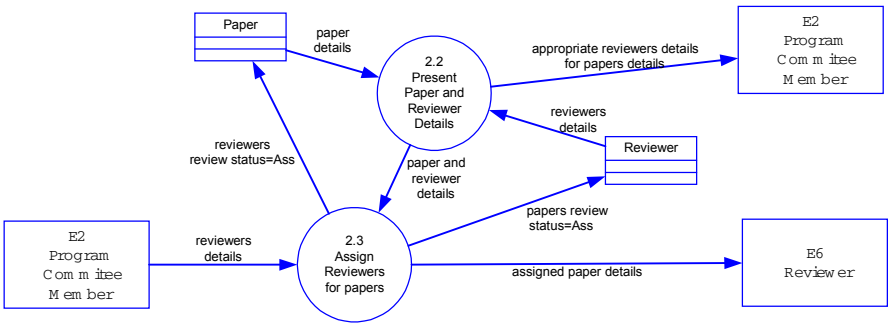
Design of the Interface: The Menus Class

This stage is performed following the ADISSA methodology (Shoval, 1988, 1990). A menu-tree interface is derived in a semialgorithmic way from the hierarchy of OO-DFDs. Note the correspondence of the menus and menu items to the respective general functions and elementary functions in the OO-DFDs. The menu tree is translated into a new class — the "Menus class." The instances (objects) of the Menus class are the individual menus, and the attribute values of each object are the menu items. Note that some of the selections within a given menu may call (trigger) other menu objects, signified by S (selection) while other selections may trigger transactions, signified by T. Transactions will be implemented as methods of the Transactions class (as will be detailed later). Hence, at run time, a user who interacts with the menu of the system actually works with a certain menu object. He/she may select a menu item that will cause the presentation of another menu object, or invoke a transaction, which is a method of the Transactions class. Figure 7 shows the Menus class and some of its objects.

Design of the Inputs and Outputs: The Forms and Reports Classes

This stage too is performed according to ADISSA methodology and is based on the input and output lines appearing in each of the transaction descriptions. Hence, for each "Input from ..." line, an input screen/form will be designed, and for each "Output to ..." line an output screen/report will be designed. Depending on the process logic of each transaction, some or all of its input or output screens may be combined. Eventually two new classes are added to the Class Diagram: "Forms class" for the inputs and "Reports class" for the outputs. Obviously the instances (objects) of each of these class types are the input screens and output screens/reports, respectively. Such classes are usually defined in OO-programming languages and can be reused.

Figure 6. Top-Level Description of Transaction "2.2-2.3"



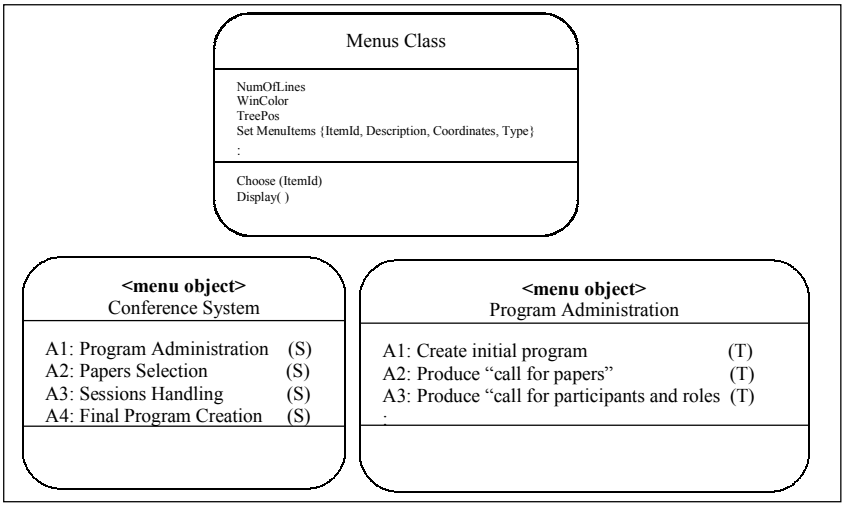
Begin transaction "2.2-2.3"
For each paper that is not assigned:
{Read from Class: Paper - paper details
  For each reviewer
  {Read from class: Reviewer
    Execute Function 2.2 – Present Paper and reviewer details
    Output to E2: Program Committee Member - appropriate reviewers details for papers
  }
 move to Function 2.3
 Input from E2: Program Committee Member – selected reviewers details for the paper
 For each selected reviewer
 { Execute Function 2.3 – Assign reviewers for papers
   Write to Class : Paper – reviewers details and update paper status to Assigned
   Write to Class : Reviewer – paper details and update reviewer status to Assigned
   Output to E6: Reviewer - assigned paper details
 }
}
End transaction.

Design of the System Behavior

In this stage we have to convert the top-level descriptions of the transactions into detailed descriptions of the application programs and application-specific methods. A detailed description of a transaction may consist of procedures that can be handled as a certain procedure may be identified as a basic method of some class. Another procedure may be defined as a new, application-specific method to be attached to a proper class. Remaining procedures (which are not identified as basic methods or defined as specific methods) will become a Transactions method, which is actually the "main" part of the transaction's program. Hence, every transaction is represented as a Transactions method of the Transaction class. Once triggered by the user via proper menus selections, it may call (namely, send messages to) other methods of respective classes, depending on the process logic of the transaction.

We can categorize the application transactions according to their complexity—depending on how many classes and methods they refer to. For example, a simple transaction (e.g., one that finds a certain object and displays its state or that updates attributes of an object) may be implemented as a small procedure (namely a Transactions method) that simply sends a message to a basic method of a certain

Figure 7. The Menus Class and Objects



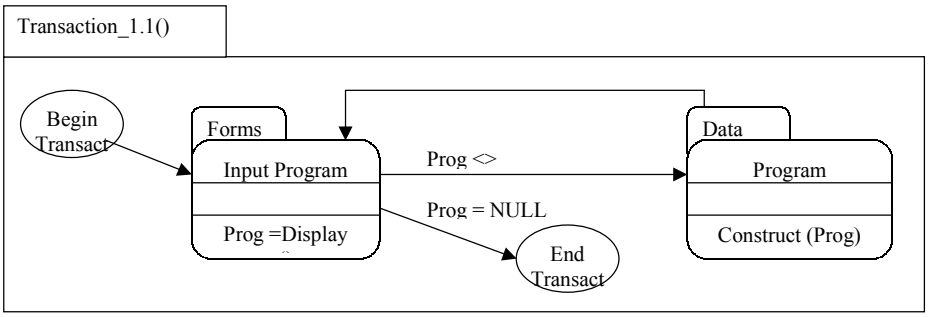
class. A complex transaction (e.g., one that finds and displays objects that belong to different classes, that updates objects that belong to different classes or that both retrieve and update various objects) may be implemented as a more complex procedure that sends messages to basic methods and specific methods of various classes. Generally an application may consist of many transactions with different levels of complexity. Note that at run time, when a user wants to "run" any transaction, he/she actually approaches the Menus class and makes proper item selections within the menu objects until a menu item that actually fires the desired Transactions method is selected. From here, the execution of a transaction depends on the process logic of the Transactions method, and the other methods it calls.

The detailed description of a transaction is expressed in two complementing forms: pseudo-code and message diagram. A pseudo-code (Figure 8) is a structured description that details the process logic of the Transactions method as well as any other class method. The transition from a top-level description of a transaction to its detailed pseudo-code description is done as follows: Every "Input from ..." and "Output to ..." line in the top-level description is translated to a message calling an

Figure 8. Pseudo-Code of Transaction_1.1

```
Menu-0.Display; Menu-0.Choose(A1)
Transaction 1.1()
{
Program=Input_Program.Display   /* Input_Program is a form, which enable the user to fills in details
on Program. When the user finish to fill the form it returns the filled Program object */
Program.Add(Program)
}
```

Figure 9. Message Diagram of Transaction_1.1



appropriate method of the Forms/Reports class. Every "Read from ..." or "Write to ..." line is translated to a message calling a basic method (e.g., "Get," "Const," "Set," and "Del") of the appropriate class. Every "Execute-Function ..." line is translated to messages calling one or more basic methods of certain classes, to new, specific methods that will be attached to proper classes, or to procedures that remain as part of the Transactions method.

A Message Diagram (Figure 9) shows the classes, methods and messages included in a transaction, in the order of their execution. A message diagram is actually a partial class diagram that shows only the classes involved in the transaction (including Data, Menus, Forms, Reports and Transactions classes), the method names (and parameters) included in that transaction, and message links from calling to called classes. Message diagrams supplement the pseudo-code descriptions of transactions.

To summarize, the products of the design phase are: (a) a complete class diagram, including Data, Menus, Forms, Reports and Transactions classes, each with various attribute types and method names (and parameters), and various associations among the classes; (b) detailed menu objects of the Menus class, each menu listing its items (selections); (c) detailed form and report objects of the Forms and Reports classes, each detailing its titles and data fields; (d) detailed transactions descriptions in pseudo-code and (e) message diagrams at least for nontrivial transactions.

FUTURE RESEARCH

Our further research and development agenda includes development of a set of CASE tools to support the methodology and evaluation of the methodology by means of experimental comparisons with other methodologies on various dimensions, e.g., comprehension of schemas by users, quality (i.e., correctness) of designed schemas, ease of learning the methods, etc.

CONCLUSION

The advantages of FOOM are that system analysis (i.e., specification of user requirements) is performed in functional terms via OO-DFDs — a natural way for users to express their information needs — and in data terms via an initial class diagram. System design follows the analysis and uses its products. The class diagram is augmented with a Menus class, which is derived from the menu-tree that was designed earlier from the OO-DFDs, and with Inputs and Outputs classes, which are derived from the input forms and the outputs of the system (earlier products of the design stage). The class methods and the application programs are generated from the transaction descriptions, which are also derived from the OO-DFDs.

REFERENCES

Booch, G. (1991). *Object-Oriented design with applications.* Benjamin/Cummings.

Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *The unified modeling language user guide*. Longman, MA: Addison-Wesley.

Clee, R., & Tepfenhart, W. (1997). *UML and C++: A practical guide to Object-Oriented development*. Englewood Cliffs, NJ: Prentice Hall.

Coad, P., & Yourdon, E. (1990). *Object-Oriented analysis.* Englewood Cliffs, NJ: Prentice Hall.

Coad, P., & Yourdon, E. (1991). *Object-Oriented design.* Englewood Cliffs, NJ: Prentice Hall.

DeMarco, T. (1978). *Structured analysis and system specification*. New York: Yourdon Press.

Dori, D. (1996). Object-Process Methodology: the analysis phase. *Proceedings of TOOLS USA'96*.

Dori, D. (2001). Object-Process Methodology applied to modeling credit card transactions. *Journal of Database Management, 12*(1), 4-14.

Jacobson, I. (1992). *Object-Oriented software engineering: A use case driven approach*. ACM Press.

Larman, C. (1998). *Applying UML and patterns — An introduction to Object Oriented analysis and design.* Upper Saddle River, NJ: Prentice Hall.

Maciaszek, L.A. (2001). *Requirements analysis and system design — Developing information systems with UML*. Addison-Wesley.

Martin, J., & Odell, J. (1992). *Object-Oriented analysis & design*. Englewood Cliffs, NJ: Prentice Hall.

Mathiasssen, L., Munk-Madsen, A., Axel Nielsen. P., & Stage, J. (2000). *Object Oriented analysis and design.* Aalborg, Denmark: Marko Publishing ApS.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorensen, W. (1991). *Object-Oriented modeling and design*. Englewood Cliffs, NJ: Prentice Hall.

Shlaer, S., & Mellor, S. (1988). *Object-Oriented analysis: Modeling the world in data*. Englewood Cliffs, NJ: Yourdon Press.

Shlaer, S., & Mellor S. (1992). *Object life cycles: Modeling the world in states*. Englewood Cliffs, NJ: Yourdon Press.

Shoval, P. (1988). ADISSA: Architectural design of information systems based on structured analysis. *Information System*, *13*(2), 193-210.

Shoval, P. (1990). Functional design of a menu-tree interface within structured system development. *Int'l Journal of Man-Machine Studies*, *33,* 537-556.

Shoval, P. (1991). An integrated methodology for functional analysis, process design and database design. *Information Systems*, *16*(1), 49-64.

Shoval, P., & Kabeli, J. (2001). FOOM: Functional- and Object-Oriented analysis & design of information systems — An integrated methodology. *Journal of Database Management, 12*(1), 15-25.

UML Rose (1998). Unified Modeling Language, Vol. 4. Retrieved from http://www.rational.com.

Wirfs-Brock, R., Wilkerson, B., & Wiener, L. (1990). *Designing Object-Oriented software*. Englewood Cliffs, NJ: Prentice Hall.

Yourdon, Y., & Constantine, L.L. (1979). *Structured design*. Englewood Cliffs, NJ: Prentice Hall.

Section II

Managing Software Projects

Chapter VIII

A Quantitative Risk Assessment Model for the Management of Software Projects

Dan Shoemaker
University of Detroit Mercy, USA

ABSTRACT

This chapter presents a comprehensive quantitative management model for information technology (IT). It is assessment based and can be easily implemented without imposing an unacceptable organizational-change solution. It supplies detailed information about the functioning of processes, which will allow managers to both effectively oversee operations, as well as assess their prospective and ongoing risks of execution.

INTRODUCTION

The first point that must be understood is that quantitative management is not process improvement. The professional consensus is that the proper role of

quantitative management is to insure the stability of software processes (Paulk, 1999). According to Paulk (1999), quantification makes the process "repeatable." That is a critical requirement because an organization that does not embody repeatable outcomes cannot adequately gauge its effort and cannot estimate the time or cost of its products (Humphrey & Watts, 1994). With repeatable processes, the organization can plan its work and monitor its projects. Thus, according to Paulk (2000), quantifying an undefined and ad-hoc process leads to decreased cost of production. More importantly, quality cost, and schedule are predictable (Humphrey & Watts, 1994).

Quantitative management fulfills two fundamental requirements of IT governance's best practice: the necessity to foster common understanding of the process and the responsibility to evaluate performance. Embodied within a strategic infrastructure it lets an organization "strategically align" its IT processes with its business goals, as well as to evaluate and economically justify each of its projects on a risk-adjusted basis. This assures that the overall mix of projects will best utilize the company's resources and special abilities.

THE PROBLEM

Because it supports the execution of stable repeatable processes, quantitative management looks like the best answer to any concern about efficient operation. The problem lies in its consistent application. When IT processes are not implemented or measured as consistently as they should be, an element of unacceptable variability is induced. This is the most common complaint when arguing that quantitative management cannot be applied to software (Ould, 1996). In fact, according to Ould (1996), the only significant roadblock to an organization achieving a quantitative management capability lies in insuring reliable communication: "It is crucial to have a fully defined process and understand the context of the data when doing cross-project comparisons." Thus, according to Ould (1996), businesses interested in implementing a successful quantitative management capability must concentrate on four "understanding" factors:

1. Universally understood and accepted operational definitions;
2. Clear and unambiguous (business) contextual definitions and associations;
3. The ability to trace from data back to that original context;
4. Consistent measurement of stable well-defined organizational variables.

So a single practical control framework is an absolute necessity. That is because definition entails the consistent identification of the elements that constitute the entity under study (ISACA, 2000). The problem with IT is that most of its operational variables are intangible, continuously changing or widely dispersed. For instance, the range of capital assets that should be accounted for in an ordinary IT operation span the gamut from facilities, personnel, hardware/system assets, software applications

to such intangibles as information and data, interfaces, business relationships and agreements (ISACA, 2000). With the exception of personnel, hardware and facilities, none of these is easily describable, let alone measurable. The organization can rarely account for the exact status of its software applications, enumerate its information and data assets or precisely describe its organizational interfaces and agreements. This is unfortunate because the information outcomes of technical work, across organizational boundaries as defined by agreement, are the basic mission and primary justification for IT investment (ISACA, 2000).

Moreover, without some kind of objective information based system, there is almost a direct correlation between the level of experience of the individual manager and project success. As a result, according to Jones and Capers (1997), inexperienced or inadequately trained managers are noted with distressing frequency on canceled projects and projects that experience cost overruns and missed schedules. Inadequate management training is also commonly associated with the problems of low productivity, low quality, and, of course, management malpractice. According to Humphrey and Watts (1994), without prior experience, it is practically impossible to assess the prospects of a prospective project or monitor its execution once the commitment has been made.

THE SOLUTION: STANDARD BEST PRACTICE

The alternative to individual virtuosity is systematization, which in IT is based on universally understood "best practice." A body of knowledge in best practice has always existed within the industry. This is captured and promulgated in a wide range of professional standards. These standards create and maintain the consistent policy and procedure framework necessary to satisfy Ould's (1996) requirements for achieving a successful quantitative management capability.

The essence of successful day-to-day control of the IT function lies in the ability to measure performance. Practically, managers must have precise insight into the operation of their areas in order to do this effectively. Yet managers never know as much about what is going on as the technical people they oversee. But individually, they are the ones who are responsible for the ultimate success or failure of each product (Humphrey & Watts, 1994). As Jones and Capers (1997) point out, without prior experience it is practically impossible to estimate in advance the prospects for success or failure of a software project, or to monitor its execution once that commitment has been made. This is particularly true where the requirements are highly complex. According to an exhaustive study of 8,000 software projects, where projects failed the most common cause cited was a lack of project management (execution) and an inability to oversee project activity on the part of the project manager (KPMG, 2001).

This leads to the inescapable conclusion that any organization, large or small, simple or complicated, functions better with a formally defined standard-based

management infrastructure that will insure that the organization's people, equipment and financial resources are utilized effectively in the fulfillment of its business goals. That is why the quantitative management aims of level four CMM are so attractive to software managers. Those KPAs allow them to use the systematic data provided by the key practices installed at that level (and the prior two) to evaluate potential commitments and monitor performance as the project unfolds. This in turn helps managers to identify and overcome the inevitable problems in production as they occur and minimize the risks of project failure.

Nevertheless, the problem with level four is that the prior two levels force the organization to change in ways that are often unacceptable to the people doing the work. In fact, one of the primary stumbling blocks to the implementation of any externally imposed process improvement framework (be it CMM or ISO 9000) is that IT workers must adjust their current (and sometimes highly valued) work habits to meet the model requirements.

While the methodology we are about to discuss achieves the same purposes as level four CMM (to improve organizational performance and increase productivity using focused management data), it is based on assessment rather than change. Thus, instead of being forced to follow a staged, lockstep implementation scheme that can require considerable behavior change and generate unproductive resistance, organizational capability can be evolved within the unique culture and norms of the business itself.

ASSESSMENT-BASED RISK ESTIMATION: A SHORTCUT TO LEVEL FOUR

This methodology is based in principle on the determination of the capability of a given set of required processes. It provides information about the effectiveness of each of these processes in meeting business goals, whether those have been set for a project, or the organization as a whole. The primary difference between this model and the way CMM Level Four defines capability is that our model assesses each process directly (based on a set of management attributes similar to the common features of CMM) to find out how capable it is. As a result this approach does not require an organizational change. Instead it simply provides specific information about the functioning and effectiveness of the target processes. It is assumed that managers can then use that information to evolve each process to a higher level of productivity and efficiency within the context of the business culture. The key element to keep in mind is that this information is obtained in a noninvasive way (e.g., one that is acceptable to the organizational culture as a whole).

The approach centers on the assessment of the level of adherence to commonly understood best practice. Over the past several years, it has become manifestly clear that the only reference on which to base such an assessment is the lifecycle process definitions contained in the ISO 12207 (or IEEE 12207.0) international standard (the

best evidence of that might be the mass harmonization of the IEEE Software Engineering Standards Collection to 12207 starting in 1995 in Moore (1998)). The complete set of defined processes derived from 12207 constitutes a coherent definition of best practice in software work.

Consequently any assessment that is carried out using this standard as a point of reference provides a picture of current status with respect to best practice, as well as a profile that can be used as a roadmap to achieve business goals (and evolve the desired level of capability for each process). Functionally this assumes that a correct organization is one that performs the proper (to its purpose) primary processes of the standard along with all of the necessary supporting and organizational processes independent of intricate phasing (as is the case with CMM). This means that information derived from an assessment, such as we are about to describe, can help managers to both prioritize and plan for improvements in the functioning of each process based on the realities and constraints of their business situation (Shoemaker, 2001).

OVERVIEW: THE ASSESSMENT PROCESS

The primary purpose of this assessment is to develop information about the ongoing status of the organization's processes with the aim of better resource deployment and utilization. The model supports strategic management in that purpose by determining the current and potential capability of a software process. It supports project management by identifying priority processes for further development. In effect it pinpoints the potential risks associated with a given required capability. This enables managers to put in place appropriate controls for risk containment. It also establishes a quantified basis for evaluating project requirements against current unit capability.

This approach provides a structured methodology for the ongoing assessment of software processes. It addresses the adequacy of the **execution** of the process as well as the capability of the **management** of the process. It allows the organization to factor in the context in which the assessed process operates for both of these dimensions. The assessment model relates the practices and capabilities of the process instance to its defined purpose[1]. In that respect the ratings of a given process should be repeatable by different qualified assessors. The assessment process compares an organizational unit's processes against a common reference model. As we said earlier, this is most commonly based on the lifecycle process definitions provided by the ISO 12207 Standard (in the U.S. IEEE 12207.0) or alternatively the process reference model contained in ISO 15504, Part 2 (Base Practices Guide). Either common framework describes the complete set of best practices considered essential for effective software engineering work performance.

THE ASSESSMENT PROCESS: BASE AND GENERIC PRACTICES

The presence or absence of requisite (as defined by the reference mode) base practices allows the assessor to make judgments about the state of a given process. Each process is describable by its base practices, which in a practical sense are just the activities that should correctly be performed in order to achieve a given purpose.

A base practice is a software engineering or management action that addresses the purpose of a particular process. It is assumed that consistently performing the base practices of a process will aid in consistently achieving its purpose. The base practices (from whichever reference model is selected) are tailored (e.g., individually determined) for each assessment target. This is normally highly context sensitive and depends primarily on the organization's interest in a critical software engineering process, the need to address some particular area of management concern or the need to determine whether the unit can meet a defined level of risk/capability for a prospective enterprise. Within either one of the common reference models (e.g., 12207 or 15504), the processes that are defined and assessed fit into one of five categories:

- Customer-Supplier;
- Engineering;
- Management;
- Support;
- Organization.

The **Customer-Supplier** category consists of processes that directly impact the customer/supplier relationship, and the **Engineering** process category consists of software engineering practices that specify, implement or maintain a software item. The **Management** category consists of processes that establish the project and manage its resources. The **Support** process category consists of processes that may be employed by other processes that include such common elements of best practice as configuration management, SQA, and peer reviews. The **Organization** process category consists of processes that establish the business purposes and goals, such as process definition and improvement, training and reuse.

The model also specifies a set of generic management practices that are embodied in all of these base practices and determine their capability (e.g., the management capability of each of the base practices can be assessed in terms of the presence or absence of these features). These may be used to characterize a given level of capability maturity for that particular process. There are 11 of these "common features," which embrace 26 generic management practices, and are grouped into five levels of increasing capability.

The first level of maturity is the "Performed Informally" level (level one). This level is composed of a single common feature and one generic practice: Perform the Base Practice (the base practice is performed).

The second level is called "Planned and Tracked." This level is composed of four common features characterized by 12 generic practices: Planning Performance (allocate resources, assign responsibility, document the process, provide tools, ensure training, plan the process); Disciplined Performance (use plans/standards and do configuration management); Verifying Performance (verify process compliance, audit work products) and Tracking Performance (track with measurement, take corrective action).

The third level of maturity is the "Well Defined" level. This level is composed of two common features embodied in five generic practices: Define a Standard Process (standardize the process and tailor the solution) and Perform the Defined Process (employ a well-defined process, perform peer reviews, use well-defined data).

The fourth level of maturity is the "Quantitatively Controlled" level. This level is composed of two common features characterized by three generic practices: Establish Measurable Quality Goals (establish quality goals) and Objectively Manage Performance (determine and use process capability).

Finally, the fifth level of maturity is the "Continuously Improving Level." This level is composed of two common features and five generic practice characteristics: Improve Organizational Capability (establish process effectiveness goals, continuously improve the standard process) and Improve Process Effectiveness (perform causal analysis, eliminate defect causes, continuously improve the defined process).

THE ASSESSMENT PROCESS: PERFORMING THE ASSESSMENT

The assessment process is intended to characterize both the persistence of the process and its level of capability. Processes are always defined in terms of three things: Purpose, Practices and Capabilities.

Purpose Statements. Each process must be described in terms of a purpose statement. These statements specify the unique practices of the process when instantiated in a particular environment. The purpose statement also includes how the existence of that process will be confirmed.

The Process Dimension. The process dimension assures that the process is performed in accordance with standard best practice. Neither 12207 nor 15504 define how or in what order a process purpose should be achieved. They simply dictate the characteristics that must be present. However, they do stipulate that this

must be done through activities and tasks that they specify for the processes that they embody. Performance of these tasks and characteristics of the work products are the indicators that demonstrate that a process purpose has been met.

The Capability Dimension. In addition the model specifies a set of generic common features (e.g., this set applies to ALL processes). These are grouped by capability level. The assessment output consists of a set of process capability level ratings for each process instance. In the aggregate, these ratings are the basis for the determination of overall organizational capability. Each process instance can potentially be rated at one of five capability levels. That rating is dependent on how many of the common feature characteristics of a given level of maturity are embodied in that process instance. The capability of a unique instance can be estimated by determining the degree to which it performs the generic management practices that make up the common features.

There are essentially two aspects involved in the assessment. First, the process dimension, which is characterized by process purpose statements and the base practices (that address that purpose), is assessed to determine whether the specified (by the reference model) best practices are being performed. Then the capability dimension, which is characterized by the common feature attributes that describe a given level of capability, is assessed to determine the level of capability maturity of a process instance.

Depending on the overall strategic goals of the organization the assessment can actually be carried out in two different ways: self-assessment: which can be both team based and/or continuous, and independent audit (e.g., third party). The typical assessment comprises seven stages:

1. Reviewing the Assessment Input;
2. Selecting the Process Instances;
3. Preparing the Assessment;
4. Collecting and Verifying Information on Practices;
5. Determining the Ratings for Process Instances;
6. Validating the Ratings;
7. Presenting the Assessment Output.

Stage One: Reviewing Assessment Inputs

At a minimum, purpose, scope, constraints, responsibilities of the assessors, required practices not in the common reference model and any other information to be collected must be defined prior to the assessment.

Stage Two: Selecting the Process Instances

The first substantive step in the assessment process requires the organization to select the process instances and map them to the reference model (e.g., the

processes to be assessed are mapped to the corresponding processes defined in the reference model). Instances targeted for assessment must fit within the assessment scope for that process, and there must be at least one documented instance for all processes identified in the assessment scope.

Stage Three: Preparing the Assessment

This step involves selecting and orienting the assessment team. The size of this team will depend on scope, size of the organizational unit, skills and experience of the available resources, as well as cost/benefit. It is necessary to ensure that the team has understanding of assessment inputs, purpose, constraints, output and processes and has identified all of the risk factors. For example, the assessment team should identify any significant risk factors that may cause the assessment to fail including:

- Changes in commitment of the sponsor;
- Unplanned changes in the assessment team;
- Organization change;
- Changeover to a new standard process;
- Changes in assessment purpose or scope;
- Organizational resistance;
- Lack of financial or other resources;
- Lack of confidentiality.

Once it is formed and briefed the team selects the assessment technique that suits the purpose, scope and style of the organization, as well as an assessment instrument. Elements that need to be considered include the type of instrumentation required, the level of support for security and confidentiality, the level and detail of reporting required and the level of organizational commitment to the rating and analysis function.

Following this, the team develops an assessment plan that details the assessment inputs, the roles and responsibilities of the assessors, estimates of schedule, cost and resources, the control mechanisms and checkpoints, the interface between the team and the unit, the expected outputs, the identified risk factors and any logistical requirements. Also the organizational unit being assessed needs to appoint a coordinator to represent it in the assessment. That coordinator is responsible for determining and supporting all of the various logistical issues raised for the organizational unit, as well as interfacing with the assessment team and establishing the environment and conditions necessary to facilitate the assessment activities themselves.

Stage Four: Collecting and Verifying Information on Practices

The assessment team has to capture information on every base and generic practice defined as required for each process instance to be assessed. Therefore, the

organizational unit must designate participants to be interviewed who can best represent the process instances. In addition the unit must insure that the appropriate internal expertise is available as required by the assessment team.

The assessment team may also require access to support documentation and records. This may be particularly important for a third-party assessment. Participants or the organizational unit coordinator have to ensure that these are available. Usually the access requirements are specified in the assessment plan.

The actual process of information collection is always iterative (e.g., it involves a series of information collection and analysis stages). General categories of information collected focus on confirming the existence/adequacy of the base practice and the adequacy of the common features. Typically an assessment instrument is used to collect information about all of this. Support documentation and records are then used to verify the information collected. The amount required normally depends on the assessment team's knowledge of the organizational unit.

Stage Five: Determining the Ratings

Each base practice within each assessed process must be given a validated rating for both its existence and the adequacy of its performance. In addition base practice ratings of a process instance must be explicitly traceable to the reference model. This is usually accomplished through a unique (coded) label that links the assessed and documented base practice to the reference model process category for each uniquely defined process instance. Base practice existence is rated based on the nominal scale of nonexistent (the base practice is either not implemented or does not produce any recognizable work products) or existent (the base practice produces identifiable work products). Base practice adequacy is rated based on the following scale:

- **Not adequate**;
- **Partially adequate —** the base practice contributes little to the process purpose;
- **Largely adequate —** the base practice largely satisfies the process purpose;
- **Fully adequate —** the base practice satisfies the process purpose.

The level of capability of the performance of the management of the base practice is rated using the common feature rating scale in the manner described earlier. It is possible that an individual base practice though fully implemented is not satisfying its process purpose. Thus the process capability rating provides insight into the effectiveness of the process. A present process capability level rating must be determined for each base practice within every process instance. For broader management purposes an aggregated capability rating can be derived from the assessment of the capability of each process instance. This provides a capability rating for the organizational unit as a whole.

Stage Six: Validating the Ratings

The base and generic practice adequacy ratings are validated by comparing results to those from previous assessments, looking for consistencies between connected or related processes, or looking at proportional ratings across capability levels.

Stage Seven: Presenting the Assessment Output

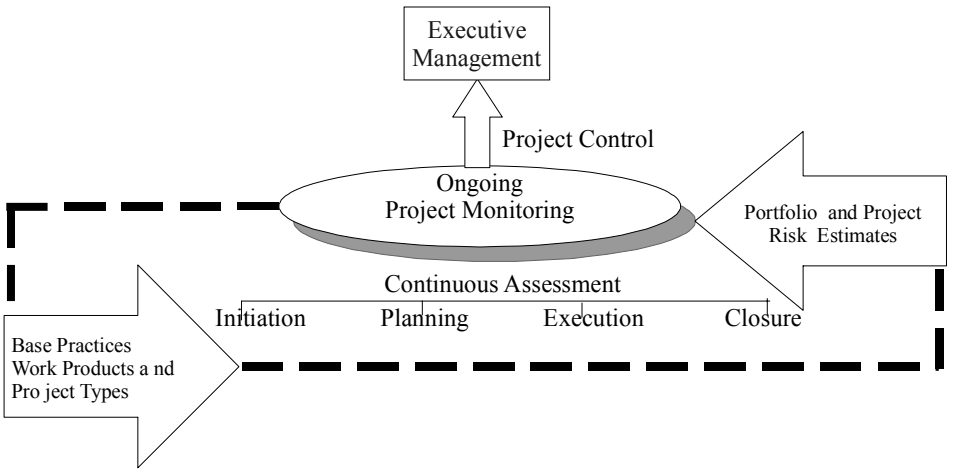
The assessor must ensure that all information specified in the assessment output is archived and that ratings for the assessed instances are recorded in a process profile. The presentation of this may simply be in the form of a single presentation. It may also be in the form of a detailed report with action plans. The presentation may be in absolute terms (e.g., numbers) or relative terms (since last assessment).

A THREE-PHASE IMPLEMENTATION MODEL

This model is implemented in three organic phases. In the first phase, the reference model is popularized within the organization. This involves selecting an appropriate standard and defining the degree to which it should be applied (e.g., how thoroughly the base practices will be assessed). It does not involve forcing anybody to change behavior. It is simply a planning function. Essentially the organization undertakes a process of deciding which reference model best fits its situation and philosophy.

The next phase begins once a foundational set of best practices has been defined and agreed on. In this "structural" phase, a process is employed to install quantifiable, repeatable processes and the measures to assess them. Project

Figure 1. Phase One: The Foundation Model



launches and workshops are the means to accomplish this. The reader should note here that this is an educational rather than a change activity. The procedures and measures are introduced to the work force as a whole through training, in-house consulting (by designated champions) and mentoring, not by requiring adoption of a procedure as a condition of compliance.

The assessment outputs provide management with performance ratings that let them judge how adequately base practices are instituted. And it is this "adequacy" rating that allows them to assess the prospective and ongoing risks associated with the execution of the project. Essentially, nobody is forced to change but the "risks" of not performing base practices can be evaluated. This gives executives the freedom to decide future courses of action in the case of "high risk" projects (e.g., those where the base practices are not being followed). Furthermore, since these assessments are ongoing managers can respond appropriately to the known situation as it evolves.

In the third phase the organization builds an enterprise project management architecture. Projects are assessed within its framework. As a consequence every project is monitored and continuously assessed throughout its execution cycle.

THE RISK ANALYSIS REPORT

Any project can be characterized at any point in its functioning using the integration model. As a result, it is possible to nail down the level of risk associated with a project at any point in its lifecycle. From that understanding the risk can be identified, mitigated and controlled. This allows the organization to monitor the ongoing risks associated with the entire portfolio, thus providing all of the information

Figure 2. Phase Two: The Structural Model

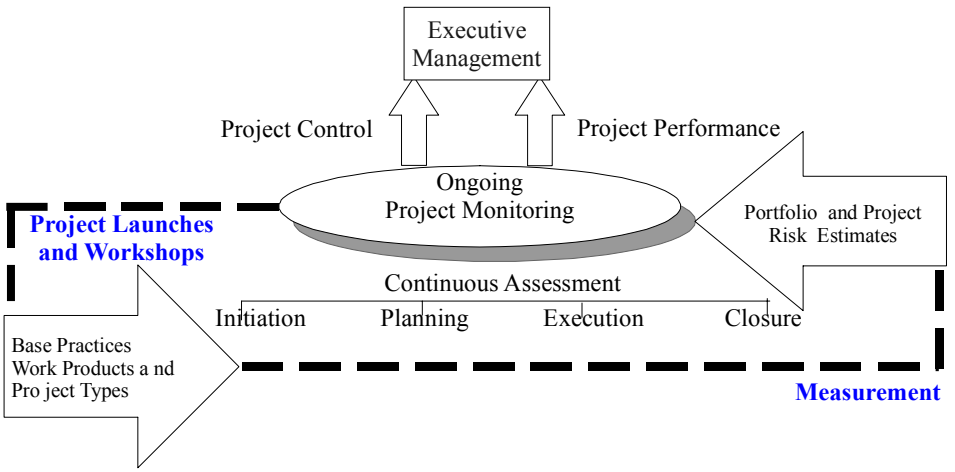
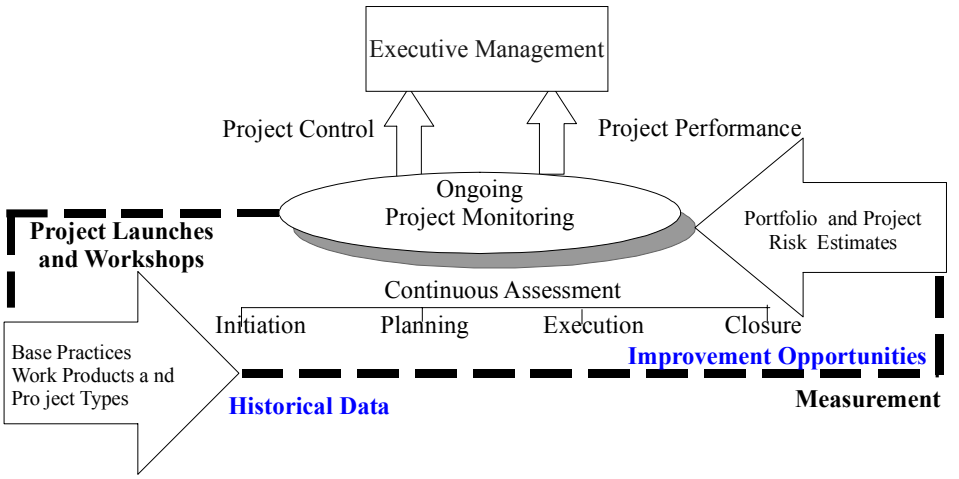


Figure 3. Phase Three: The Integration Model



necessary to allow decision makers to guide the organization as effectively as possible. The total project risk is determined using the following inputs:

1. The expected, or target, rating for the project;
2. The actual assessed rating for the project (e.g., Fully, Largely, Partly, Not);
3. The gap between the target and actual ratings (as a percent);
4. The probability for problems occurring because of that gap (as a percentage);
5. The risk of potential impacts from problems occurring.

This is captured and presented in a Project Risk Analysis Report. What is reported is the level of performance of the process instance with respect to current and targeted levels of capability. In accordance with the rating scale discussed in the prior section, the potential levels of performance of these base practices is Fully (F), Largely (L), Partly (P) and Not (N). The other dimension is capability. Because the assessment model is based on a six-level classification structure (e.g., level 0 through 5), capability is expressed on a six-point ordinal scale. The scale represents increasing capability from performance that is not capable of achieving its goals (level 0: Incomplete) to performance that is capable of meeting relevant process and improvement goals explicitly derived from the organization's business plan (level 5: Optimizing). This can also be expressed on a percentage scale and, therefore, provide a more detailed insight into the specific aspects of a project. This scale is shown in Table 1.

The following examples represent possible assessments of three different projects. The first column presents the degree of performance of the base practices (from Not performed to Fully performed) and the third column reports the level of assessed capability (from nonexistent to optimizing). Columns two and four provide

Table 1. Percentage Scale Showing Process Attribute Ratings

PROCESS ATTRIBUTE RATINGS		
Fully Achieved	86-100%	F
Largely Achieved	51-85%	L
Partially Achieved	16-50%	P
Not Achieved	0-15%	N

a visual readout of the same information. For instance, the first example shows the rating for a medium-risk project. The first cell states that the base practices are assessed as Largely (L) fulfilled whereas the target is Fully (F), which produces a minor gap and a slight probability of failure. Thus the risk associated with that would be medium. The other cells can be read in the same fashion.

In the second project the base practices are assessed as Fully (F) carried out. Since the target is F there is no gap and the probability of failure is accordingly very slight. Thus the risk associated with that would be relatively nonexistent.

A more likely set of outcomes is shown in the third project. In this example the base practices are assessed as Partly (P) carried out (in the sense that they can be confirmed as understood and executed but not in a systematic fashion). Since the target is Fully (F) there is a Major gap and the probability of failure is accordingly Substantial. Thus the risk associated with that would be very High. With respect to their assessed capability, the fact that they are understood is not represented in the documentation and, therefore, the assessed capability level is 1 (not capable), and there would be a significant probability of failure attached to this project and the likelihood of failure would be high.

The entire portfolio of projects can be represented in such a fashion and easily understood by busy executives. Notwithstanding the potential "heads-up" that this would provide for them, the detailed assessment information embodied in this

Figure 4. Project One: A Medium Risk Project

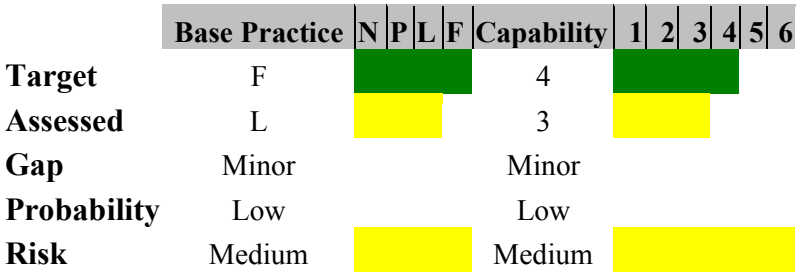
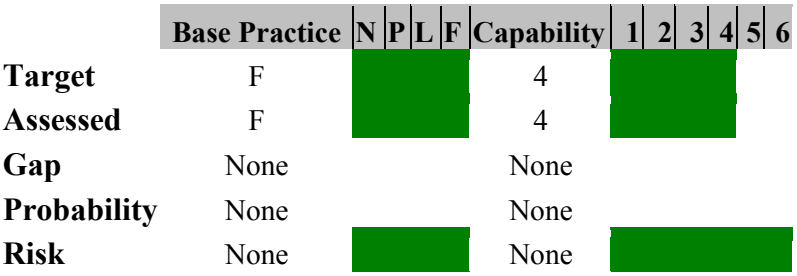


Figure 5. Project Two: A Low Risk Project



provides the elaboration sufficient to allow for their staff to make informed corrections.

CONCLUSIONS

Organizations cannot be effectively managed without information. Yet the very nature of IT work makes it hard for decision makers to get effective oversight. This is the case because of the unique nature of software work. Organizations, be they software shops or the Prussian general staff function on three mutually supporting hierarchical levels. The diagram in Figure 7 outlines these.

In theory a few policy makers are the captains, who decide where the ship is going. This decision is delegated to a larger set of individual managers who are kings in their functional areas but who do not set overall policy. They implement that decision by supervising the hoisting of the sails. The role of the operations people is to do the heaving. They should not be deciding anything more long term than whatever it takes to get the canvas up the mast. Of course, this all presupposes that the captain can see where the ship's going and the managers can actually oversee

Figure 6. Project Three: A High Risk Project

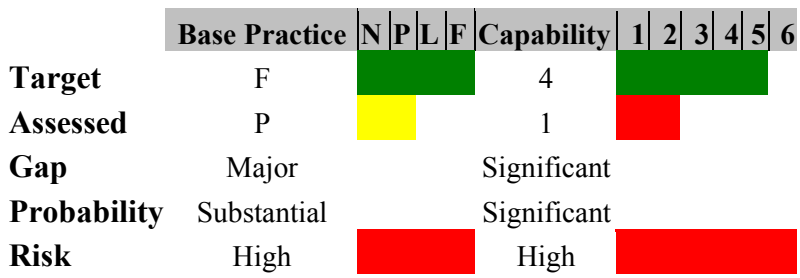
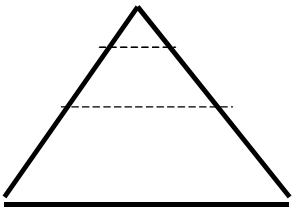


Figure 7. Diagram of the Functions of Organizations on Three Hierarchical Levels



whatever task is delegated to them. In fact, since the development and maintenance of software involves highly complex creative and abstract (meaning conceptual) behavior, the only people who actually know what's going on with the product are the people who are making it, which creates an organizational disconnect something like what is shown in Figure 8.

That is why software infrastructure is notoriously flat, meaning (in essence) completely unmanaged. And it is also summarizes the whole purpose of most popular process improvement models, which is to enforce a monitoring and control function that looks like Figure 9.

In essence the point of these ideal frameworks is to let the appropriate decision maker see what's going on in the process so that they can actually enforce control over their areas of responsibility. Ideally, the point of this is to ensure that every decision maker in the organization has sufficient focused information in front of them when they are required to make decisions that affect their functional areas. This will allow them to deploy resources efficiently, maximize coordination and react effectively to any and all contingencies as they arise in the day-to-day process of development and maintenance. Moreover it is the specific lack of sufficient management vision (as represented by that lack of information) that has created the current cost and performance problems in the IT business.

Of course a lot of software projects get run very well, but the evidence is also manifestly clear that this happens where the project manager has sufficient experience with similar projects to allow them to steer around the rocks. The

Figure 8. Organizational Disconnect

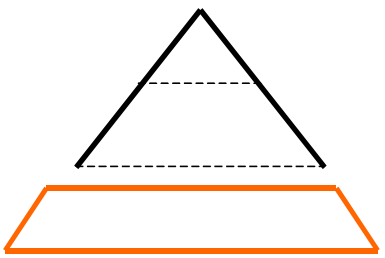
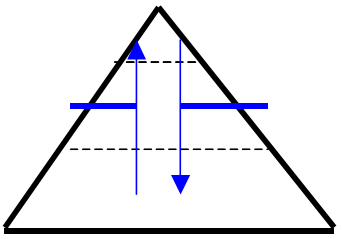


Figure 9. Illustration of a Monitoring and Control Function



evidence is also unmistakable that in the case of new development (e.g., something outside of the experience of the manager) or inexperienced project managers the results are consistently abysmal.

So the question remains, is there a systematic way to manage software projects without resorting to climbing up a very steep and costly learning curve, or even worse bringing in a guru who will cost the organization an arm an a leg just to assure that the project will be delivered close to budget and schedule? The answer is that the only way to insure that projects will be consistently managed effectively is for project and executive managers to have systematic access to pertinent, reliable, accurate and quantifiable information. This means that it has to be possible to measure and report on the performance of each project in the company's portfolio throughout its lifecycle. Thus as each IT project moves through concept to development to eventual use, the information system will be able to provide up-to-date tracking telemetry about project functioning as well as the level of technical and financial risk to the people responsible for guiding it.

Creating such a system is a relatively simple process. It entails the establishment of a measurement-based management system targeted specifically at generating relevant and focused decision support information. In turn it implies the requirement for a scientifically defined enterprise-oriented project management architecture, where methodology is the driving force (rather than the individual experience of employees). Only a standard methodology can reliably guarantee that information will be systematically derived and reported, which in turn implies the absolute requirement for quantitative data. This is obtained by measuring project performance on an ongoing basis. And it is the only way to make certain that the right resources are available to deliver the projected benefit as well as ensure that historical data, known risks and organizational best practices are used as inputs to project planning. Finally, and most importantly, a systematic methodology is the only way to reliably ensure the necessary monitoring and control over all projects. That is the purpose of the quantitative management model.

As can be seen, any software project can be managed using this approach. The organization's (or its suppliers') capacity to successfully deliver the project is estimated based on the performance of best practices. The risk associated with committing to a project, or selecting a certain supplier, can then be translated into an

objective index, which the organization can use to account for and control its ongoing functioning. The Risk Assumed report enumerates the **management** capability of the project, which is assigned based on the management practices that are embodied in the overall conduct of the project. One important point needs to be noted, the risks identified are at the overall organizational management level, and they should be considered separate from software engineering risks that are identified and mitigated during the project lifecycle. Thus the information derived from this model can objectively identify which projects are at risk **before** the project starts. This enables an organization to focus on and manage the risks identified as most likely to cause the project to fail during its lifecycle, which is immeasurably valuable.

REFERENCES

Boehm, B. (1987, September). Improving software productivity. *Computer, 20*(9), 43-57.

Card, D., & Comer, E. (1994, September). Why do so many reuse programs fail? *IEEE Software,* 11(5), 114-115.

Construx Software Builders. (2001). Retrieved August 2001 from http://www.construx.com.

Curtis, W. (1995). Building a cost-benefit case for SPI. *Proceedings from the 7th SEPG Conference*. Boston, MA: DACS.

Dion, R. (1993, July). Process improvement and the corporate balance sheet. *IEEE Software*.

Dorofee, A.J., Walker, J.A., & Williams, R.C. (1997, April). Risk management in practice. *Crosstalk, 10*(4).

Fenton, N. (1993). How effective are software engineering methods. *Journal of Systems and Software, 22.*

Humphrey, W. (1994). *Managing the Software Process.* Reading, MA: Addison-Wesley.

Humphrey, W. (1995). *A Discipline for Software Engineering.* Reading, MA: Addison-Wesley.

International Organization For Standards. (1995). *ISO/IEC 12207.* Geneva, Switzerland.

International Organization for Standards. (1998). *TR- 15504.* Geneva, Switzerland.

ISACA (2000). *Framework*, *CobiT* (3rd Ed.). Rolling Meadows, IL: The Information Systems Audit and Control Association and Foundation.

Jones, C. (1997). Assessment and control of software risks. Englewood Cliffs, NJ: Prentice-Hall.

KPMG Technology and Services Group. (2001). Retrieved September 2001 from http://www.kpmg.ca.

Laker Consulting. (2000). Retrieved August 2000 from http://www.laker.com.au.

Lee, E. (1997). Software inspections: How to diagnose problems and improve the

odds of organizational acceptance. *Crosstalk, 10*(8).
McGarry, F., & Jeletic, K. (1993). *Process improvement as an investment: Measuring its worth* (Report No. SEL-93-003). NASA Goddard Space Flight Center.
McGibbon & Thomas (199). *A business case for software process improvement revised.* DoD Data Analysis Center for Software (DACS).
Moore, J. W. (1998). *Software Engineering Standards: A User's Road Map*. Los Alamitos, CA: IEEE Computer Society.
Office of Management and Budget Evaluating Information Technology Investments. Retrieved August 2001 from http://www.itmweb.com.
Ould & Martyn, A. (1996, December). CMM and ISO 9001. *Software Process Improvement and Practice, 2*(4) 281-289.
Paulk & Mark, C. (1999). Toward quantitative process management with exploratory data analysis. *Proceedings from the International Conference on Software Quality,*
Paulk, M.C., Goldenson, D., David, & M., White. (2000). *Survey of high maturity organizations.* (Special Rep. CMU/SEI-2000-SR-002).
Rozum, J. (1993). *Concepts on measuring the benefits of software process improvement*. (Rep. No. CMU/SEI-93-TR-09, ESC-93-TR-186).
Shoemaker, D. (2001). Requirements Based Estimation. *Proceedings from the Decision Sciences International Conference.* San Francisco, CA.
Software Engineering Institute. Retrieved 1998 from http://www.sei.cmu.edu.
Strassman, P.A. (1990). *The business value of computers.* New Canaan, CT: The Information Economics Press.
Violino, R. (1997, June 30). Measuring value: Return on investment. *Information Week,* (637), 36-44.
Yamamura, G., & Wigle, G.B. (1997, August). SEI CMM Level Five: For the right reasons. *Crosstalk, 10*(8), 3-6.

ENDNOTES

[1] For the purposes of this discussion, a process instance is defined as a singular instantiation of a process that is uniquely identifiable and about which information can be gathered in a repeatable manner.

Chapter IX

Software Metrics, Information and Entropy

Jana Dospisil
Monash University, Australia

ABSTRACT

This chapter describes the foundation and properties of object-oriented software measures. Many software measures for object-oriented applications have been developed and tested in the development environment. However, the process of defining new measures is still alive. The reason for such development lies in difficulties associated with understanding and maintaining object-oriented applications. It is still difficult to relate the measures to the phenomena we want to improve. Do our measurements indicate problems in reliability, maintenance, or too much complexity of some portions of the application?
In order to reduce the complexity of software, new development methodologies and tools are being introduced. An example of the new approach to development is separation of concern. The tools, such as Aspect/J (Kiezales et al., 1997) or Hyper/J (Ossher & Tarr, 1998), facilitate the development process. There does

not seem to be a sound metrics suite to measure complexity and efficiency of applications developed and coded with Aspect/J or Hyper/J. In this chapter, we attempt to review the current research into object-oriented software metrics and suggest theoretical framework for complexity estimation and ranking of compositional units in object-oriented applications developed with Hyper/J.

INTRODUCTION

Software metrics is a term used to describe a wide range of techniques concerned with measurement of the quality of software products. These techniques range from collecting quantitative aspects of code (e.g., Lines Of Code) to models used for prediction of software quality. The objectives of software engineering are to improve the quality of developed software and provide tools for reducing software complexity. These objectives can lead to reduced cost for software development, facilitate maintenance and allow evolution and extension of the software.

For some time it has been estimated that over 70 percent of software development is spent in testing and maintenance of software (Zuse, 1994). The reports from large commercial projects, which utilize the object-orientated techniques, indicate that the expected cost savings in maintenance have not been delivered. The increased **complexity** and size of software projects have led to the development of many different concepts for breaking a system into less complex and manageable modules (Dijkstra, 1976).

The principle of **separation of concerns** also made its way into object-oriented design. The source of the problem in software development is that some kinds of behavior or functionality cross cut or are orthogonal *to* classes in many object-oriented components, and they are not easily modularized to a separate class. Examples of such behavior include the following: synchronization and concurrency, performance optimization, exception handling and event monitoring, coordination and interaction protocols, and object views.

Recent research at Xerox PARC in aspect-oriented programming (AOP) and a multidimensional separation project at IBM seek to alleviate this. This new programming model language constructs and compiles interleave components and aspects or concern definitions (programs) appropriately to formulate a unified and executable program. The objective is to reduce the complexity and promote easy maintenance.

The chapter is organized as follows: The chapter begins with an introduction of object-oriented structures and provides an overview of established metrics. Then, it introduces the notion of complexity and **entropy-based metrics** for complexity. It is followed by a section that deals with the concept of separation of concern, and provides the theoretical underpinning of Hyper/J. Furthermore, the proposed metric suite for Hyper/J is presented.

OBJECT-ORIENTED METRICS

There is a growing concern that software metrics should have a solid base. Measurement theory has been suggested for translating mathematical properties of measures to empirical (intuitive) properties. Rigorous approach to object-oriented metrics has been suggested by a number of researches. In literature, more than 100 software measures for object-oriented applications can be found.

Object-Oriented Structures

Object-oriented development methods have their own terminology to reflect the structural concepts. The main concept is classification meaning that the elements containing similar attributes and operations are encapsulated in a template called class. Each class can have subclasses that inherit the attributes and methods of the superclass. Class instance is an object which contains concrete data or information. Adhering to the principle of encapsulation, message passing is the only means to access an object's state. Object A is coupled with another object B if, and only if, A sends a message to B. Cohesion describes the binding of the elements (attributes and methods) within the same class while ignoring instance variables and inherited methods.

Brief Overview of Established Metrics

Since 1995 the trend towards incorporating measurement theory into all software metrics has led to identification of scales for measures, providing some perspective on dimensions. When we consider measurement units, we need to understand the different measurement scale types implied by the particular unit. The most common scale types based on measurement theory (Roberts, 1979), and the applied approach explanation in Henderson-Sellers (1996) and Kitchenham (1996) are:

- **Ordinal —** an order set of categories (often used for adjustment factors in cost models based on a fixed set of scale points);
- **Interval —** numerical values where the difference between each consecutive pair of numbers is an equivalent amount, but there is no 'real' zero value;
- **Ratio —** similar to interval scale, but it includes an absolute zero;
- **Nominal —** a set of categories into which an item is classified.

The unit's scale type determines the admissible transformations and statistics we can use to analyze measures. Fetchke (1995) and Zuse (1991) analyzed the properties of object-oriented software metrics on the basis of measurement theory. The underlying notion of measurement theory is based on intuitive or empirical existence of relationships among objects within our Universe of Discourse. These relationships then can be formally described in a mathematically derived formal relational system. Zuse (1991) also investigated how and under what conditions the

software measures may be viewed as ordinal, ratio, nominal and interval. He admits that these scale types present very little meaning with regard to maintainability and error proneness of the application.

Zuse's object-oriented model allows four levels of abstraction: class, method, inheritance tree and delegation (uses hierarchies). They defined a set of empirical relations (1994) considering concatenation operations on class level and method level. In addition to Chidamber and Kemerer's (1994) concatenation operations, Zuse's quantitative criteria for software measures are based on the theory of extensive structure and a set of empirical conditions (axioms). The approach of extensive structure can also be applied to cost estimation models, design and maintainability measure. Zuse also proved that with the dynamic nature of object-oriented software we need to consider quantitative and qualitative probabilities and belief structures (e.g., Dempster-Shafer Function of Belief, the Kolmogoroff axioms and others).

The goal of Zuse's new axiom system is briefly described in Figure 1. The relations defined by Fetchke and Zuse include the following operations: **Class unification** C=CUNI(A,B) results in a new class C, which combines the properties of the classes A and B. The implication of the concatenation on inheritance relationship whether the unification is applied to classes with the same root class (father) or the classes with separate father are subject of unification. **Class intersection** CINT(A,B) is not a simple combination of methods to build a new class. In addition, it allows the definition of an empty class and associated metrics. The same definitions were applied to methods. The following is the equation for class unification and empty class:

$$CINT(A, B) = \varnothing$$
$$\mu(\varnothing) = 0 \qquad (1)$$

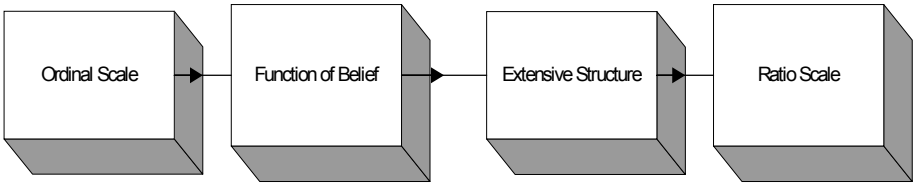
Krantz, Luce, Suppes and Tversky (1971) suggest that significant effort in designing and analyzing measurement goes into finding a good axiom system. They argue that at least one axiom is required to construct a representation. One of the odd axioms is the Archimedean axiom: for any positive number x, no matter how small and any number y, no matter how large, there exists an integer n such that:

$$nx \geq y \qquad (2)$$

With regards to scale type, Zuse argues that in an object-oriented environment the behavior of measures relevant to concatenation operations do not assume extensive structure, and consequently they cannot be used as a (additive) ratio scale. Some other measures assume indempotency, which is the concatenation rule defined as:

$$a \in A \rightarrow a \circ a = a \qquad (3)$$

Figure 1. From Ordinal Scale to Extensive Structure



The implication is that they do not meet the Archimedean axiom.

The contribution of Zuse and Fetchke's work (also in Fenton & Pfleger [1997]) is an introduction of a specific perspective of measures. They emphasize preciseness for the definition of scales as well as a definition of the attribute that is measured.

An axiomatic approach was proposed by Weyuker (1988). This framework is based on a set of nine axioms listed in Table 1 against which software could be formally evaluated. The notations used are as follows: Classes P, Q, R, P+Q denote a combination of classes P and Q, and m denotes the chosen metric. We denote m(P) as the value of this metric for the class P. If two classes P and Q provide the same functionality, then P º Q. The description and criticism can be found in Henderson-Sellers (1996).

In this metric proposal we observe the formalization of structural inheritance complexity metrics. The property 9 means that splitting one class into two classes can reduce the complexity. From the practical point of view and as argued by Chidamber and Kemerer (1994), the complexity of interaction may even increase when classes are divided.

Table 1. Weyuker's Axioms

Axiom	Name	Description
1	Noncoarseness	$(\exists P)(\exists Q)(\mu(P) \neq \mu(QQ))$
2	Granularity	Let c be non-negative number. Then there is finite number of class complexity = c
3	Nonuniquesniss	There is distinct number of classes P and Q such that μ(P) = μ(Q)
4	Design detail matter	$(\exists P)(\exists Q)(P \equiv Q and \mu(P) \neq \mu(Q))$
5	Monotonicity	$(\forall P)(\forall Q)(\mu(P) \leq \mu(P+Q) and \mu(Q) \leq \mu(P+Q))$
6	Non-equivalence of of of interaction	$a)(\exists P)(\exists Q)(\exists R)\mu(P) = \mu(Q) and \mu(P+R) \neq \mu(Q+R)$ $b)(\exists P)(\exists Q)(\exists R)\mu(P) = \mu(Q) and \mu(R+P) \neq \mu(R+Q)$
7	Interaction among statements	Not considered among objects
8	No change on renaming	If P is renaming of Q then $\mu(P) = \mu(Q)$
9	Interaction can increase complexity	$(\exists P)(\exists Q)(\mu(P) + \mu(Q) < \mu(P+Q))$

Fenton and Pfleger (1997) use the term software metrics to describe the following artifacts:

- A number which is derived, usually empirically, from a process or code (for example, Lines of Code (LOC) or number of function points);
- A scale of measurement (an example is used in Fenton's book is nominal scale or classification);
- An identifiable attribute that is used to provide specific functionality (an example is "portability" or class coupling metric);
- Theoretical or data-driven model describing a dependent variable as a function of independent variables (an example can be the functional relationship between maintenance effort and program size).

These descriptions typically lead to a wide spread confusion between models and their ability to predict desired software characteristics thus their suitability to be used for estimation purposes.

The metrics of Chidamber and Kemerer (1994) summarized in Table 2 have a foundation in measurement theory. The authors do not base their investigation on the extensive structure [see Roberts (1979) for more details]. The criticism by Churcher and Shepperd (1994) points to the ambiguity of some metrics, particularly WMC. Hitz and Montazeri (1996) and Fetchke (1995) show that CBO does not use a sound empirical relation system because it is not based on extensive structures. Furthermore, LCOM allows representation of equivalent cases differently, thus introducing an additional error.

Coupling measures form the important group of measures in assessment of dynamic aspects of design quality. Coupling among objects is loosely defined as the measure of the strength of the connection from one object to another. The stronger the coupling is the more inter-related the objects are. Consequently, understanding and maintenance become more complex.

Table 2. Chidamber and Kemerer Metrics (1994)

Weighted Methods per Class (**WMC**),	$WMC = \sum_{i=1}^{n} c_i$ where c_i is the static complexity of each of the n methods.
Depth of Inheritance Tree (**DIT**),	With multiple inheritance the max DIT is the length from the node to the root
Number of Children (**NOC**),	Number of immediate subclasses
Coupling Between Object Classes (**CBO**),	Number of other classes to which a particular class is coupled. CBO maps the concept of coupling for a class into a measure
The Response for a Class (**RFC**),	The size of response set for a particular class
The Lack of Cohesion Metric (**LCOM**).	$LCOM = \lvert P\rvert - \lvert Q\rvert if \lvert P\rvert > \lvert Q\rvert = 0 otherwise$

The approaches of different authors mostly differ in definition of the measured attribute — coupling among classes. Table 3 provides the summary of differences in definitions [modified from Briand, Daly and Wurst (1999)]. However, some of the attributes involved in coupling may be known only after the implementation is completed. We believe that the CASE tools such as Rational Rose™ provide the environment for designers to identify coupling of classes during the design phase and possibly make changes to avoid unnecessary complexity of the implementation.

Two aspects impact coupling between classes: the frequency of messaging between classes and the type of coupling. The discussion in Eder, Kappel and Schrefl (1994) distinguishes among three types: interaction coupling, component coupling and inheritance coupling. The degree of coupling (strength) is based on defining a partial order on the set of coupling types. The low end is described by small and explicit inter-relationships and high end of the scale is assigned to large, complex and implicit inter-relationships. The definition is subjective and requires empirical assignment of values in order to be used as a software quality indicator.

Table 3 provides the summary of attributes and indicates the suite of metrics by the author in which particular attribute is considered.

Many metrics deal predominantly with static characteristics of code. Hitz and Montazeri (1995) clearly distinguish between static and dynamic class method invocation: number of methods invoked by a class compared to frequency of method invocation. A concise survey and discussion of coupling including critique of current metrics can be found in Briand et al. (1999) and Yacoub, Ammar and Robinson (1999).

Metrics suites capable of capturing the dynamic behavior of objects with regard to coupling and complexity have been presented by Yacoub et al. (1999). Dynamic

Table 3. Comparison of Attribute Definition for Coupling

Attribute definition	Eder et al.	Hitz & Montazeri	Briandt et al.
Public attribute visibility	X		
Method references attribute		X	
Method invokes method	X	X	X
Aggregation	X		X
Class type as a parameter in method signature or return type	X		X
Method's local variable is a class type	X		
A method invoked from within method passes class type as a parameter	X		
Inheritance	X		
Method receives pointer to method			X

behavior of an implementation is described by a set of scenarios. The Export and Import Object Coupling metrics are based on percentage of messages exchanged between class instances (objects) to the total number of messages. The Scenario Profiles introduce the estimated probability of the scenario execution. The complexity metrics are aimed predominantly at assessment of stability of active objects as frequent sources of errors.

Cohesion is defined as a degree to which elements in a class belong together. The desirable property of a good design is to produce highly cohesive classes. Comparison of different frameworks and a thorough discussion can be found in Briandt, Daly and Wurst's work (1997). We have to distinguish between class cohesion and method cohesion. Eder et al. (1994) provide a comprehensive framework that requires semantic analysis of classes and methods. This approach represents a problem in the automatic collection of data. The metrics of Chidamber and Kemerer (1994) define LCOM as the number of disjoint sets created by the intersection of the *n* sets. The definition in (4) does not state how inheritance of methods and attributes is treated with regard to method override and overload and the depth of inheritance tree:

$$LCOM = \begin{Bmatrix} |P| - |Q|, if\ |P| > |Q| \\ 0, otherwise \end{Bmatrix} . \quad (4)$$

In Henderson-Sellers (1996) the cohesion measure is based on number of attributes reference by a method:

$$LCOM = \frac{\frac{1}{a}\sum_{j=1}^{a}\mu(A_j) - m}{1-m} , \quad (5)$$

where *a* is number of attributes, and $\mu(A_j)$ is the measure which yields 0 if each method in the class references all attributes and 1 if each method in a class references only single attribute. Obvious criticism is again how should we treat inheritance?

Discussion

One of the criticisms of many proposed metrics is the lack of theoretical basis. Many metrics show dimensional inconsistencies, or their results are derived from correlative or regression analysis (Henderson-Sellers, 1996). As reported in Gurasaran and Gurdev (2001), experienced object-oriented designers found memory management and run-time errors are more problematic and difficult to deal with. With regard to complex Java applications, our experience is that concurrency and multithreaded architectures are often to blame for difficult maintenance and hidden run-time errors.

Object-oriented middleware and Interface Definition Language (IDL) represent further complexity impacting the testing process, reliability and maintenance of distributed software. The object-oriented metrics discussed above do not provide either sufficient theoretical background or experimental framework for such applications.

Class size problems represent a confusing effect with regard to validity of object-oriented metrics. The confounding effect of class size has been reported by Khaled, Saida, Nishith and Shesh (2001), provoking some doubts with regard to the validity of software metrics currently being used as early quality indicators. The relationship of high coupling factor to faults proneness seems to support the hypothesis that large class size may have a negative impact on quality and the occurrence of faults. The author's suggestion is to pay attention to class size in design.

Effective testing relies on selection of testing techniques and a well-defined set of test scenarios. Software metrics as the quantitative indicators of software complexity could be used in the selection process. There are some approaches relevant to risk management suggested by Khaled. These approaches, mostly built on historical data collections, include classification models, rank models and thresholds.

COMPLEXITY MEASURES

When studying the software development process, we are interested in certain measurable quantities, e.g., occurrence of faults and code maintainability, that we want to predict and control. Any large software application can be composed of many elementary entities, each one of them behaving according to its own parameters. They form a dynamic system. In theory we could describe any system by a set of differential equations each describing one of the entities, perhaps some 10^{20} degrees of freedom. We are interested in two related phenomena:

- **Observables** arising from their average behavior, such as correct interactions, error-handling aspects and unexpected behavior. Observables are obtained via well-defined testing suites.
- Impact of **evolutionary changes** caused by introducing additional methods, attributes, subclasses, etc. These changes typically result in increased disorders and consequently in increased complexity.

Sometimes the evolutionary (mostly maintenance-based) changes may introduce some degree of randomness and thus contribute to the unpredictable behavior of an application. We often say that the application is too "complex" and its maintenance too difficult. If we define complexity as the relationship between the observer and the application, then the complexity becomes a subjective property. If the observer is satisfied with a simple model and its accuracy, then the application

is not complex and well-defined test suites uncover possible unreliability. On the contrary, if the observer requires high degree of reliability, the application then represents a complex system with many degrees of freedom.

The attempts to measure the complexity of code are numerous. McCabe (1976) defined cyclomatic complexity measures of software modules based on the topology (connected graph of the module).

$$\text{Cyclomatic complexity } \boldsymbol{CC = E\text{-}N + p,} \tag{6}$$

where $\boldsymbol{E}$ is the number of edges, N is the number of nodes and $\boldsymbol{p}$ is the number of connected components.

In order to automate and consistently count the elements, tools have been developed. The common application of cyclomatic complexity is to compare measures to a set of thresholds. Table 4 shows results of our experiment for method complexity. Typically methods for "set" and "get" are low risk (some computations add complexity and concurrent code or remote procedures) and distributed objects using middleware are high risk.

Halstead's (1977) complexity measures were developed to measure a module's complexity directly from source code. The suite of Halstead's measures is composed of four measures that emphasize computational complexity of a module. Table 5 provides the summary and relevant formulas.

Halstead's suite of measures, introduced in 1977, has been used and experimented with extensively. These measures are simple to calculate. However, in order to automate the calculation process, strong rules for identifying the operators and operands have to be established.

Entropy and Information

From the brief survey of complexity measures presented earlier in the chapter, we can see that object-oriented metrics are considered from different perspectives. Cognitive complexity is measured by coupling and cohesion metrics, for example.

Table 4. Cyclomatic Complexity for Methods

Complexity value for a method	Risk thresholds
1-5	Simple, low risk of defects
5-20	Complex, moderate risk
21-60	Complex, possibly high risk
60 and higher	Consider unstable

Table 5. Halstead's Complexity Measures

Measure type	Formula	Note
Program length	$N = N_1 + N_2$	N_1 = total number of operators N_2 = total number of operands
Vocabulary	$n = n_1 + n_2$	n_1 is number of distinct operators n_2 is number of distinct operands
Volume	$V = N * \log_2 n$	
Difficulty	$D = \left(\frac{n_1}{2}\right) * \left(\frac{N_2}{n_2}\right)$	
Effort	E = D*V	

Dynamic behavior metrics are related to export and import object coupling and execution scenarios (Yacoub et al., 1999). Furthermore, we can view software metrics at multiple levels, such as system level or class level.

Statistical entropy is a probabilistic measure of uncertainty or ignorance. Information is a measure of a reduction in that uncertainty. Entropy and information are fundamental quantitative measures in cybernetics, which extend the more qualitative concepts of variety and constraint to the probabilistic domain. Variety and constraint can be measured by introducing probabilities. Let's assume that we do not know the precise state of *i* of a system. But we know the probability of distribution *P(i)* that the system would have in state *i*. Variety *V* then can be expressed as entropy H[1]:

$$\sum_{i=1}^{N} p(i) = 1$$

$$H(P) = -\sum_{i \in I} P(i) \log P(i) \qquad (7)$$

H (P) reaches its maximum value if all states are equiprobable, that is we cannot assume that one state is more probable than another. In this case entropy H is reduced to variety *V*. The H reaches value 0 (H=0) if, and only if, the probability of certain state is 1 (and all other states are 0). It that case we have complete information about what state the system is in. Let's define constraint as the difference between maximal and actual uncertainty. If we obtain some information about the state of the system through observations, then this will reduce uncertainty about the system's state by reducing the probability of occurrence of these states. The degree to which the uncertainty is reduced is given by (8):

$$I = H_{before} - H_{after} \quad (8)$$

We also note that there are other methods of weighting the state of a system which do not adhere to probability theory's "additivity" condition. These theories involve concepts from fuzzy systems theory and possibility theory.

Entropy-Based Complexity Measures

Entropy-based complexity measures are based on theory of information. The approach taken by Davis and LeBlanc (1988) who quantify the differences between **anded** and **neted** structures using Shannon's (1949) concept of information entropy. It is an unbiased estimate of the probability of occurrence of event *m*. This measurement is based on chunks of FORTRAN and COBOL code (represented by nodes in the DAG) with the same in-degree and the same out-degree to assess syntactic complexity.

In 1976, Belady and Lehman (1976) elaborated on the law of increasing entropy: the entropy of a system (level of its "unstructuredness") increases with time, unless specific work is executed to maintain or reduce it. Entropy can result in severe complications when a project has to be modified and is generally an obstacle to maintenance.

The use of entropy as a measure of information content has been around since 1992 when it was introduced by Harrison (1992). He assessed the performance of his entropic metrics in two commercial applications with the total number of lines of code more than 130,000. He also claims good performance in terms of associating modules with their error span[2]. Harrison's software complexity metric is based on empirical program entropy. A special symbol, reserved word or a function call is considered an operator [It is assumed that they have certain natural probability distribution (Zweben & Halstead, 1979).] The probability p_i of i^{th} most frequently occurring operator is defined as:

$$p_i = f_i \Big/ N_i \ , \quad (9)$$

where f_i is the number of occurrences of the *ith* operator and N_i is the total number of "nonunique" operators in the program.

Then the complexity is defined as:

$$H = -\sum_{i=1}^{Ni} p_i \log_2 p_i \ . \quad (10)$$

The Average Information Content Classification (AICC) measure is defined as:

$$AICC = -\sum_{i=1}^{Ni} \frac{f_i}{N_i} \log_2 \frac{f_i}{N_i} \quad . \tag{11}$$

This metric is intended to order programs according to their complexity and was tested on C code. However, it does not indicate the "distance" between two programs. It provides only the ordinal position thus restricting its usage.

The work of Bansiya, Davis and Etzkorn (1999) introduces similar complexity measure — Class Definition Entropy (CDE) replaces the operators of Harrison with name strings used in a class. The assumption that all name strings represent approximately equal information is related to the possible error insertion by misusing the string. The metric has been validated on four large projects in C++ and results have been used to estimate Class Implementation Time Complexity measure.

Single-valued measure of complexity is appealing to managers as the simple indicator of development complexity. However, as discussed in Fenton and Pfleger's (1997) book single value cannot be used for the assessment of quality of the entire product. The measures are bound to a single product attribute (e.g., comprehensibility or reliability, etc.). Therefore the results cannot be used as prediction models or as guidance for improving the quality of the product.

MULTIDIMENSIONAL SEPARATION OF CONCERNS (MDSOC)

Separation of concerns is often referred to as the ability to identify, encapsulate and manipulate those software entities that are relevant to a particular concept or purpose. Concern is a primary element of decomposition. In an object-oriented context, it may represent a class, exception handling policy, object interaction protocol, synchronization constraints in multithreaded system and optimizations techniques (caching). The resulting code, instead of being well modularized, contains fragments of different concerns (cross-cutting) thus increasing "chaos" and reducing "understandability" and maintainability of the application. At present, the following are the solutions to this problem: Hyper/J [IBM product (Tarr, Ossher, Harrison & Sutton, 1999)] and Aspect/J [Xerox Palo Alto (Kiczales et al., 1997)].

Hyper/J. Tarr et al. (1999) assert that the separation of concerns is at the core of software engineering and aspect-oriented programming, which has sparked the development of many tools and approaches to modularization of software. There is now widespread realization in the software engineering community that a design and implementation that is solely based on class components has its limitations. This is because object-oriented languages, such as C++, Smalltalk and Java, are not capable

of expressing certain aspects of applications in a reusable way. Tarr et al. (1999) state that "Done well, separation of concerns can provide many software engineering benefits, including reduced complexity." To measure the quality of separation either in N-dimensional space or even the orthogonal separation only as seen in Aspect/J, the new set of metrics is required.

Aspect/J. Aspects tend not to be units of system's functional decomposition, but rather the properties that impact the performance of semantics of components (Kiczales & Lopes, 1997). The examples include synchronization of concurrent objects and memory access patterns. The objective is to alleviate some of the complexity and tangled code that results due to cross-cutting/orthogonal behavior. This behavior is inevitable in any real application. Aspect Weaverth provides compositional service based on the concept of join points.

So far, there have been no metrics or measures to clearly indicate that the complexity of the code has been reduced, and improvements in maintainability have been achieved. This paper deals with theoretical underpinning of proposed metrics to measure the complexity of modules in Hyper/J.

Overview of Hyper/J Concepts

Software is a collection of artifacts that describe the composition and functionality of an application. In Hyper/J, a unit is defined as a syntactic construct within the programming language definition. A unit might be a declaration, statement, class, method or interface. Primitive units include methods and instance variables. Primitive units are treated as atomic. Compound units, such as a class or a package, group primitive units together.

The term MDSOC denotes the separation of multiple, arbitrary kinds (dimensions) of concerns simultaneously, having no single dimension of concern predominantly decomposed. A clean separation of concerns allows isolation and encapsulation of all concerns, which then promotes traceability and reduces complexity. **Concerns** are defined as the primary entities for decomposing software into manageable and comprehensible modules (Ossher & Tarr, 1998), such as classes, features, aspects and roles. The prevalent kind of concern is a class (data type). A hyperspace describes the following properties of concerns: identification, encapsulation and mutual relationships. The concern matrix organizes units according to dimensions and concerns. The encapsulation of concerns is accomplished by introducing mechanism of hyperslices (a set of declaratively complete units). A hypermodule comprises a set of hyperslices being integrated and a set of relationships, which determine mutual dependency between hyperslices. The level of mutual dependency is an important parameter.

Formally, hyperspace is a tuple $\{U, M, H\}$, where U is a set of units (methods are primitive units; classes are modules), M is a concern matrix and H is a set of hypemodules. Hypermodule is a tuple (HS, CR), where HS is a set of hyperslices and CR is a set of composition relationships. A hyperslice is a declaratively complete

concern ($hs \in C$). A concern is modeled as a predicate, c, over units U. It indicates whether a unit addresses that concern. The unit set is then defined as:

$$U(c) = \{u \in U \mid c(u)\}. \quad (12)$$

Concerns are said to overlap if their unit sets are not disjoint. A dimension of concern is a set of concerns whose unit sets partition U. It implies that the concerns within a dimension cannot overlap and must cover all the units. This leads to the declarative completeness constraint.

Declarative completeness in Hyper/J is defined as: Software artifacts are subject to a completeness constraint in which each declaration unit in a system must correspond to compatible definition or implementation in some hyperslice. Declarative completeness serves as a mechanism to reduce high coupling in interrelated units (methods). In order to make hyperslice declaratively complete, we have to at least declare units from other hyperslices (thus allowing later binding). For example, the unit $u_1 \in hs_1$ calls unit $u_2 \in U$, then $\rightarrow u_2 \in hs_2$ and it must be implemented in some other hyperslice. We have to use the declaration u_{2decl} to satisfy the completeness constraint.

Hyper/J also defines the Implementation set as:

$$I(hs) = \{u \in \text{hs} \mid \neg decl(u)\}. \quad (13)$$

A Composition Relationship is a tuple (I, r, f, o), where I is a tuple of input units, r is a correspondence relationship characterizing the relationship of units in I and o is an output unit produced using f, which is the composition function defined as:

$$f : (I \times r) \mapsto U. \quad (14)$$

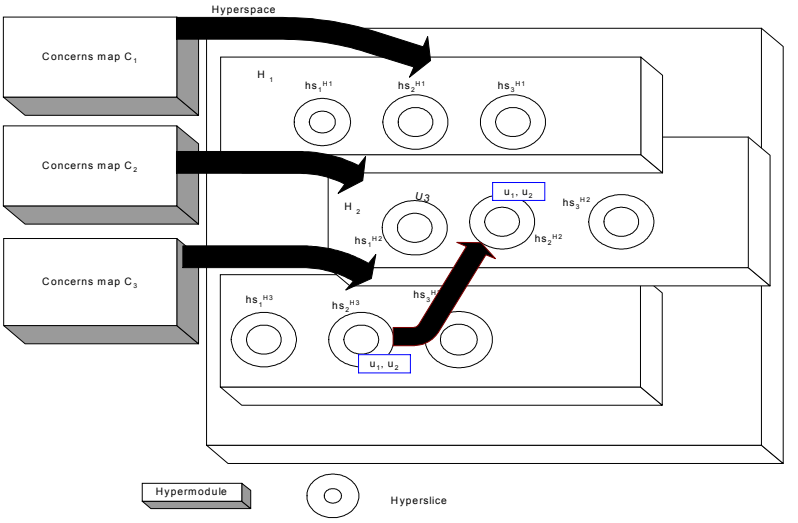
This property means that *the* hyperslice is self-contained providing we define the association called correspondence and provide corresponding units u_{decl}.

Entropy Based Metrics Suite for Separation of Concerns (MDSOC)

Figure 2 shows graphically conceptual representation of overlapping hypermodules, concerns matrix (concern map c_1, concern map c_2 and concern map c_3) and self-contained hyperslices. The hypermodules H_2 and H_3 overlap through hyperslices $hs^{H3}{}_2$ and $hs^{H2}{}_2$. The units (called operations in Hyper/J syntax) u_1 and u_2 are present in both hypermodules.

In each hyperspace, we define the set of composable classes together with mapping these implemented classes. Uncomposable classes are those classes that do not take part in separation. The example in shows units u_1 and u_2, which represent

Figure 2. Hyperspace and Overlapping Hypermodules



methods in hyperslices hs^{H3}_{2} and hs^{H2}_{2}. When the developer creates the concerns matrix, these units will be associated with specific a feature dimension [see Hyper/J tool manual for details (Ossher & Tarr, 1998)]. Reusability and maintenance advantages are achieved by isolating overlapping parts in hyperslices. Our hypothesis is that by establishing a suite of complexity metrics we are able to find "overloaded" hyperslices (with high entropic numbers) and elaborate on dependency between all units and composable only units.

Proposed Metrics Suite

Hyperslice is a set of concerns that are declaratively complete. It contains units that can be related in many different ways. We consider the hyperslice text to be a message in which the special symbol carrying the information is a unit u_i. We define a unit of information u (primitive unit) as a class method (called operation in Hyper/J), class variable or interface definition. In addition, Hyper/J distinguishes other units such as functions, classes and UML diagrams. These units are not included in our metrics.

In a complex system, the invocation of a unit u_i is not deterministic; it depends on the occurrence of outside events — execution scenarios. From the written code we can only obtain the static measures.

Hyperslice Complexity Ranking

We propose that the complexity of a hyperslice is inversely proportional to the average information content of its units. Let the unit space U be a discrete random variable that occurs with the probability mass function $p(u_i)$. The static measure of

uncertainty of U (entropy) is defined as:

$$H(U) = -\sum_{i=1}^{M} p(u_i)\log_2 p(u_i). \quad (15)$$

The entropy depends only on the probabilities. The probability $p(u_i)$ of the most occurring unit is defined as percentage

$$p(u_i) = \frac{f_{ui}}{N_{all}},$$

where f_{ui} is the number of occurrences of the unit u_i and N_{all} is total number of nonunique unit uses in the hyperslice. Nonunique unit uses means that we are also counting the units that are only declared in a hyperslice in order to comply with declarative completeness.

Concerns Matrix Ranking per Hypermodule

Concerns are defined in the concern mapping files (Figure 3). This set of dimensions and classes defines how the classes, interfaces and methods treat concerns. It is important for clarity and maintenance purposes that the concern files should not be overly complex.

We can assess the complexity of the concern matrix defined over the set of composable units U as the entropy:

$$H(C) = -\sum_{i=1}^{M} p(c_i)\log_2 p(c_i); \quad (16)$$

$p(c_i) = \frac{f_{ci}}{N_{Call}}$, where f_{ci} is the frequency of occurrence the unit u_i (operation Operation_name) using the concern c_i (Feature.Concern_name) and N_{Call} is the number of all nonunique units encapsulated by the hypermodule with no regard to

Figure 3. Concern File (Ossher & Tarr, 1998)

```
package demo.ObjectDimension : Feature.Kernel

operation check : Feature.Check
operation display : Feature.Display
operation eval : Feature.Eval

operation check_process : Feature.Check
operation display_process : Feature.Display
operation eval_process : Feature.Eval
operation process : Feature.None
```

which hyperslice they belong. (Note that operation Operation_name and Feature.Concern_name indicate that all methods named Operation_name in the hyperspace, with no regard to class or interface, to what they belong to or their signature address the same concern Feature.Concern_name.)

These two measures (hyperslice and concern ranking metrics) are ordinal measures and enable only ranking of hyperslices and concerns within hypermodules with regard to their entropic complexities of hyperslices and concerns. This measure indicates design differences among hyperslices and concern matrixes.

Weighted Entropy Measure

Sometimes we deal with units where it is necessary to take into account their importance and some qualitative characteristics. We need to associate an elementary unit with both the probability with which it occurs and its qualitative weight. A criterion for a qualitative differentiation of the units of a given scenario is represented by the relevance, the significance, or the utility of the information they carry with respect to an outcome and a qualitative characteristic. The occurrence of a unit removes a double uncertainty: the qualitative one, related to the probability with which it occurs, and qualitative one, related to a given qualitative characteristic. For instance, a unit of small probability can have a great utility with respect to concurrency aspects; likewise, a unit of great probability can have small impact on maintainability (we shall relate this observation to utility). Therefore, the assignment of a weight to every elementary unit is not a simple decision. These weights may have either objective or subjective character. Thus, the weight of one unit may express some qualitative objective characteristics but also may express the subjective utility of the respective unit with respect to the software complexity. The weight ascribed to an elementary unit may also be related to the subjective probability with which respective units are used, and it does not always coincide with the objective probability.

In order to include dynamics in our metrics, we need to distinguish two measures: "number of method invocations" [dynamic measure in Yacoub, et al. (1999)] in contrast to "number of methods invoked" [static, derived from method signatures in Briand, et al. (1999)].

Consider the scenario of guarded waits (Lee, 2000) shown in Figure 4.

Figure 4. Guarded Class Example

```
public class GuardedClass {
       protected boolean cond = false;
public GuardedClass() {
       super();
}
protected void awaitCond() throws java.lang.InterruptedException {
       while(!cond) wait();}
public synchronized void guardedAction() {
       try {
              awaitCond();
       } catch (InterruptedException ie) {
              System.out.println("failed");
       }
       System.out.println("actions");
       }
}
```

This code could fail just because the unrelated object invoked notify() or notifyAll() on the target object by mistake. Errors like this one are difficult to find with static evaluation of the code. GuardedClass is a concern used by objects in many hyperslices and implemented in a single hyperslice. It is difficult to design a suite of tests that would safely discover where and when code starts failing (and often in peculiar ways), due to some unpredicted temporal conditions. In this case, we deal with units where it is necessary to take into account their importance and some qualitative characteristics. We need to associate the elementary unit with both the probability with which its invocation occurs and its qualitative weight (expected participation in failure).

Case 1: Static Evaluation of the Code Derived from Method Signatures

A criterion for a qualitative differentiation of the units of a given code segment represented by the relevance, the significance, or the utility of the information they carry with respect to an outcome and to a qualitative characteristic. The occurrence of a unit removes a double uncertainty: the quantitative one, related to the probability with which it occurs (it is found in the code), and the qualitative one, related to a given qualitative characteristic (anticipated failure risk factor). For instance, a unit of a small probability (coded just a few time in an inheritance tree) can have a great utility with respect to concurrency aspects; likewise, a unit of great probability can have a small impact on maintainability (we shall relate this observation to utility). The weights may have either objective or subjective character. Thus, the weight of one unit may express some qualitative objective characteristics, but also may express the subjective utility of the respective unit with respect to the software complexity.

In order to distinguish the units u_1, u_2, u_3, ... u_n in the unit space *U,* according to their importance with respect to a given qualitative characteristics of implemented or referred to concern, we assign each unit a non-negative weight proportional to its importance and significance:

$$H\left(w(u)_i; q(u)_i\right) = -\sum_{i=1}^{n} w(u)_i\, q(u)_i \log_e q(u)_i \tag{17}$$

$$q_i = q(u)_i = q(u_i) = \frac{f_{ui}}{N_{all}} \ . \tag{18}$$

In the case shown in (19), weights of every elementary occurrence of this unit have an objective character representing the ratio of the objective probability of occurrence of this unit to the amount of information it supplies:

$$w(u)_i = -\frac{q(u_i)}{\log_e q(u_i)} \ . \tag{19}$$

In this case we obtain the following expression for weighted entropy:

$$H(u_i) = \sum_{i=1}^{n} q(u_i)^2 \quad . \tag{20}$$

Weighted entropy of hyperslice calculated from (20)
assesses the static complexity of a component. It does not account for dynamic situations.

Case 2: Dynamic Evaluation

In dynamic evaluation, we consider a probabilistic experiment whose elementary events (method invocations v_k) have the objective probabilities (derived from the code) $q_1 \ldots . q_n$ and subjective probabilities $p_1 \ldots . p_n$ (derived from the test suite). If we ascribe to every elementary event the subjective weight based on the probability of participation in failure:

$$w_k = \frac{q_k}{p_k} \quad ,$$

representing the ratio of the objective probability to subjective probability of the event $v_{k,}$ then we obtain the following formula for dynamic entropy:

$$H_n(u_k) = -\sum_{k=1}^{n} q_k \log_e p_k \quad . \tag{21}$$

Based on test suite coverage of all possible scenarios, we can compare objective entropy $H(u_i)$ with dynamic entropy $H_n(u_k)$ within every hyperslice exposed to concurrency.

Similarly, we can compute the entropy values for hypermodules and produce a table of ordered hypermodules, thus exposing the ones of highest failure risk.

Example Calculation

We have used the modified example from Hyper/J toolkit - DemoSEE. The code for this example has been extended and new capabilities have been added. The UML diagram is in Figure 5. Some concerns and hyperslices have been preserved from the original example (Table 6). We have extended the example using multiple threads running in class Driver_n and several methods for GUI presentation.

Figure 7 and Figure 8 show the graphical representation of entropy and weighted entropy of hyperslices sorted by composable unit count and all unit counts () in a hyperslice. In all cases the weighted entropy shows lower complexity than entropy.

Table 6. Hyperlices and Concerns in DemoSEE

Operations	**Features**	**Hyperslice encapsulation**	**Hypermodule**
process	None		DemoSEE
	Kernel	Kernel (H3)	
check	Check	Check (H2)	
display	Display	Display (H1)	
eval	Eval	Eval (H4)	
check_process	Check		
display_process	Display		
eval_process	Eval		
Additional concerns for Guarded access			
access	Access	Access (H5)	
Guarded_access	Access		

The entropic values follow the trend in number of composable and all unit counts in a hyperslice, respectively.

Figure 9 shows unit distribution and trends in probability of occurrence of units $p(u_i) = \frac{f_{ui}}{N_{all}}$ in Features Kernel, Check and Display sorted by number of composable units in the given Feature. Weighted entropy for Feature Display shows increase due to possible overloading by implementing concurrent access for multiple threads (see for details).

Figure 5. UML Diagram of DemoSEE Example

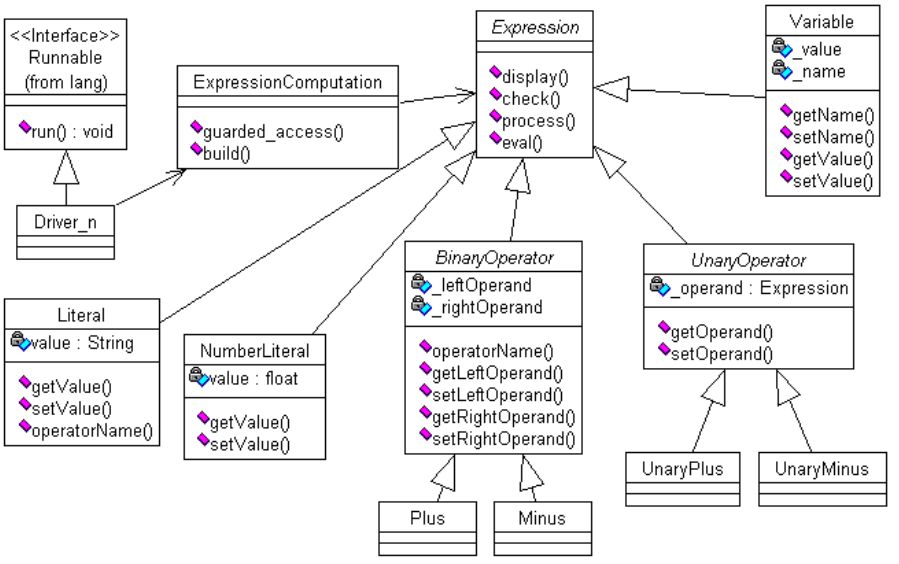


Figure 6. Hyper/J Project Specification

```
Hyperspace Demo
   composable class demo.ObjectDimension.*;
   composable class demo.ObjectDisplay.Driver_n;
   composable class demo.ExpressionComputation

hypermodule ExpressionSEE
   hyperslices:
      Feature.Kernel,
      Feature.Check,
      Feature.Display,
      Feature.Eval,
     Feature Access
relationships:
      mergeByName;
      equate operation Feature.Kernel.process,
                       Feature.Check.check_process,
                       Feature.Display.display_process,
                       Feature.Eval.eval_process;
                      Feature.Access.guarded_access;
end hypermodule;
//concern.cm ->concerns file
package demo.ObjectDimension : Feature.Kernel
operation check : Feature.Check
operation display : Feature.Display
operation eval : Feature.Eval
operation check_process : Feature.Check
operation display_process : Feature.Display
operation eval_process : Feature.Eval
operation guarded_access : Feature.Accessoperation process : Feature.None
```

Figure 10 depicts the dependency of the ratio (probability) $p(u_i) = \frac{f_{ui}}{N_{all}}$ in each hyperslice. The hyperslices H1 and H5 have highest weighted entropy due to the impact of concurrent access implementation.

Future Work

The theoretical underpinning of the entropy-based complexity metrics for applications developed with the tool for multidimensional separation of concerns — Hyper/J — has been presented. We have proposed a suite of complexity metrics for

Figure 7. Comparison of Entropy and Weighted Entropy per Hyperslice (H_n)

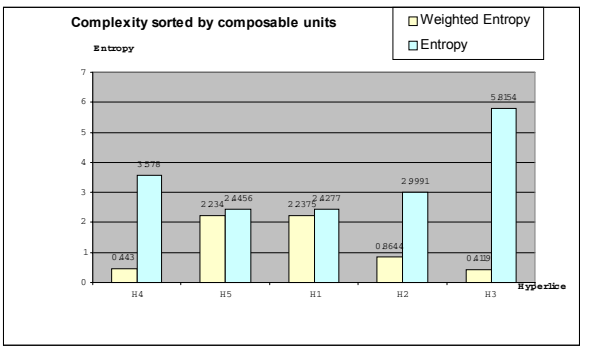
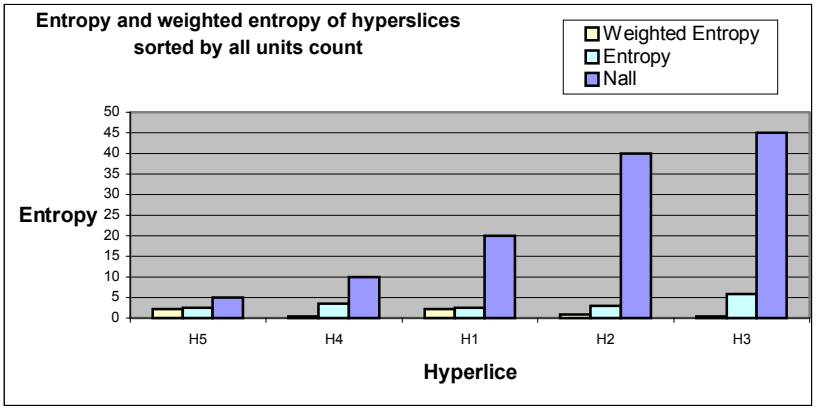


Figure 8. Entropy and Weighted Entropy Sorted by all Units Count (Nall)



ordering of concerns and hyperslices. We have proposed the following complexity measures for development with Hyper/J:

- Hyperslice complexity ranking,
- Concerns matrixes ranking per hypermodule,
- Weighted entropy for ranking units and concerns according to their contribution to utility.

We acknowledge that some aspects of data collection and validation have to be addressed in our future research:

- Data collection rules and measurement framework are being constructed and tested to allow automatic data collection and consistent evaluation in Java classes and Hyper/J. For data collection, we have used the original source code files as well as Hyper/J by-products called unparsed files (*.unp);

Figure 9. Features Kernal, Check and Display and Unit Occurrence Distribution

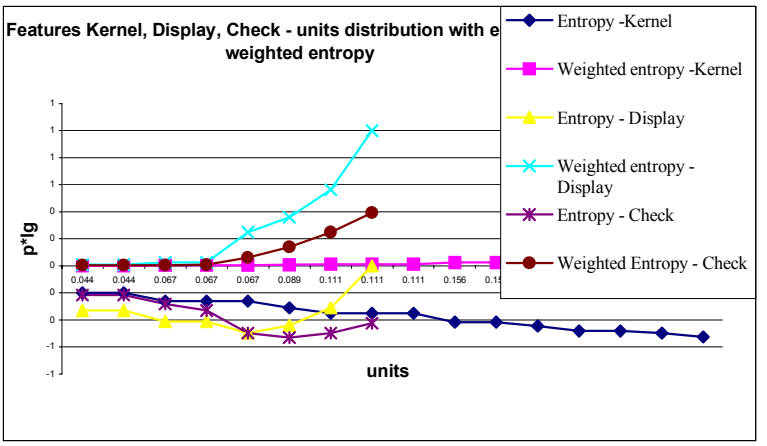
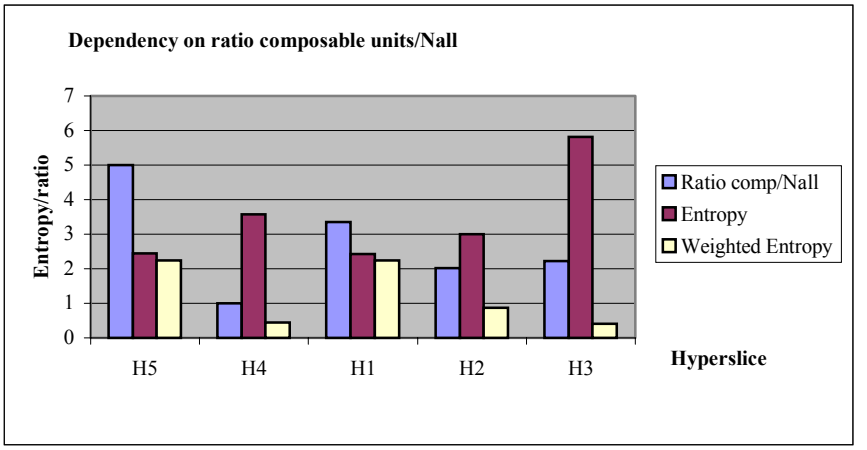


Figure 10. Dependency on Ratio Composable Units/Nall



- Validation study on a larger scale must be conducted covering at least two comparative applications:
  - Case 1: The application is designed and coded without separation of concerns concept in mind;
  - Case 2: The application is designed and coded specifically for Hyper/J.

We also need to produce a clearer approach to the meaning of each entropy-based metric and its impact on application design.

USABILITY REMARKS ON METRICS

This chapter has provided an overview of metrics in used object-oriented software development. Several aspects should be noted, as follows.

For data collection and evaluation process, many metrics presented in this chapter lack a measurement framework, which would allow a consistent comparison and prediction process. Such a framework must contain a precise description (or tool) for the data collection mechanism and metrics explanation (meaning). The entropy-based metrics we have proposed also lack satisfactory framework.

In entropy of class hierarchies, software applications, in particular distributed ones, undergo frequent maintenance change that may lead to chaotic development and, consequently, to disorder. This phenomenon can be expressed as entropy. Object-oriented code is characterized by class hierarchies, which are shared structures. Very often some additional subclassing, modification to existing classes, restructuring of the hierarchy itself, and changes in visibility of attributes and sometimes even methods is needed. Given these changes and possibly lack of comprehensive documentation and time constraints, we assume that class hierar-

chies will become a subject of disorder and exhibit entropic tendencies in the actual code as well as in content. Content entropy is caused by increasing inconsistency in subclassing in the deeper end of the hierarchy. We have observed that the probability that a subclass will not consistently extend the content of its superclass is increasing with the depth of hierarchy. The tools like Hyper/J and Aspect/J support the separation of concerns, thus allowing a different approach to evolving the content rather than extending the class hierarchies.

REFERENCES

Bansiya, J., Davis, C., & Etzkorn, L. (1999). An entropy-based complexity measure for object-oriented designs. *Theory and Practice of Object Systems*, *5*(2), 11-118.

Baumer, D., Riehle, D., Siberski, W., & Wolf, M. (1997). Role object. *Proceedings of the Fourth Annual Conference on the Pattern Languages Of Programming (PLOP) '97*, (September 2-5, pp. 353-369) Monticello, IL.

Belady, L. A., & Lehman, M. M. (1976). A model of a large program development. *IBM Systems Journal*, *15*(3), 225-252.

Bieman, J. M., & Byung-Kyoo, K. (1998, February). Measuring design-level cohesion. *IEEE Transactions on Software Engineering*, *24*(2), 111-124.

Briand, L., Daly, J., & Wurst, A. (1997). *A unified framework for cohesion measurement in object oriented systems.* (Tech. Rep. ISERN-97-05) Retrieved from the WWW on January 15, 2003 at www.iese.fhg.de/network/ISERN/pub/isern_biblio_tech.html#97.

Briand, L., Daly, J., & Wurst, A. (1999, January/February). A unified framework for coupling measurement in object oriented systems. *IEEE Transactions on Software Engineering*, *25*(1), 99-121.

Chidamber, S., & Kemerer, C. (1994, June). A metric suite for object oriented design. *IEEE Transactions on Software Engineering*, *20*(6), 476-449.

Churcher, N., & Shepperd, M. (1994, June). A metric suite for object oriented design. *IEEE Transactions on Software Engineering*, *20(*6), 476-449.

Cover, T. M., & Thomas, J. A. (1991). *Elements of Information Theory*. (Wiley Series in Telecommunications). New York: John Wiley & Sons.

Davis, J. S., & LeBlanc, R. J. (1988). A study of the applicability of complexity measures. *IEEE Transactions on Software Engineering*, *14*(9), 1366-1371.

Dijkstra, E.W. (1976). *A discipline of programing*. Englewood Cliffs, NJ: Prentice Hall.

Eder, J., Kappel, G., & Schrefl, M. (1994). *Coupling and cohesion in object oriented systems*. (Tech. Rep.) University of Klagenfurt.

Fenton, N., & Pfleger, S. L. (1997). *Software metrics: A rigorous & practical approach*. London: International Thomson Computer Press.

Fetchke, T. (1995). *Software Metriken bei der Objectorientierten Programmierung*. Unpublished diploma thesis. GMD, St. Augustin.

Gray, R. M. (1990). *Entropy and information theory.* New York: Springer-Verlag.

Groth, B., Herman, S., Jahnichen, S. & Koch, W. (1995, April). Project integrating reference object library (PIROL): An object-oriented multiple-view SEE. *Proceedings of 7th Conference on Software Engineering Environments (SEE'95)* (pp. 184-193). IEEE Computer Society Press.

Gursaran & Gurdev, R. (2001, April). On the applicability of weyuker property 9 to object oriented structural inheritance complexity metrics. *IEEE Transactions on Software Engineering*, *27*(4), 381-384.

Halstead, M. H. (1977). *Elements Of Software Science*. New York: Elsevier North-Holland.

Harrison, W. (1992, November). An entropy-based measure of software complexity. *IEEE Transactions of Software Engineering* (*18)*(11), 1025-1029.
Henderson-Sellers, B. (1996). *Object-oriented metrics measures of complexity*. Prentice Hall PTR.

Harrison, W., & Osher, H. (1993, September). Subject-oriented programming (a critique of pure objects). *Proceedings of the Conference on Object Oriented Programming: Systems, Languages, and Applications* (pp. 4111-428). Washington, D.C.

Hitz, M., & Montazeri, B. (1995). Measuring product attributes of object-oriented systems. *Proceedings 5th European Software Engineering Conference (ESEC'95*) (pp. 124-136). Barcelona, Spain.

Hitz, M., & Montazeri, B. (1996, April). Chidamber & Kemerer's metric suite: a measurement theory perspective. *IEEE Transactions on Software Engineering, 22*(4), 270-276.

Khaled, E. E., Saida, B., Nishith, G., & Shesh, N. R. (2001, July). The confounding effect of class size on validity of object-oriented metrics. *IEEE Transactions on Software Engineering*, *27*(7).

Khaled, E. E. (DATE NEEDED). A primer on Object-Oriented measurement. *IEEE Software*, 185-187.

Kiczales, G., & Lopes, C. (2001, October). *Tutorial: aspect-oriented programming w/AspectJ™*. Retrieved January 15, 2003 from Xerox PARC Web site: http://*www.parc.xerox.com/aop*.

Kiczales, G., Ashley, J. M., Rodriguez, L., Vanhdat, A., & Bobrow, D. G. (1993). Metaobject protocols: Why we want them and what else they can do. In A. Paepcke (Ed.), *Object oriented programming: the CLOS perspective* (pp. 101-118). MIT Press.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., & Irwin, J. (1997). *Aspect oriented programming*. Retrieved Xerox Corp. Web site: http://www.parc.xerox.com/spl/projects/aop/.

Kim, E. M., Chang, O. B., Kusumoto, S., & Kikuno, T. (1994). Analysis of metrics

for object-oriented program complexity. *Proceedings of 18th COMPSAC* (pp. 201-207) Computer Society Press.

Kitchenham, B. (1996). *Software metrics*. Oxford, UK: Blackwell.

Krantz, D. H., Luce, R. D., Suppes, P., & Tversky, A. (1971). Foundations of measurement. In *Additive and Polynomial Representation* (Vol. 1). New York: Academic Press.

Lee, D. (2000). Concurrent programming in Java. In *Design Principles and Patterns* (2nd Ed.). Addison-Wesley.

McCabe, T. (1976, December). A complexity measure. *IEEE Transactions on Software Engineering*, *SE-2*(4), 308-320.

Ossher, H., & Tarr, P. (1998). Multi-dimensional separation of concerns and the hyperspace approach (Tech. Rep.). New York: IBM T.J. Watson Research Center.

Ossher, H., & Tarr, P. (1998). *Multi-dimensional separation of concerns in hyperspace* (Position paper). New York: IBM T.J. Watson Research Center.

Roberts, F. S. (1979). Measurement theory with applications to decision making, utility, and social sciences. In *Encyclopedia of mathematics and its applications*. Addison Wesley Publishing Co..

Shannon, C. E. (1949). *A mathematical theory of communication.* Illinois: University of Illinois Press.

Tarr, P., Ossher, H., Harrison, W., & Sutton, Jr. (1999). N degrees of separation: multi-dimensional separation of concerns. *Proceedings Of the 21st International Conference on Software Engineering*.

Weyuker, E. J. (1988, September). Evaluating software complexity measures. *IEEE Transactions on Software Engineering*, *14*(9), 1357–1365.

Yacoub S. M., Ammar, H. H., & Robinson, T. (1999, November 4-6). Dynamic metrics for object oriented designs. *Proceedings of 6th International Symposium on Software Metrics (METRICS'99)* (pp. 50-61). Boca Raton, FL.

Zuse, H. (1994). Software complexity metrics/analysis. In J. Marciniak (Ed.), *Encyclopedia of software engineering* (Vol. I, pp. 131-166). John Wiley& Sons, Inc.

Zuse, H. (1991). *Software complexity measures and methods*. Berlin, Germany: De Gruyter Publisher.

Zweben, S., & Hasltead, M. (1979, March). The frequency distribution of operators in PL/I programs. *IEEE Transactions of Software Engineering*, *SE-5*, 91-95.

ENDNOTES

1 It was derived from Boltzmann's statistical mechanics, 2nd termodynamic law.

2 Average number of tokens between error occurrences.

Chapter X

Temporal Interaction Diagrams for Multi-Process Environments

T. Y. Chen
Swinburne University of Technology, Australia

Iyad Rahwan
University of Melbourne, Australia

Yun Yang
Swinburne University of Technology, Australia

ABSTRACT

This chapter introduces a novel notion of temporal interaction diagrams for distributed and parallel programming. An interaction diagram is a graphical view of computation processes and communication between different entities in distributed and parallel processes. It can be used for the specification, implementation and testing of interaction policies in distributed and parallel systems. Expressing interaction diagrams in a linear form, known as

fragmentation, facilitate automation of design and testing of such systems. Existing interaction diagram formalisms lack the flexibility and capability of describing more general temporal order constraints. They only support rigid temporal order, and, hence, have limited semantic expressiveness. We propose an improved interaction diagram formalism in which more general temporal constraints can be expressed. This enables us to capture multiple valid interaction sequences using a single interaction diagram.

INTRODUCTION

Various attempts have been made to formalize interaction among computational entities, such as distributed, parallel, object-oriented and multi-agent systems (Hoare, 1985; Magee, Dulay, Eisenbach & Kramer, 1995; Ronnquist & Low, 1996; Koskimies, Männistö, Systä & Tuomi, 1996; Kinny, 1998; Chen, Rahwan & Yang, 2002; Bauer, 1999; Bauer, Müller & Odell, 2001). Traditionally, the system design stage involves a description of the steps taken in processing a particular task. In distributed systems, however, the implementation requires a clear picture of the separate computational threads of different processes. In parallel systems, it would be very helpful to be able to express the computation flow descriptions for different processes. Interaction diagrams are designed to support the representation and processing of these mingling activities. The linear representation of these diagrams facilitates the automation of the processes of diagram manipulation for design, report generation and testing. In particular, testing can be performed by comparing execution traces against specifications expressed in terms of interaction diagrams.

However, existing interaction diagram formalisms support quite rigid temporal order constraints only. An execution trace is said to be valid if it satisfies the interaction diagram. In other words, there is no way of specifying multiple valid traces without writing multiple versions of fixed execution traces. This can become a difficult job, especially in systems with sophisticated interactions. In such systems, the number of valid interactions can be quite large, and there is a demand to more concisely express such flexibility in a single interaction diagram. Our research is a step towards achieving this goal.

In this chapter, we give an overview of existing interaction diagram formalisms and present an enhancement of a particular framework in such a way as to support more flexible interaction specification. The chapter is organized as follows. In the next section, we present an overview of an existing interaction diagram framework and introduce the basic notation to be used throughout this chapter. Then we introduce our novel extension. After that, an example demonstrating the merit of our framework is presented. Then we discuss current and future trends in interaction diagram frameworks. Finally, we conclude and summarize our ideas and results.

BACKGROUND

In this section, we present a particular interaction diagram model proposed in Ronnquist and Low (1996). The work described in this chapter is an extension of this framework. An interaction diagram is a graphical representation of the computation threads and communications in a distributed system and the like. It is a graph showing each process symbolically as one or more vertical bars and the messaging between processes as horizontal arrows between these bars. A sample interaction diagram is shown in Figure 1, describing three processes A, B and C and the message sequences among them. There are a number of properties that are worth mentioning with respect to the meaning of this interaction diagram according to the formalism in Ronnquist and Low (1996).

A fragmentation is an algebraic representation of an interaction diagram. In order to convert an interaction diagram into a fragmentation, we need to decompose it into its graphical elements, which correspond to fragments. Figure 2 shows some types of these fragments. These fragments include the beginning of a process, the end of a process, a process sending a message and a process receiving a message. Underneath every fragment is its corresponding algebraic form. Algebraic atoms can be used (forming a fragmentation) to describe a particular interaction diagram.

There are two types of fragmentation, however, we are only interested in computation flow fragmentation (Ronnquist & Low, 1996), which orders the fragments by grouping those of the same process together. Each process' fragments are ordered according to a top-to-bottom sequence (or temporal sequence). This represents the computation flow of a process. The computation flow fragmentations for Figure 1 are as follows:

For process A: <beg(A), snd(A,m1), rcv(A,m4), end(A)>;
For process B: <beg(B), rcv(B,m1), snd(B,m2), rcv(B,m3), snd(B,m4), end(B)>;
For process C: <beg(C), rcv(C,m2), snd(C,m3), end(C)>.

Figure 1. Simple Interaction Diagram

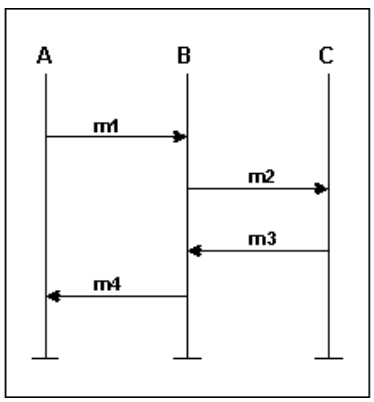
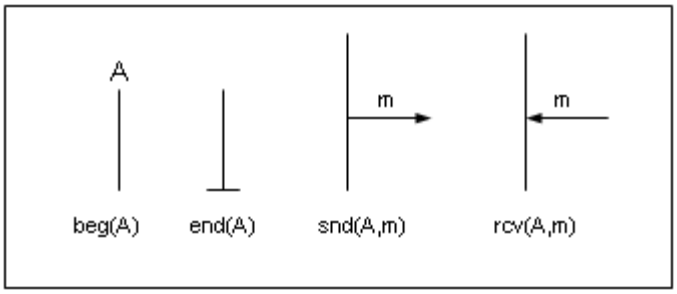


Figure 2. Graphical Elements (Fragments)



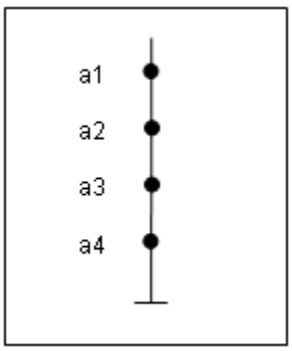
According to Ronnquist and Low (1996), when actions concern a single process, the order of actions has significance for the interpretation of the diagram. The meaning of a single process computation flow is that the order in which the fragments occur reflects the order in which the corresponding actions take place. In other words, they state the exact temporal order in which the fragments or messages should take place. There is no mechanism to express the temporal constraints between different fragments. This is a severe limitation to the expressiveness of the framework. What we would like to achieve is a formal framework for supporting more flexibility in the temporal constraints.

In order to explain the limitation more clearly, we will give a simple example of one process. Suppose we would like to express the following temporal order constraints on the events of the diagram in Figure 3, where an event may be send (snd) or receive (rcv).

We use a coffee-making machine as an example for an illustration. Suppose one process needs to perform the following tasks:

- First, send a message to the resources process requesting sugar and coffee (call this event a1).

Figure 3. Single-Process Diagram



- Then, receive a message from the resources process indicating sugar is ready (call this event a2). Also receive another message from the resources process indicating coffee is ready (call this event a3). It does not matter which of these acknowledgments occurs first, but they both need to be completed before moving to the next step.
- Now, send a message to the water process requesting hot water to be poured in the cup (call this event a4).

More abstractly, we have the following constraints:

- a1 must be before a2, a3 and a4;
- the order of a2 and a3 is not important;
- a4 must take place after both a2 and a3.

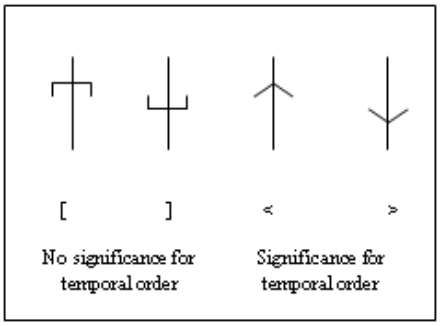
In fact, we can see the problem as follows. In a parallel system design and communication specification stage, instead of specifying a single valid execution sequence that must be followed, we may like to specify a number of valid sequences. Recall that interaction diagrams can be used for checking execution traces against specified order. In the example above, the designer's intention is to treat both event sequences a1, a2, a3, a4 and a1, a3, a2, a4 as valid. Using the old formalism, we need to provide two different (rigid) interaction diagrams. If the execution trace satisfies one of these diagrams, then the trace is correct. This technique becomes less practical in more complex settings where the number of possible valid sequences of event is large. In such a situation, a separate interaction diagram needs to be provided for each valid sequence and checking needs to be carried out against all interaction diagrams until one (or none) matches. In the next section, we present an extension to the existing framework, which supports more general and flexible constraints.

PROPOSED ENHANCEMENTS

In this section, we propose the enhancement of the current interaction diagram formalism as follows. We would like to distinguish between two types of temporal order constraints: namely, groups of events among which the order is important, and groups of events among which the order is not important.

We will extend the set of graphical elements by including the fragments shown in Figure 4. A couple of fragments, corresponding to the "<" and ">" symbols (we will call them angular brackets), represent the start and end of a group of fragments among which the top-to-bottom order is the actual temporal order (that is as in the original formalism). This temporal order also states that, in addition to the specified temporal order, no external event is allowed to interleave with the enclosed events. The other couple of fragments, corresponding to the "[" and "]" symbols (we call them square brackets), represent the start and end of a group of fragments among

Figure 4. Ordering Fragments



which the top-to-bottom order does not necessarily reflect the actual temporal order. Throughout this chapter, we will use the term "bracket" to refer to all types of brackets.

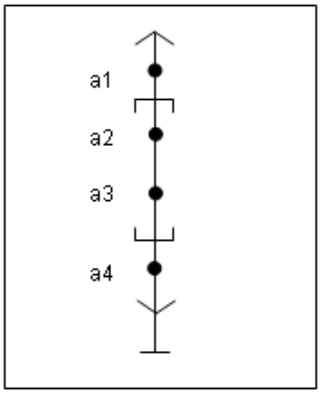
A sample interaction diagram using some of the proposed fragments is shown in Figure 5. This interaction diagram expresses the temporal constraints mentioned in the previous section, which the previous formalism failed to express. Simply, the temporal order of the events within the square brackets is not important. However event a1 must precede this group, and a4 must take place after it. The following is the corresponding modified computation flow fragmentation. (Note that a1, a2, etc., are placeholders for normal fragments, such as snd and rcv):

< beg(A), a1, [a2, a3], a4, end(A) >.

Note that this fragmentation is semantically equivalent to the following fragmentation (that is, changing the order of events inside the square brackets does not affect the meaning of the diagram):

< beg(A), a1, [a3, a2], a4, end(A) >.

Figure 5. Extended Single-Process Diagram



These fragmentations have the following properties:

- An opening fragment of a particular type should appear before any closing fragment of that type.
- At any stage of parsing the fragmentation, the number of opening fragments must be greater than or equal to the number of corresponding closing fragments of the same type.
- At the end of the interaction diagram (or the corresponding fragmentation), the number of opening and closing fragments should be equal (that is each opening fragment should have a corresponding closing fragment).
- Overlapping groups are not permitted, that is, if a bracket of group g1 opens, and within that a bracket of group g2 opens, then the closing bracket of group g2 must close before the closing bracket of group g1 does. This means that each bracket is matched with the nearest corresponding bracket.
- An entity is either an atomic event or a group of entities enclosed within a matched pair of brackets. All entities are treated the same with respect to the higher level group to which they belong.
- The temporal relation between any two events is determined by the type of the nearest complete pair of brackets (angular or square) that contains them.

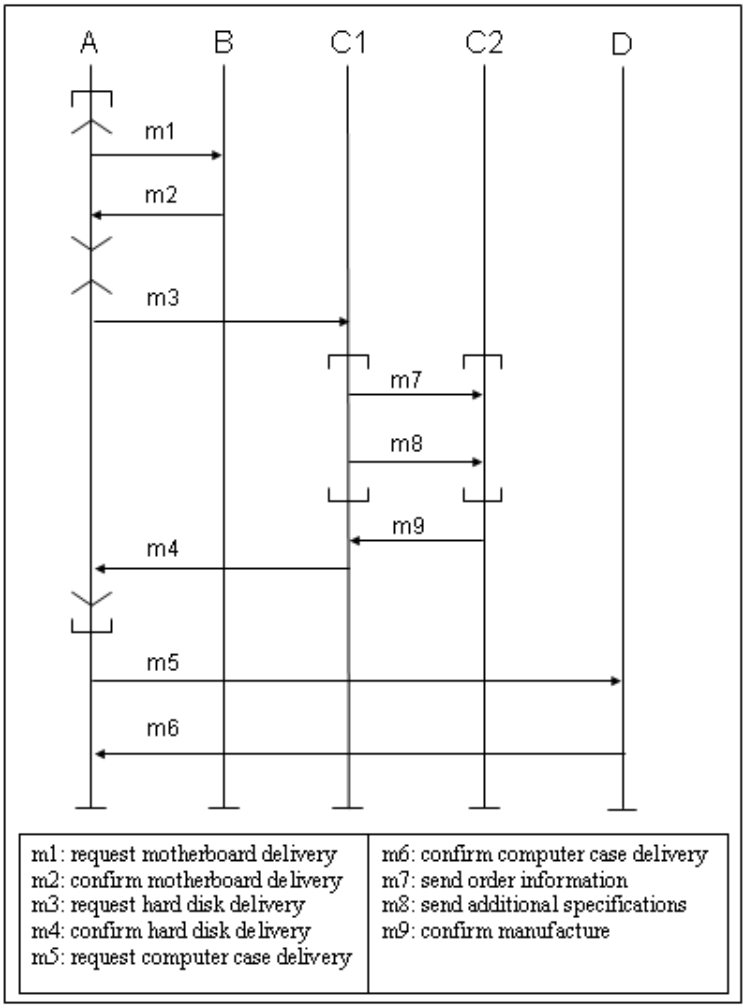
EXAMPLE AND OBSERVATIONS

In this section, we show another example to further illustrate our notation. We will exploit the supply chain automation domain. This domain is a typical example of situations in which there is a need for flexible specification of multiprocess interactions. Suppose we have five processes:

- *Process A:* Representative of a personal computer (PC) manufacturing company. This company does not manufacture all computer parts, but rather purchases them from known partners (shown below). There are partners from whom this party purchases motherboards, hard disks and computer cases.
- *Processes B, C1 and D:* Representatives of manufacturers for PC motherboard, hard disks, and computer cases, respectively.
- *Process C2:* Representative of the manufacturing plant associated with process C1.

Figure 6 shows the interactions between the different processes, that is different messages passed between them. For simplicity we do not include angular brackets at the beginning and the end of every process. The intuition behind the diagram with respect to process A is that, before sending the order for computer cases, A first needs to order and receive motherboards and hard disks. This may be because it is very costly to store computer cases, so it would be more efficient to order them after all other components have been received. This way computer assembly

Figure 6. PC Manufacturing Example



would take place as soon as the cases arrive. Intuitively no deliveries should take place unless an order has been placed. This is reflected in the fact that no confirmation message can be received before the order. Moreover ordering motherboards and ordering hard disks can take place in any order. It does not matter which order process is performed first since assembly will not take place until the receipt of cases.

Now let us consider processes C1 and C2. After receiving a request for hard disk delivery through message m3, C1 must submit the order information to its manufacturing plant. C1 also needs to send additional technical specifications circulated internally from its design team (also represented by C1). It does not matter

in what order these requests take place, but it is only after receiving both messages that the manufacturing plant can start manufacturing. This is why m9 can only be sent after receiving both m7 and m8. Note that according to the diagram, there is nothing to prevent messages m7 and m8 from being sent in one order and received in another. For example, C1 might send m7 followed by m8, but, due to network latency, C2 might receive m8 before m7. Processes B and D only need to receive orders before they confirm delivery. The computation flow fragmentations of the system in Figure 6 are as follows:

<beg(A), [<snd(A,m1), rcv(A,m2)>, <snd(A,m3), rcv(A,m4)>], snd(A,m5), rcv(A,m6), end(A)>
<beg(B), rcv(B,m1), snd(B,m2), end(B)>
<beg(C1), rcv(C1,m3), [snd(C1, m7), snd(C1, m8)], rcv(C1, m9, snd(C1,m4), end(C1)>
<beg(D), rcv(D,m5), snd(D,m6), end(D)>
<beg(C2), [rcv(C2,m7), rcv(C2, m8)], snd(C2,m9), end(C2)>.

Observation 1

Recall that the power of our framework stems from its ability to capture multiple valid execution or interaction sequences in a single diagram. Following are some interaction sequences that satisfy the diagram presented in Figure 6.

1. beg(A), beg(B), beg(C1), beg(C2), beg(D), snd(A, m1), rcv(B, m1), snd(B, m2), rcv(A, m2), snd(A, m3), rcv(C1, m3), **snd(C1, m7), rcv(C2, m7), snd(C1, m8), rcv(C2, m8)**, snd(C2, m9), rcv(C1, m9), snd(C1, m4), rcv(A, m4), snd(A, m5), rcv(D, m5), snd(D, m6), rcv(A, m6), end(A), end(B), end(C1), end(C2), end(D).
2. beg(A), beg(B), beg(C1), beg(C2), beg(D), snd(A, m1), rcv(B, m1), snd(B, m2), rcv(A, m2), snd(A, m3), rcv(C1, m3), **snd(C1, m7), snd(C1, m8), rcv(C2, m7), rcv(C2, m8)**, snd(C2, m9), rcv(C1, m9), snd(C1, m4), rcv(A, m4), snd(A, m5), rcv(D, m5), snd(D, m6), rcv(A, m6), end(A), end(B), end(C1), end(C2), end(D).
3. beg(A), beg(B), beg(C1), beg(C2), beg(D), *snd(A, m1), rcv(B, m1)*, snd(B, m2), rcv(A, m2), *snd(A, m3), rcv(C1, m3)*, **snd(C1, m8), snd(C1, m7), rcv(C2, m7), rcv(C2, m8)**, snd(C2, m9), rcv(C1, m9), snd(C1, m4), rcv(A, m4), snd(A, m5), rcv(D, m5), snd(D, m6), rcv(A, m6), end(A), end(B), end(C1), end(C2), end(D).
4. beg(A), beg(B), beg(C1), beg(C2), beg(D), *snd(A, m3), rcv(C1, m3)*, snd(C1, m8), snd(C1, m7), rcv(C2, m7), rcv(C2, m8), snd(C2, m9), rcv(C1, m9), snd(C1, m4), rcv(A, m4), *snd(A, m1), rcv(B, m1)*, snd(B, m2), rcv(A, m2), snd(A, m5), rcv(D, m5), snd(D, m6), rcv(A, m6), end(A), end(B), end(C1), end(C2), end(D).

Interaction sequences 2 and 3 differ from sequence 1 in the relative order for messages m7 and m8, which is highlighted in bold. Sequence 4 differs from sequence

3 in the relative order for messages m1 and m3 (that is in the order of the processes for ordering motherboards and hard disks), which is highlighted in italic. Note that the above are only examples of some valid sequences. Other valid sequences include all other combinations of orderings of interaction between processes A and C1 with the interaction between C1 and C2. Also, processes can start and terminate in different orders as long as each process is created before it starts sending or receiving messages and terminates at the very end. For example, it does not matter which of beg(C2) or beg(D) takes place first.

We have just seen that using the relatively simple diagram presented in Figure 6, we are able to specify a large variety of valid interaction sequences among processes (or objects or agents). We do not have to specify multiple separate interaction diagrams to cater for all these sequences. This results in significant saving of time and space.

Observation 2

Note that in the above fragmentation for process A, the process of ordering motherboards must completely finish (the request as well the reply must both take place) before the process of ordering hard disks starts or vice versa. This seems unnatural. One alternative approach is to replace the following notation:

[<snd(A,m1), rcv(A,m2)>, <snd(A,m3), rcv(A,m4)>]

with the following:

< [snd(A,m1), snd(A,m3)], [rcv(A,m2), rcv(A,m4)] >.

This means that sending out the first two orders can happen in any order, and receiving the corresponding responses may also happen in any order with the only condition being that both orders should be sent out before any order gets received. On the other hand, we might want to express that they can both happen in any order as long as no reply occurs before a request without disallowing a response to be received before the second order is sent. This is one of the limitations of this formalism. This is due to the fact that we do not allow interleaving between events belonging to different bracket pairs. We treat all events within a bracket pair as a single entity with respect to all other events outside that pair.

CURRENT AND FUTURE TRENDS

Software engineering practitioners face many challenges in the 21st century. With the rate at which software systems are scaling up and with the necessity of extensive interaction among distributed software and computer systems, there is an urgent need for sound software engineering frameworks that allow people to deal

with such complexity. In distributed systems, it is very important to have efficient tools for the design, manipulation, deployment and testing of distributed computational entities. In Koskimies, Männistö, Systä and Tuomi (1996), for example, "scenario diagrams" have been used to study the interactions among object-oriented systems in dynamic object models. SCED (Koskimies, Männistö, Systä & Tuomi, 1998) is a tool for using scenario diagrams for specifying object systems and generating state machines for describing the behavior of each object from the scenario diagram specification.

Low, Ronnquist and Chen (1997) presented the interaction diagram framework on which this work was based. They have built a tool for specifying interaction diagrams among multiple computational threads. They showed how an interaction diagram may be represented in different types of symbolic notations (called fragmentations). They presented a tool for translating an interaction diagram into fragmentations and vice versa.

Interaction diagrams also have many important applications in multi-agent systems (Jennings, 2001; Wooldridge, Jennings & Kinny, 2000; Wooldridge, 1999). Jennings and Wooldridge (1998) proposed that an intelligent agent is capable of social behavior. That is, it is capable of communicating with other agents in the system. In this context, there is a need for formalizing such interaction in multi-agent systems. Such formalization would be useful for designing and testing multi-agent systems, as well as visualizing the computation flow of each agent and communication among agents.

Bauer, Müller and Odell (2001) presented an extension of the unified modeling language (UML), a de facto standard for object-oriented analysis and design, by adapting it for modeling protocols in multi-agent interaction, resulting in the Agent UML framework. Huget (in press, a) used this framework to specify interaction in a supply chain management domain. Furthermore, Huget (in press, b) presented a framework for communicating UML interaction protocol diagrams among agents using an XML-based language.

We envision many important developments in the field of interaction frameworks in the next few years. Frameworks will emerge that facilitate different levels of interaction among computer systems. There is an urgent need for standardizing methodologies for specifying interaction such that interaction specifications become more portable. Algorithms need to be put into place for manipulating interaction diagrams and for checking the validity of interaction traces. Furthermore, there will be a need for formal verification of the computational properties of those algorithms.

CONCLUSION

Existing interaction diagram formalisms and their corresponding linear notations (known as fragmentations) require different strict interaction diagrams for different execution sequences. This is because each interaction diagram is capable of

expressing only a single, strict valid execution sequence. In this chapter, we have presented a formalism for describing flexible interaction diagrams that are able to express many possible execution sequences using one interaction diagram. Our formalism describes interaction diagrams that have a mix of two types of temporal relationships between events in a process — ordered and unordered temporal relationships. An ordered temporal relationship means that events should take place in the order specified, while an unordered temporal relationship means that the order of executing two events is irrelevant. We have shown how combinations of these two types of relationships could be used to represent more complicated scenarios through a linear fragmentation.

REFERENCES

Bauer, B. (1999). Extending *UML for the specification of agent interaction protocols*. (OMG Publication No. ad/99-12-03). Analysis and Design Task Force (ADTF) Concord, CA: FIPA.

Bauer, B., Müller, J. P., & Odell, J. (2001). Agent UML: A formalism for specifying multi-agent software systems. *International Journal of Software Engineering and Knowledge Engineering, 11*(3), 207-230.

Chen, T. Y., Rahwan, I., & Yang Y. (2002). Temporal interaction diagrams. *Proceedings of the 2002 Information Resource Management Association International Conference* (pp. 843-846). Seattle, WA. Hershey, PA: Information Resources Management Association.

Hoare, A. (1985). *Communicating sequential processes*. Englewood Cliffs, NJ: Prentice Hall.

Huget, M. P. (in press, a). An application of agent UML to supply chain management. In P. Giorgini, Y. Lesperance, G. Wagner, and E. Yu (Ed.), *Proceedings of Agent Oriented Information System.* Bologna, Italy, iCue Publishing, Berlin.

Huget, M. P. (in press, b). Extending agent UML protocol diagrams. *Proceedings of Agent Oriented Software Engineering.* Lecture Notes in Computer Science. Bologna, Italy. Springer.

Jennings, N. R. (2001). An agent-based approach for building complex software systems. *Communications of the ACM, 44*(4), 35-41.

Jennings, N. R., & Wooldridge, M. (1998). Applications of intelligent agents. In N.R. Jennings & M. Wooldridge (Eds.), *Agent Technology: Foundations, Applications, and Markets* (pp. 3-28). Berlin, Germany: Springer-Verlag.

Kinny, D. (1998). The AGENTIS agent interaction model. *Proceedings of the 5th International Workshop on Agent Theories, Architectures, and Languages* (pp. 331-344) Lecture Notes in Artificial Intelligence, Vol. 1555, 1999. Paris, France. Berlin, Germany: Springer.

Koskimies, K., Männistö, T., Systä, T., & Tuomi, J. (1996). On the role of scenarios in object-oriented software design. *Proceedings of Nordic Workshop on Programming Environment Research* (pp. 53-57). Denmark, Aalborg University. Aalborg: Computer Science Departmentat Aalbrog University.

Koskimies, K., Männistö, T., Systä, T., & Tuomi, J. (1998). Automated support for modeling of OO software. *IEEE Software, 15*(1), 87-94.

Low, C. K., Ronnquist, R., & Chen, T. Y. (1997). An automated tool (IDAF) to manipulate interaction diagrams and fragmentations for multi-agent systems. *International Journal of Software Engineering and Knowledge Engineering, 9*(1), 127-149.

Magee, J., Dulay, N., Eisenbach S., & Kramer J. (1995). Specifying distributed software architectures. *Proceedings of EsEC'95, Lecture Notes in Computer Science, 989*, 137-153.

Ronnquist, R., & Low, C. K. (1996). Formalisation of interaction diagrams. *Proceedings of the 3rd Asia-Pacific Software Engineering Conference* (pp. 318-327). Seoul, Korea. Boston, MA: IEEE.

Wooldridge, M. (1999). Intelligent agents. In G. Weiss (Ed.), *Multiagent systems* (pp. 27-78). MIT Press.

Wooldridge, M., Jennings, N. R., & Kinny, D. (2000). The Gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems, 3*, 285-312. Cambridge, MA.

Section III

Applications and Implementations

Chapter XI

Toward an Integrative Model of Application-Software Security

Vijay V. Raghavan
Northern Kentucky University, USA

ABSTRACT

Populist approaches to studying information systems security include architectural, infrastructure-related and system-level security. This study focuses on software security implemented and monitored during systems development and implementation stages. Moving away from the past checklist methods of studying software security, this study provides a model that could be used in categorizing checklists into meaningful clusters. Many constructs, such as principle of least privilege, execution monitoring, social engineering and formalism and pragmatism in security implementations, are identified in the model. The identification of useful constructs to study can form the basis of evaluating security in software systems as well as provide guidelines of implementing security in new systems developed.

INTRODUCTION

While academicians and industry practitioners have long recognized the need for securing information systems and computer architectures, there has recently been a heightened awareness of information technology (IT) management on computer-related security issues (Hulme, 2001). IT managers are increasingly worried about possible attacks on computer facilities and software, especially for mission critical software. There are indeed many dimensions to providing a secure computing environment for an organization, including computer viruses, Trojan horses, unauthorized accesses and intrusions and thefts to infrastructure. This complexity and multidimensional nature of establishing computer security require that the problem be tackled at many fronts simultaneously. Research in the area of information systems security has traditionally focused on architectural-, infrastructure- and systems-level security (Oppliger, 1997; Nelson, 1997). Emerging literature on application-level security, while providing useful paradigms, remains isolated and disparate (James, Joshi Walid, Aref & Spafford, 2001; Schneider, 2000; Bakersville 1993; Landerwehr, 1981). The current study focuses on a single, albeit an important, dimension of providing a safe and secure computing environment — application-software security.

THEORETICAL FOUNDATIONS

One of the difficulties in specifying data security requirements for an application is its complexity. Ting (1993) states that the characteristics of application-dependent security policies and requirements have not been clearly understood due to this complexity. Bellovin (2001) affirms that we cannot have "secure computer systems until we can build correct systems" and points out that "we don't know how to accomplish this, and probably never will." Schneider (2000) supports this view by highlighting the need for application-dependent special-purpose security policies. Current notions of architectural and infrastructure security do provide checklist items that could be transformed to an application-security context. A clear understanding as well as synthesizing current paradigms of computer security and transplanting relevant ideas to an application-development context is essential to move toward an integrative model of application-software security. Landwehr (1981) argues that formal models of computer security help designers decide exactly what "secure" means for their particular needs. In addition security regulations written in plain English tend to be "descriptive" rather than being "prescriptive" as in formal models.

Bakersville (1993), in his seminal exposition of application-development related security issues, finds the metaphor of "generations" useful in identifying the evolution of computer security paradigms. His exposition is akin to Nolan's stage hypothesis that is well known to the IS community. Bakersville identifies three major phases of evolution in computer security literature: the early checklist methods, second generation mechanistic-engineering methods and a third generation logical-transfor-

mational method. Checklist methods provide a set of guidelines for computer security audits. Mechanistic-engineering methods are founded on the belief that a detailed examination of the functional requirements of the system will lead to security specification details of the system. Third generation logical transformational methods rely on modeling essential security attributes of a system. While there cannot be a great deal of disagreement on these "phases" of our understanding of security concerns, we cannot discard checklist methods as obsolete and, hence, not useful in the software-development context. A model of software security that uses checklists as not simply an unrelated collection of items but clusters of items identifying well-known concepts evolving from security literature would form the core of our integrative model of application-software security. Bakersville (1993) claims that published security checklists were designed as guidelines for evaluating software systems and not for specifying its security attributes. However it is not entirely clear from his discussion as to why the criteria established for evaluation of security cannot be used in specifying security during the design stage. It is only beneficial if designers and developers of software systems are made aware of the criteria for a later security assessment of the system. A premise of the current endeavor of developing an integrative model is that a checklist based on sound theoretical constructs as elicited from prior studies would provide practical guidelines for application-software development. There are three major security checklists available: SAFE: Security audit and field evaluation for computer facilities developed by Krauss (1972, 1980); Computer Security Handbook (Hoyt, 1973; Hutt, Bosworth & Hoyt, 1988) and a checklist for computer center self-audits compiled by Browne (1979). These checklists concern themselves with all forms of security including architectural and software related. Cohen (1999) provides a list of guidelines to follow during systems development that will make the systems more secure.

Our immediate goal is to first identify the concepts that will enable us to view these checklists as identifying useful security constructs rather unrelated clusters of items dealing with different processes. We use the framework presented by Webster and Watson (2002) to synthesize the past work in the area of computer security in an effort to identify the constructs. An important contribution for security design has been made by Saltzer and Schroeder (1975). In spite of its date, many modern writers (Bace, 1999; Morales, 2000) have recognized the power of the fail-safe security design principles enumerated by them. They are: (1) Least privilege. Relinquish access when it is not required; (2) Fail-safe defaults. When the power goes off, the lock should be closed; (3) Economy of mechanism. Keep things as small and simple as possible; (4) Complete mediation. Check every access to every object; (5) Open design. Do not attempt "security by obscurity," as Bace (1999) puts it; assume the adversary can find your hiding places; (6) Separation principle. Do not make privilege decisions based only on a single criterion; use the onion-skin model; (7) Least common mechanism. Minimize shared channels and (8) Psychological acceptability. Make security painless, transparent and ubiquitous (Saltzer & Schroeder, 1975).

Table 1.

Authors	Constructs	Goal of work
Saltzer and Schroeder (1975).	Least privilege, Fail-safe defaults, Economy of mechanism, Complete mediation, Open design, Separation of principle, Least common mechanism, Psychological acceptability	General Application Dependent Security
Schneider (2000)	Access Control, Information Flow, Availability, Execution Monitoring, Least Privilege	Developing Enforceable Security Policies
Morales (2000)	Social Engineering (fraudulent deceptive requests for confidential information)	Intrusion Detection
Nelson (1997)	Formalism, Pragmatism	Architectural Security

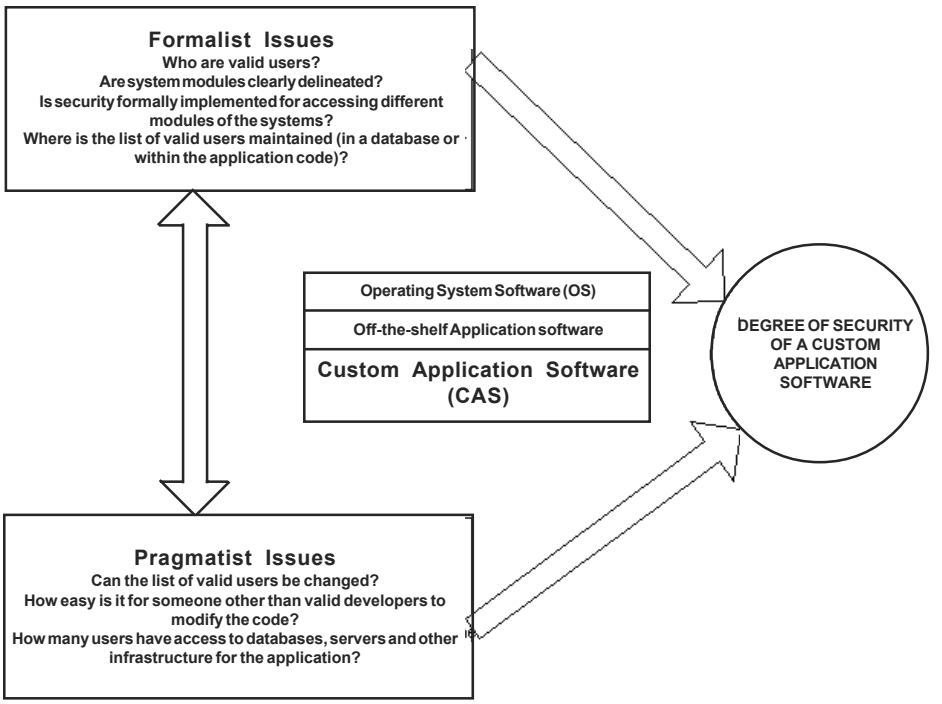
A useful method of establishing security in software is to allow and disallow performing operations on critical resources based on users and their roles. In this role-based security, Nelson (1997) identifies two distinct approaches to security research: Formalist school that focuses on correctness and universality by attempting to ensure system-independent methods (notably, access control) and pragmatist stream that focuses on attacks on and countermeasures for real systems (notably intrusion detection). Nelson (1997) has argued for integration of these two schools of research. Nelson is also of the opinion that traditional research on security methods has been divorced from mainstream systems design, development and operation. The present study attempts to evaluate current practices of software security of Custom Software Applications. Security issues relating to the two major streams of security research will be identified and evaluated. The concept of Run-Time Security Evaluation (RTSE) proposed by Serban and McMillin (1996) highlights the need to provide an ongoing security evaluation after the system is operational. They advocate RTSE in addition to securities implemented during developmental life cycles of a CSA.

Formalist Issues of Software Security: In order for a software application to be secure, a group of valid users must be designated. The responsibility for maintaining this list of valid users must be accounted for in the design of the system. It is also conceivable that the application must have distinct components, which are open to users with different levels of security. These different modules must have distinct entry points that can be controlled. Multiple entry points for different modules must be clearly identified and checks must be performed before users are allowed to enter those modules. A matrix list of valid users and the components that they are allowed to access must be an integral part of the security model for the application. The location of this matrix, whether it is part of a secure database that is accessed by the application or hard coded into the application, may determine the security level of the application.

Pragmatist Issues of Software Security: The set of issues under the pragmatist school focus on deliberate intrusion into the system after it is built. Even in cases where the application designers have exercised sufficient care in implementing a security model of the system, once the system is operational there is potential for security breaches through the infrastructure where the application is running. As an example, if a list of valid users and modules are maintained in a database, there is still a potential for security breaches if access to databases is not sufficiently controlled. Although this is often outside the responsibility of application developers, it still presents a potential security problem for the system. The location of the application code and the physical security of the code are critical for the security level of the CSA. In addition, if the infrastructure (access to operating system, servers and database servers) in which the application is running is not secure, it will compromise the security of the CSA as well.

Although formalism and pragmatism have been identified as two distinct streams of security research, the degree of security that a CSA enjoys is a function of formalist and pragmatist security concerns. It is imperative that they both are considered during System Development Life Cycle (SDLC), and security research must attempt to integrate them (Nelson, 1997).

Figure 1. An Integrative Model of Software Security



CONCLUSION

This model allows us to progress from a purely checklist approach to specifying and evaluating security requirements of custom-software applications. It provides a structured way of categorizing software requirements that can lend itself to better monitoring of security requirements during application development and implementation stages. It can also lead to development of instruments to measure the rigor of security in applications enforced during systems development process and later on in the implementation of those systems.

REFERENCES

Bakersville, R. (1993). Information systems security design methods: Implications for information systems development. *ACM Computing Surveys, 25*(4), 375-414.

Bellovin, S.M. (2001). Computer Security — An End State? *Communications of ACM*, 44(3), 131-132.

Browne, P. (1979). *Security: Checklist for computer center self audits*. Arlington, VA: AFIPS Press.

Cohen, F. (1999, October). Achieving airtight code. *Software Development.* Retrieved May 28, 2002, from http://www.sdmagazine.com/print/documentID=11278.

Department of Defense Trusted Computer Security Evaluation Criteria (1985, December). DoD 5200-28-STD, National Computer Security Center.

Hoyt, D. (1973). *Computer Security Handbook*. New York: MacMillan.

Hutt, A., Bosworth, S., & Hoyt, D. (Eds.). (1988). *Computer security handbook* (2nd ed.). New York: Macmillan.

Hulme, G. V. (2001). Management takes notice. *Information Week*, 28-34. September 3.

James, B. D., Joshi W. G., Aref, A. G., & Spafford, E. H. (2001, February). Security models of web-based applications. *Communications of the ACM, 44*(2), 38-44.

Krauss, L. (1972). SAFE: Security audit and field evaluation for computer facilities and information (revised ed.). New York: Amacon.

Krauss, L. (1980). SAFE: Security audit and field evaluation for computer facilities and information. New York: Amacon.

Landwehr, C. E. (1981). Formal model of computer security. *Computing Surveys*, *13*(3), 247-278.

Meadows, C. (1997).

Morales, A. W. (2000, April). Intrusion detection. *Software Development*. Retrieved May 28, 2002 from http://www.sdmagazine.com/print/documentID=11202.

Nelson, R. (1997). Integration formalism and pragmatism: architectural security. New Security Paradigms Workshop, Langdale, Cumbria, UK.

Oppliger, R. (1997). Internet security: Firewalls and beyond. *Communications of the ACM*, *40*(5), 92-102.

Saltzer, J. H., & Shroeder, M. D. (1975, September). The protection of information in computer systems. *Proceedings of the IEEE, 63*, (9).

Schneider, F. B. (2000). Enforceable security policies. *ACM Transactions on Information and System Security*, *3*(1), 30-50.

Serban, C. & McMillin, B. (1996). Run-time security evaluation: Can we afford it? ACM – New Security Paradigms Workshop, Lake Arrowhead, CA.

Ting, T. C. (1993). Modeling security requirements for applications. *Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications* (p. 305). Washington United States.

Webster, J., & Watson, R. T. (2002, June). Analyzing the past to prepare for the future: writing a literature review. *MIS Quarterly*, *26*(2), xii-xxii.

QUESTIONNAIRE

- The extent to which your application uses external programs to provide enhanced functionality?
- The extent to which your application uses executable content (such as Java and Active X)?
- The extent to which an independent security protection audit is implemented before release?
- The extent to which mechanisms are in place to ensure that your users are using the correct version of your application? (Cohen, 1999).

Chapter XII

Learning Systems and Their Engineering: A Project Proposal

Valentina Plekhanova
University of Sunderland, UK

ABSTRACT

This chapter presents a project proposal that defines future work in engineering the learning processes in cognitive systems. This proposal outlines a number of directions in the fields of systems engineering, machine learning, knowledge engineering and profile theory, that lead to the development of formal methods for the modeling and engineering of learning systems. This chapter describes a framework for formalization and engineering the cognitive processes, which is based on applications of computational methods. The proposed work studies cognitive processes in software development process and considers a cognitive system as a multi-agents system of human-cognitive agents. It is important to note that this framework can be applied to different types of learning systems, and there are various techniques from different theories (e.g., system theory, quantum theory, neural networks) can be used for the description of cognitive systems, which in turn can be represented by different types of cognitive agents.

BACKGROUND

It is recognized that most software development tasks are cognitively driven and the focus on people quality and their management may provide considerable software process improvement (Curtis, 1981; Kellner & Hansen, 1989; Kellner& Rombach, 1991; Sommerville & Rodden, 1996). However, most existing process models and conventional project management approaches do not consider cognitive processes (Plekhanova, 1999a) and human resource quality (Sommerville & Rodden, 1996). Instead, they over emphasize the technical components. For this reason, their practical application is restricted to those projects where human resources are not a critical variable. Formal representation and incorporation of cognitive processes (Plekhanova, 1999a) and human aspects in modeling frameworks is seen as very challenging for software engineering research (Kellner & Hansen, 1989; Kellner & Rombach, 1991; Rombach, 2001).

The proposed project brings together work in systems engineering, knowledge engineering and machine learning for modeling cognitive systems and cognitive processes. We consider engineering the cognitive processes as the application of mathematical techniques and engineering methods to cognitive processes. We believe that the establishment of engineering methods with a sound theoretical basis can lead to the improvement of cognitive processes in software projects. We also use a synthesis of formal methods and heuristic approaches to engineering tasks for the evaluation, comparison, analysis, evolution and improvement of processes.

In this work we consider human resources as a cognitive system. The aims of the project are to develop a formal method for the modeling and engineering of a cognitive system in order to support the required learning processes.

In order to define learning processes, we engineer cognitive processes via a study of knowledge capabilities of cognitive systems. We are not interested in chaotic activities and interactions between cognitive agents, nor interested in detailed tasks descriptions, detailed steps of performance of the tasks and internal pathways of thoughts. Rather, we are interested in how available knowledge/skills of cognitive agents satisfy required knowledge/skills for the performance of the cognitive tasks.

We address the problem of cognitive system formation with respect to the given cognitive tasks and consider the cognitive agent's capabilities and compatibilities factors as critical variables, because these factors have an impact on the formation of cognitive systems — the performance processes and different learning methods.

We recognize that different initial knowledge capabilities of the cognitive system define different performance and require different hybrid learning methods. We study how human-cognitive agents utilize their knowledge and skills for learning the cognitive tasks. Learning methods lead the cognitive agent to the solution of cognitive problems/tasks. We consider a learning method as a guider to the successful performance. That is, we correlate initial knowledge capabilities of human agents with learning methods that define cognitive processes. We analyze

impact of different cognitive processes on the performance (or behavior) of human agents.

We will provide support for a solution to resource-based problems in knowledge integration and scheduling of cognitive processes to form a capable cognitive system for learning the required tasks.

In the proposed project, we use the profile theory (Plekhanova, 1999a, 2000b) for formalization of cognitive systems and cognitive processes and for the identification of critical areas in software development where improvement should be taken (Plekhanova, 1999a, 2002b). In particular, we consider engineering the cognitive processes to provide improvement of software development by means of integrating adaptive machine learning into the profile theory. In order to model cognitive processes in software projects, we combine the profile theory, which is used for knowledge engineering (analysis, integration, scheduling) and machine learning methods, which are applied to the initial available knowledge capabilities of the cognitive system to define learning methods for the tasks. (It is expected that different initial knowledge capabilities of the cognitive system require different hybrid learning methods.)

Therefore, the adventure in our research is that cognitive processes will be incorporated into system development of agents by a synthesis of systems engineering with knowledge engineering and machine learning methods. The combination of adaptive machine learning methods with profile theory will provide a more flexible adaptive framework for software system development. That is, the proposed method for the modeling and engineering of cognitive systems and cognitive processes can be used in software/systems engineering and machine learning for a formalization of cognitive processes, cognitive systems, and capability and compatibility aspects.

AIMS AND OBJECTIVES

The aims of the project are to develop a formal method for the modeling and engineering of cognitive processes in software development. In order to support the formation of a cognitive system that will be capable of learning the required tasks within the given constrains, we consider the knowledge integration and scheduling for cognitive system modeling, taking into account critical capability and compatibility factors. In this work, we study conditions for learning in cognitive systems.

The proposed project is based on work in systems engineering, software engineering, knowledge engineering and machine learning for modeling of cognitive processes that define a unique integration of methods in these fields. In particular,

- Profile theory (Plekhanova & Offen, 1997; Plekhanova, 1999a, 2000b) will be used to formalize cognitive processes and tasks, and knowledge of human and/or nonhuman agents.
- Heuristics and hybrid machine learning will be used to model and adapt profiles of agents in a flexible manner.

- Knowledge integration and scheduling problems will be addressed in learning of cognitive systems.

In particular, we will study cognitive processes and knowledge capabilities of cognitive systems to ensure the required level of the learning and performance of the cognitive systems. We address the problem of the formation of cognitive systems with respect to the given tasks and consider the capabilities and compatibilities factors of cognitive agents as critical variables because these factors have an impact on the formation of cognitive systems — the performance processes and define different learning methods.

The individual measurable objectives are:

1. Evaluation of knowledge integration and scheduling approaches in cognitive systems;
2. Evaluation of existing machine-learning approaches in cognitive systems;
3. Determination of the impact of capability and compatibility factors on the formation of cognitive systems;
4. Development of knowledge integration metrics;
5. Development of knowledge integration models for the formation of the cognitive systems;
6. Development of scheduling models for learning of cognitive systems; and
7. Development of a software prototype for modeling the cognitive processes.

METHODOLOGY AND JUSTIFICATION

In order to identify the best learning processes, we analyze the cognitive processes. The scenario for engineering the cognitive processes is based on the following steps where we (Plekhanova, 1999a):

1. model learning tasks;
2. measure and analyze the agent's knowledge capabilities and compatibilities with respect to the tasks;
3. use an integration model to form a cognitive system which consists of knowledge-interdependent agents;
4. match learning tasks, capable (cognitive and noncognitive) agents and learning methods that can be used for the learning of cognitive tasks;
5. identify critical areas for improvement of the learning processes;
6. schedule/allocate resources (agents) and machine-learning methods to the specific tasks;
7. measure results of the learning processes with respect to the required artefact (e.g., software product);
8. repeat step 1, if new tasks and/or requirements are identified.

In the proposed project, we:

a. Use the profile theory for the formalization of the cognitive agents capabilities; measurement of their capabilities and compatibilities; measurement of knowledge integration in order to form a cognitive system. This system will be capable of learning the cognitive tasks (Plekhanova, 1999a).
b. Use the profile theory for the development of an approach to knowledge scheduling to ensure learning processes that satisfy performance and learning constraints (Plekhanova, 1999a);
c. Analyze and apply machine-learning methods (e.g., Boosting, Lazy Learning, Associative Networks, Incremental Decision Tree Learning, Support Vector Machine) with respect to the initial knowledge capabilities of the available cognitive system. In order to find the learning method suitable for a certain application, we need to identify not only the formal task [e.g., Reinforcement Learning (Sutton & Barto, 1998), Clustering] that covers the learning problem (as in traditional machine-learning approaches), but also we need to define the initial capabilities of the cognitive agents and the system. (Note that different initial knowledge capabilities require different hybrid learning methods.) That is, it is necessary to determine initial conditions for learning.

The methodology of the proposed project is based on the following new theoretical basis.

Profile Theory and Machine Learning

There are a number of real-world examples when an object factor cannot be described by just one characteristic. As opposed to a classical set, which consists of factors (elements or objects), a profile describes an object by a set of (homogeneous and/or heterogeneous) factors that are, in turn, defined by factor importance, time and internal characteristics in an object representation. A profile is considered a method for describing and registering multifaceted knowledge about homogeneous and heterogeneous objects. There are important practical applications of the profile theory. For instance, internal properties of the system elements, such as capability and compatibility factors, are critical variables in modeling, design, integration, development and management of most modern complex systems and their structure. As an example, let us consider capability and compatibility problems of technical and soft systems.

In technical systems the internal characteristics of technical elements are described in specifications, standards and formal documents (i.e., are known a-priori) from which it is not difficult to conclude whether combinations of technical elements are capable and compatible or not, and whether they can be used for technical system design, development and construction. There is no emphasis on

capability and compatibility aspects in such system modeling and/or construction. Each capability and compatibility factor can be represented by one characteristic.

We consider such systems as human resources, software, information systems, cognitive agents, where the internal multifaceted properties can be changed with time, and capability and compatibility factors cannot be defined by one characteristic alone and cannot be explicitly defined and measured. These systems are termed soft systems. At the present time, there are problems in the formal definition of specifications, standards and metrics that allow one to determine the capability and compatibility of elements of such complex systems. For this reason, heuristic approaches are used for soft complex system modeling.

For formal modeling of cognitive agents/systems, we utilize the profile theory (Plekhanova, 1999a, 2000b). Internal factors of the cognitive agents are defined by multiple characteristics, such as weight of factor importance, time or factor existence/nonexistence and other specific internal multifaceted properties. Profile theory is used since existing mathematical theories are limited. Contemporary mathematical theories describe objects where each internal factor is represented by one meaningful piece (e.g., set theory — an element) or two pieces of meaningful information (e.g., fuzzy set theory — an element and a membership function).

Knowledge factors are considered as basic factors in the modeling of cognitive agents, since agents must have particular knowledge capabilities to perform and learn their tasks. In a description of the knowledge of cognitive agents, we identify the importance of the factor for the performance of the task, the property (level, grade, degree) and existence or nonexistence of the factor.

Knowledge of the cognitive agent is described by a set of knowledge factors; each factor is defined by multiple characteristics. A set of such factors forms a knowledge profile (Plekhanova & Offen 1997; Plekhanova, 1999a, 2000b). We represent a factor by qualitative and quantitative information. Quantitative description of the *i*th knowledge factor is defined by an indicator characteristic, property and weight. To give a simplified illustration in this proposal, we define a profile b as a set of factors $b_1, b_2, \ldots, b_n$: $b = \{b_i, i = \overline{1,n}\}$, where the ith factor b_i is represented by a pair = $(b_i = \iota_i, e_i)$ with:

- n — a number of factors;
- ι_i — an identification of the ith factor, i.e., a name or label or type of the ith factor;
- e_i — the 3-tuple of the ith factor as the Cartesian product: $e_i = <\varepsilon_i, v_i, w_i>$, where
- ε_i — indicator characteristic that indicates and expresses, by factor presence in the description of a cognitive agent, the existence of certain conditions. In particular,
  - ε_i may be defined as a time characteristic of the ith factor $\varepsilon_i = \varepsilon_i(t)$;
  - ε_i may also represent a number of times of factor utilization ;
  - ε_i may represent a binary case;

- v_i — property of the ith factor, $v_i \geq 0$. Since property may be changed with time, v_i can be defined as a function of time $v_i = v_i(t)$;
- w_i — weight of a factor, which defines either the factor importance or the factor priority, $w_i \geq 0$. Factor weights can vary, and, therefore, w_i can be also considered as a function of time $w_i = w_i(t)$.

Machine-learning methods are used for formalization and modeling of learning processes via applications of the profiles. It allows consideration of dynamics in learning processes (i.e., modeling of the ith profile factor $e_i(t) = <\varepsilon_i(t), v_i(t), w_i(t)>$ in the profile b). We will analyze existing machine-learning methods, match them to learning tasks with relevance to available knowledge capabilities of cognitive agents and consider cognitive processes. A profile is considered as a model for the description of cognitive system behavior (or cognitive processes). That is, a new machine-learning method will be developed and incorporated into an engineering framework for cognitive processes.

In this research we consider knowledge/skill factors as critical variables in learning processes and address problems in the formation of a cognitive system, which can be capable of learning. In particular, we study a complex system as a cooperation of knowledge-interrelated cognitive agents. We define a cognitive system by knowledge capabilities of cognitive agents, cognitive structure and cognitive processes. We address the problems of knowledge integration and analysis of knowledge capabilities of the cognitive system in order to provide a better opportunity for learning. A challenge for learning is to ensure the existence of a desired level of performance of a cognitive system. There is a need to make a formal analysis of the available knowledge of cognitive agents in order to ensure the learning of the tasks at a desired performance level, while utilizing the available knowledge capabilities effectively and efficiently (Plekhanova & Offen, 1997; Plekhanova, 1999a).

In a cognitive system, we address the problem of agent allocation, where we consider not only task scheduling as in traditional approaches but also scheduling machine-learning methods and knowledge of cognitive agents. That is we will develop a new scheduling approach where the agent-allocation problem has specific emphasis on the following aspects (Plekhanova, 1998b, 1998c): Cognitive agents are allocated to tasks according to their multiple knowledge capabilities; the agent's knowledge capabilities must satisfy the particular combination of knowledge required for a task; agents of the cognitive systems should be compatible with each other and learning methods are relevant to the available knowledge capabilities of the cognitive agents and system. The consideration of all these aspects defines a problem of knowledge integration in cognitive software systems. We will use methods for an integration of cognitive capabilities and compatibilities, and an analysis of how system capabilities satisfy the learning of the tasks.

Capability and compatibility factors have considerable impact on the process of system integration (Plekhanova, 1998b). We determine an integration model, which encompasses integration criteria (e.g., with respect to capability and compatibility and/or performance factors), priorities of the knowledge profiles and knowledge integration goals. Knowledge integration goals are the improvement of available knowledge or generation of new/novel knowledge for better performance and/or problem solutions. We may identify different requirements (e.g., integration criteria, priorities) for integration models that define a set of integration models. We will develop knowledge integration models using integration metrics and representing different integration criteria. The goal of knowledge integration is to define the satisfaction of available knowledge from the cognitive system to the required knowledge for the performance of the task (Plekhanova, 2000b, 2002a).

PROGRAM OF WORK

The following tasks will be undertaken within the proposed project in order to achieve the research objectives:

1. Evaluation of integration and scheduling problems in cognitive systems;
2. Analysis of contemporary knowledge integration and scheduling approaches in cognitive systems, i.e., human resources in software projects, planning methods and machine learning from artificial intelligence;
3. Analysis of existing machine-learning techniques [e.g., Schapire (1999), Aha (1997), Utgoff (1989) and Vapnik (1998)] and evaluation of learning problems in cognitive systems;
4. Investigation of the impact of the capability and compatibility factors on the formation of cognitive systems. This task will include: identification of critical factors and their interrelation; data collection on knowledge capability and compatibility of agents and condition analysis for learning in cognitive systems;
5. Development of knowledge integration metrics: Existing techniques (Plekhanova, 1999a, 2000b) will be extended to measure knowledge integration of cognitive agents. We will also develop new evaluation techniques for definition of agents (knowledge/skill, learning) capability and compatibility in order to provide support for effective solutions to resource integration;
6. Development of knowledge integration models for the formation of a cognitive system: The existing approach (Plekhanova, 2000b) will be extended by using new integration metrics. We will develop knowledge integration models for learning in a cognitive system representing different integration criteria. The goal of knowledge integration is to define how available knowledge from the cognitive system satisfies the required knowledge for the performance of the task.

7. Development of scheduling models for the learning of a cognitive system: The existing approach (Plekhanova, 1998c, 1999a, 2000d) will be extended by using new knowledge integration models;
8. Application of supervised and unsupervised learning methods to cognitive tasks;
9. Development of Demonstration System: We will develop a software prototype to provide support for a solution to resource-based problems in knowledge integration and scheduling of cognitive processes, which will incorporate a new formal approach (based on the profile theory) and machine-learning methods to the modeling of cognitive systems. This tool will provide the means for analysing human resource quality and its impact(s) on the system's performance.

NOVELTY

The proposed project is particularly novel in its approach to learning processes that incorporate a synthesis of systems engineering, knowledge engineering and machine-learning methods. There are no formal methods for knowledge integration and scheduling for learning of cognitive systems where capability and compatibility factors are critical variables. Existing machine-learning approaches do not address scheduling problems in learning methods. We will develop a *new* scheduling approach where we consider scheduling machine-learning methods and knowledge of cognitive agents vs. task scheduling in traditional approaches. A new machine-learning method will be developed and incorporated into an engineering framework for cognitive processes (see Methodology and Justification section). Moreover, the proposed project brings together work in cognitive systems, systems engineering, knowledge engineering and machine learning for the modeling of cognitive processes.

The proposed project is timely because of the availability of new formal methods for engineering cognitive systems. The work is highly topical at present as demonstrated by a large interest in academia and great needs of industry and academia (Rombach, 2001). In particular in:

Software/systems engineering: Project resource capability and compatibility aspects have become the focus of software process improvement activities. However, most contemporary approaches to the formation of project resources do not examine their capability and compatibility factors. There is a need to develop evaluation techniques for people's capability, resource capability and compatibility in order to provide support for effective solutions to project resource integration management in cognitive systems. In particular, methods of particular merit are those that incorporate a comparison of cognitive processes, resource capabilities and compatibilities, and an analysis of how resources fit the project need.

Scheduling: Contemporary approaches to resource scheduling (Plekhanova, 1998b) are based on the detailed description of tasks assuming that a resource pool is given and defined by a manager, and resources are capable of performing any project task. Existing resource scheduling methods, both heuristic and optimization, address the issues of resource availability and utilization and are not concerned with the capability and compatibility of project resources. Furthermore, in traditional scheduling approaches, the objectives for the allocation of limited resources are to determine the allocation of resources that maximize total benefits subject to the limited resource availability (Plekhanova, 1998b). Contemporary approaches to resource allocation are founded on the assumption that different tasks require equal capability resources, and only one skill is involved. Hence, they cannot be successfully used for software projects where different software tasks require changing different sets of multiple knowledge and skill capabilities in an overall system (Plekhanova, 1998b).

Software tools for resource scheduling: There are many scheduling tools that provide different approaches: event-oriented (PERT), activity-oriented (CPM), actions-oriented (TASKey PERSONAL) or offer a wide variety of scheduling options (SAP). Nevertheless, there are no tools that support an analysis of people (knowledge/skill, learning) capabilities, resource compatibilities and their impact on project scheduling (Plekhanova, 1998c, 2000c). Most existing tools (Microsoft Project, SAP, Up and Running) have facilities for entering new resources, but do not deal with an analysis of cognitive processes and resource quality based on which resources can be added to the resource pool. Therefore, the existing scheduling tools cannot be effectively used for management of software development processes where human resources are a critical variable.

Theory/tools in machine learning: Existing machine-learning methods [e.g., Boosting (Schapire, 1999), Lazy Learning (Aha, 1997), Neural Nets, Incremental Decision Tree Learning (Utgoff, 1989), Support Vector Machine (Vapnik, 1998)] and contemporary machine-learning tools (e.g., WEKA, AutoClass, mySVM) have not examined an agent's capability/compatibility and scheduling problems.

There is a direct relationship between the representation and the learning mechanisms. In many cases the underlying representations in machine learning have been of limited structure (e.g., vectors, trees). A hybrid integration of various machine-learning mechanisms for software engineering of structured objects is novel and will be examined in this project in the context of the profile theory.

RELEVANCE TO BENEFICIARIES

Engineering the Complex Systems: Research in engineering of complex systems will provide insight into new methods and approaches to learning in cognitive systems. Research in machine learning will deliver adaptiveness to knowledge integration and scheduling of learning methods. Scientists in cognitive systems research will receive a formal method for modeling cognitive processes. By developing (knowledge) integration metrics, using the profile theory, we can provide analysis, development, integration, modeling and management of complex systems and their elements where weight, time and other internal multifaceted properties are critical variables. Further development of the profile theory will establish a new branch in mathematics and extend its applications.

Another benefit would be the training of an expert in the combined fields of systems engineering, software engineering, knowledge engineering and machine learning at the end of the project.

Industry: New evaluation techniques could provide support for a solution to the resource-based problems in cognitive processes in software and IT projects, such as team formation and integration in connection with process tasks.

The application of a new approach could provide learning IT and software development organisations with:

- superior management of resource capabilities and compatibilities;
- streamlining of process development through better management of project resources and tasks;
- increased opportunities for organizations to implement process improvement based on the constructive criticism derived from self analysis.

It is apparent that there is a worldwide interest in the application of this research. Since most modern processes are cognitively driven, our method can be used for the formal modeling of cognitive systems. It is important for the future competitiveness of the software and IT industry to employ a scientific (vs. heuristic) approach to the engineering of cognitive processes.

Technology: The capability/compatibility-based approach assures a virtual prototyping of system development within different environment settings. An important application of this approach is that it gives the means of providing systemic methods of study, analysis, prediction, improvement, control and management of a system development. Moreover, this technology demonstrates a modeling flexibility that permits one to represent a fine granularity of system components, as well as to generate different system models of a wide diversity of system development

processes. Thus, any traditional system model becomes a special case of the capability- and compatibility-based modeling framework.

Formal modeling of the capability and compatibility of cognitive systems ensures the automation in cognitive system modeling. It leads to development of new technologies in system modeling. Some of the enhancements that we intend to offer through this method are to provide support for development and engineering of new knowledge capabilities of cognitive systems, i.e., innovative technologies.

ACKNOWLEDGMENT

I would like to thank my colleagues from the University of Sunderland for their valuable comments on this proposal.

REFERENCES

Abdel-Hamid, T., & Madnick, S.E. (1991). *Software project dynamics: An integrated approach.* Prentice-Hall.

Aha, D. (Ed.). (1997). *Lazy learning.* Dordrecht: Kluwer Academic Publisher.

Bergadano, F., & Gunetti, D. (1995). *Inductive logic programming: From machine learning to software engineering (logic programming).* MIT Press.

Curtis, B. (Ed.). (1981). *Human factors in software development*. Los Angeles, CA: IEEE Computer Society.

Kellner, M. I., & Hansen, G. A. (1989). Software process modeling: A case study. *Proceedings of the 22nd Annual Hawaii International Conference on System Sciences, II,* (pp. 175-188).

Kellner, M. I., & Rombach, H. D. (1991). Session summary: Comparisons of software process descriptions. *Proceedings of the 6th International Software Process Workshop, IEEE Computer Society, Washington,* (pp. 7-18).

Plekhanova, V. (1998a). A formal approach to the integration of CASE tools. *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics (SCI'98) and the 4th International Conference on Information Systems Analysis and Synthesis (ISAS'98),* (pp. 371-373) *Orlando, Florida, USA, 1*.

Plekhanova, V. (1998b). On project management scheduling where human resource is a critical variable. *Proceedings of the Sixth European Workshop on Software Process Technology (EWSPT-6),* (Lecture Notes in computer science series, pp. 116-121). London, UK: Springer-Verlag.

Plekhanova, V. (1998c). A framework for people-centred approach to software process constraints scheduling. *Proceedings of the International Workshop*

on Software Process Simulation and Modelling (ProSim'98), Portland, USA.

Plekhanova, V. (1999a). *A capability- and compatibility-based approach to software process modelling*. Unpublished doctoral thesis, Macquarie University, Sydney, Australia, and the Institute of Information Technologies and Applied Mathematics, Russian Academy of Sciences.

Plekhanova, V. (1999b). Capability and compatibility measurement in software process improvement. *Proceedings of the 2nd European Software Measurement Conference - FESMA'99* (pp. 179-188) Technological Institute Publications, Antwerp, Belgium, Amsterdam, the Netherlands.

Plekhanova, V. (2000a). Profile theory and its applications. *Proceedings of the international conference on information society on the 21st century: Emerging technologies and new challenges (IS2000)* (pp. 237-240). The University of Aizu, Fukushima, Japan.

Plekhanova, V. (2000b). Applications of the profile theory to software engineering and knowledge engineering. *Proceedings of the 12th International Conference on Software Engineering and Knowledge Engineering (SEKE'2000),* (pp. 133-141). Chicago.

Plekhanova, V. (2000c). On the compatibility of contemporary project management tools with software project management. *Proceedings of the 4th World Multiconference on Systemics, Cybernetics and Informatics (SCI'2000) and the 6th International Conference on Information Systems Analysis and Synthesis (ISAS'2000),* (pp. 71-76) *Orlando, FL, USA, I.*

Plekhanova, V. (2000d). *Towards modelling the scheduling processes for software projects.* Paper presented at the 2nd International Workshop on Planning, Scheduling, and Control in Manufacturing: Putting the Human back in Control, Zurich, Switzerland.

Plekhanova, V. (2002a). Concurrent engineering: Cognitive systems and knowledge integration. *Proceedings of the 9th European Concurrent Engineering Conference, Modena, Italy* (pp. 26-31). SCS Europe (Society for Computer Simulation).

Plekhanova, V. (2002b). Engineering the cognitive processes in software projects: Machine learning, knowledge integration and scheduling. *Proceedings of 2002 Information Resources Management Association International Conference: Issues and Trends of Information Technology Management in Contemporary Organizations, Seattle, WA, USA,* Volume 2, 1076-1077.

Plekhanova, V., & Offen, R. (1997). Managing the human-software environment. *Proceedings of the 8th International Workshop on Software Technology and Engineering Practice (STEP'97)* (pp. 422-432). London, UK: IEEE Computer Society Press.

Rombach, D. (2001). *Experimental software engineering: Building a research community.* Australian Software Engineering Conference (ASWEC), Australia.

Rombach, H. D., & Verlage, M. (1995). Directions in software process research. By M. V. Zelkowitz, *Advantages in computers* (Vol. 41, pp. 1-62) Boston, MA: Academic Press, Inc.

Schapire, R. (1999). Theoretical views of boosting and applications. In O. Watanabe & T. Yokomori (Eds.), *Proceedings of the 10th International Conference on Algorithmic Learning Theory,* (pp. 13-25).

Sommerville, I., & Rodden, T. (1996). Human, social and organisational influences on the software processes. In A. Fuggetta & A. Wolf (Eds.), *Software Process, Vol. 4 of Trends in Software* (pp. 89-100) New York: J. Wiley.

Sutton, R. S., & Barto, A. G. (1998). Reinforcement learning: An introduction. Cambridge, MA: MIT Press.

Turner, M., & Austin, J. (1998). Graph matching by neural relaxation. *Neural Computing and Applications, 7*, 238-248.

Utgoff, P. E. (1989). Incremental induction of decision trees. *Machine Learning*, *4,* 161-186.

Vapnik, V.N. (1998). *Vladimir, statistical learning theory*. Chichester, UK: Wiley.

Wooldridge, M., & Jennings, N. (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review, 10*(2), 115-152.

Young, J., Lees, K., & Austin, J. (1999). Performance comparison of correlation matrix memory implementations. *Proceedings of 6th International Conference on Neural Information Processing.*

Chapter XIII

Towards Construction of Business Components: An Approach to Development of Web-Based Application Systems

Dentcho N. Batanov
Asian Institute of Technology, Thailand

Somjit Arch-int
Khon Kaen University, Thailand

ABSTRACT

Global competition among today's enterprises forces their business processes to evolve constantly, leading to changes in corresponding Web-based application systems. Most existing approaches that extend the traditional software engineering to develop Web-based application systems are based on object-oriented methods. Such methods emphasize modeling individual object behaviors instead of system behavior. This chapter proposes the Business Process-Based Methodology (BPBM) for developing such systems. It uses a business process as a unified conceptual framework for analyzing relationships

between a business process and associated business objects and for identifying business activities and designing object-oriented components called business components. We propose measures for coupling and cohesion measurement in order to ensure that these business components enable the potential reusability. These business components can more clearly represent semantic system behaviors than linkages of individual object behaviors. A change made to one business process impacts some encapsulated atomic components within the respective business component without affecting other parts of the system. A business component is divided into parts suitable for implementation of multitier Web-based application systems.

INTRODUCTION

The increasing competition caused by worldwide businesses forces the enterprise's strategies to evolve frequently. Whenever a strategy has been changed, the associated business processes must also be remodeled which in turn requires that the corresponding Web-based application systems also be re-implemented and installed quickly.

Web applications are software-intensive systems based on the typical three-tier Web application architecture, which should be centered on not only presentation modeling, but also business logic and business-state modeling (Conallen, 1999). Further, Frolund and Guerraoui (2002) described that "a typical application, distributed or not, usually includes elements that handle presentation, logic, and data" (p. 378). With this knowledge, a typical application can be modeled based on the well-known Model-View-Controller (MVC) architecture (Grasner & Pope, 1988). That is, centralized systems, client/server systems and multitier distributed systems including Web-based application systems can be modeled with respect to the MVC architecture, which make the presentation (i.e., View) component independent of the other components. Based on this aspect, a Web-based application system can be modeled into three component types corresponding to the Model, View and Controller components. Such components are modeling regardless of presentation components (e.g., browsers for Web-based systems or Windows for client/server or distributed systems), communication protocol (e.g., stateless (HTTP) for Web-based systems or stateful for the other) and data source types (e.g., relational/object-oriented database or XML document). In software-intensive Web-computing environment, the most important issue is how to model the primary element (i.e., Model) of a Web-based application system to be a rigorous and flexible enough in order to be adaptable according to such dynamic and global businesses. Moreover, the elements of the Model component should also be a seamless transformation to other components. Based on the notion of a Web-based system is a software-intensive system.

The expanding traditional software engineering is an alternative solution in modeling Web-based application systems.

Most existing approaches that extend the traditional software engineering to develop Web-based application systems are based on object-oriented methods. Although the object orientation provides powerful mechanisms, such as encapsulation, inheritance and reusability (Booch, 1994; Jacobson et al., 1995), the use of objects as building blocks in the early phase of the development result in individual object behaviors instead of system behavior. Further, since Web-based application systems emphasize both business logic and presentation, the traditional software-implementation model does not fit to the Web-implementation model (Puc-Rio, & Rossi, 2000). Therefore such existing approaches (Chen & Heath, 2001; Huang & Mak, 2001) need further enhancement in order to master dynamic and sophisticated systems.

BPBM blends advantages of the structured (Agarwal, De & Sinha, 1999) and object-oriented paradigms (Booch, 1994; Jacobson, Ericsson & Jacobson, 1994; Jacobson et al., 1995; Jacobson, Griss & Jonsson, 1997) for identifying and designing business components based on the notion that a business process consists of a function-oriented part (activities) representing object behavior and a process-oriented part acting as a collaboration of these activities (Snoeck-S & Dedene, 2000). The benefits of the proposed business component model are not only taking advantage of the structural approach, which itself meets naturally related functional requirements, but the model also conforms to the powerful MVC architecture in its implementation in a Web-based environment.

The remainder of this chapter is organized as follows. Application systems on the Web are reviewed and basic definitions are described in the next section. The proposed approach for business components and Web-implementation modeling are described. Finally, conclusions are outlined.

APPLICATION SYSTEMS ON THE WEB

Growth of Internet technology has allowed modern enterprise to move business systems onto the Internet (Kocharekar, 2001). That environment requires electronic data interchange (EDI) between enterprises to be a prime feature for providing a variety of Internet applications, e.g., E-commerce (B2B or B2C), including Web-based application systems. Making the move to the EDI Web environment raises the need to address two additional requirements, heterogeneous data sources and universal client accessibility:

1. Heterogeneous data sources used within one or across several distinct application platforms prevent interoperability. To gain the ability to provide data exchanges between enterprises, including decoupling the accessing of data from different tiers, such systems should have a mediator as a handling interface.
2. Clients should not be required to have high performance machines. Moreover, a client's presentation method should support a variety of output formats (e.g.,

HTML, PDF, etc.) and support various client devices (e.g., Personal Digital Assistants (PDAs), wireless phones, etc.).

An appropriate solution for these challenges is the use of Extensible Markup Language (XML)-based technologies (W3C-XML, 2000). XML is a "metalanguage" that provides metadata for describing other data in a document. An XML document uses semantics tags that are validated with their XML Document Type Definition (XML DTD) to describe the document's content, which enables communication across platforms. Based on such semantic tags, XML allows developers to design their own customized markup languages for limitless different types of documents.

An XML document acts as a middle-tier distributed object to interface multitier Web application systems components through the HTTP protocol. An XML document produced by a variety of proprietary database vendors can be distributed either to other distinct application platforms for EDI-enabled applications or to a requesting client. In addition to the second challenge, XML documents combined with a variety of XSL stylesheets (W3C-XSL, 2000) can also support various client devices and requirements.

BASIC DEFINITIONS

Business Objects

A business object (BO) is an object with well-defined boundaries and an identity that encapsulates a business state and behavior. A business state is a structural property represented by attributes or instance variables, while a behavior is a behavioral property represented by methods that operate on the attributes. More details on the formal definition of this notion by the Object Management Group can be found in (OMG, 1999).

Let *BO* be a business object composed of *m* attributes, $A=\{a_1, a_2, ..., a_m\}$, and *n* methods, $M=\{M_1, M_2, ..., M_n\}$. In terms of overall behaviors, *BO* can be defined as:

$$BO \equiv \{M_1(A_1), M_2(A_2), \ldots, M_n(A_n)\},$$

where $A_i \subseteq A$, for all $i = 1, 2, ..., n$. M_i operates on a set of attributes A_i.

Business Components

A business component (BC) defined by Herzum and Sims (2000) is a software unit that implements a business concept. Such a business component is large grained, which means it consists of all software artifacts necessary to represent, implement and deploy a given business concept as an autonomous, reusable element of a larger distributed application system. A business component is a kind of component that has

some basic characteristics, e.g., self-contained software construct and well-defined and well-known run-time interface (Brown & Wallnau, 1998).

Our approach in developing business components focuses on identifying business activities instead of on business objects as in, for example, component-based software development (CBSD) (Herzum & Sims, 2000) and Business Object Component Architecture (BOCA) (Digre, 1998). The associated atomic components with respective activities are then composed to build a larger component — business component.

CONSTRUCTION OF BUSINESS COMPONENTS

Business Process-Based Methodology: An Approach for Business Components Modeling

The key of the BPBM approach in modeling business components is the use of business process as a unified conceptual framework for analyzing relationships between a business process and associated business objects, for identifying business activities and for designing business components. A business system is decomposed into a set of business processes and each business process is then decomposed into business activities resulting in a two-level hierarchy of the business processes model.

A business process is performed by participation/co-operation of a number of business resources called business objects, which can be either activator or business state. Activators represent actors who initiate a business process when a business process event occurs. Business states (business data or data objects) are created and/or consumed by a business process while performing the process. Each business activity of a particular business process requires business objects as generations of input and output data, and, therefore, it can be represented as a set of interactions between business objects.

A business activity is composed of interactions among related business objects. More specifically, a business activity may require a business object to interact with one or more business objects. In Figure 1, for example, BO_1 interacts with BO_2 and BO_3 in a particular sequence. BO_1 may interact with both business objects by using a respective method. It is more convenient to separate such interactions into two different pairs of interactions, $I_1(BO_1, BO_2)$ and $I_2(BO_1, BO_3)$, as shown in Figure 1(b).

Figure 1. Details of Interactions of a Business Activity

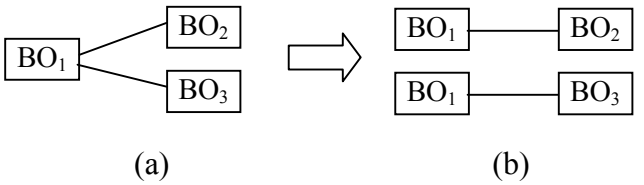
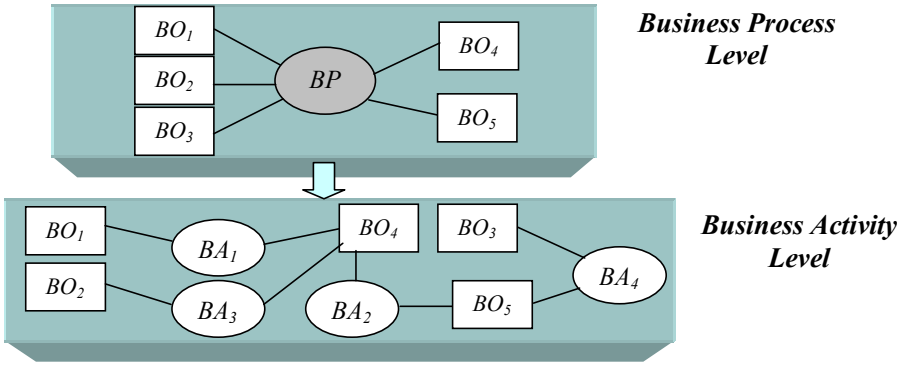


Figure 2. Two-Level Interaction Model of a Business Process



Let a business process, *BP*, be decomposed into four business activities BA_i ($i = 1, 2, \ldots, 4$), involving five BO_i ($i = 1, 2, \ldots, 5$), in different ways (Figure 2 with omitted explicit operation labels).

Using the example in Figure 2, BA_1 and BA_2 can be defined as follows:

$$BA_1 \equiv \{ \mathrm{I}(BO_1, BO_4) \} \text{ and } BA_2 \equiv \{ \mathrm{I}(BO_4, BO_5) \}$$

In general, a business activity consists of a set of interactions between business objects as formulated below:

$$\mathrm{BA} \equiv \{\mathrm{I}_1(BO_1, BO_2), \mathrm{I}_2(BO_3, BO_4), \ldots, \mathrm{I}_n(BO_p, BO_q) \},$$

where $\mathrm{I}_k(BO_i, BO_j)$ is a given interaction representing a public method invocation of business objects BO_i and BO_j. A more formal and precise definition of the interaction is defined as follows.

Postulate 1: Interaction

Let $m_i \in BO_1$ and $m_k \in BO_2$ be two methods of different business objects, bo_1 and bo_2. An *interaction* between the two business objects, denoted $I(BO_1, BO_2)$ if $\exists m_i \in BO_1, m_k \in BO_2$) such that ($m_i$ invokes m_k) v (m_k invokes m_i).

In high-level design, conceptual dependencies between business objects due to accessing operations can be determined using static analysis. We classify the degree of operations into three levels, *Creation(C) and Deletion(D)* (instance level), *Update(U)* (attribute level) and *Retrieve(R)* (read-only operation). Given these three kinds of operations, there are six possible interaction patterns between business objects: *C-C*[1], *C-U* or *U-C*, *C-R* or *R-C*, *U-U*, *U-R* or *R-U* and *R-R*. For example, the R-C interaction pattern requires one business object to retrieve business state and another business object to create a new instance. In addition, such an interaction

includes the case that a business object operates itself without interacting with others. This kind of interaction occurs due to the business activity, which needs to manipulate the business state of individual business objects.

The six interaction patterns are classified into three groups depending on the direction of interaction:

1. Two directions: This group consists of *C-C*, *C-U* and *U-U* interaction patterns and requires both associated business objects to create instances or to update the business state. Business objects involving in such interactions are masters.
2. One direction: This group consists of *R-C* or *R-U* interaction pattern. Such an interaction requires only one business object, which is the master while the other is the slave. The master business object, in this case, creates instances or updates business state.
3. No direction: This group consists of *R-R* interaction pattern. Such an interaction requires both associated business objects to be slaves.

A business activity consists of interactions among associated business objects. The result of one interaction can be an input for other interactions. Such a result can be either a new instance or updated business state of business objects, due to *C* or *U* operation, respectively, or singly data, due to *R* operation. Hence, an interaction between two business objects is represented as bidirectional regardless of its belonging to any direction group of interactions.

Analysis Model Representation

To make the analysis model simple and precise in identifying business activities and business components, using coupling and cohesion measures, the interactions between business objects are represented using an undirected graph (Hitz & Montazeri, 1996; Kenneth, 1999) called a business object interaction graph, which is similar to the input/output dependence graph (Bieman & Kang, 1998).

Definition 1: Business Object Interaction Graph (BOIG). Let $OP=\{C, U, R\}$ be the set of operations and $IP=\{C\text{-}C, C\text{-}U^2, C\text{-}R^3, U\text{-}U, U\text{-}R^4, R\text{-}R\}$ the set of interaction patterns between two particular business objects.

BOIG of a business activity *BA* is an undirected graph $G_{BA}(V, E)$, where V is the set of associated business objects, and E is the set of edges connected (interacted) between vertices (business objects) such that $E = \{<x, y> \in VxV \mid \exists i \in IP$: (x and y interaction through *i* operation)$\}$.

The loop graph (Kenneth, 1999) is used to represent interactions that involve only one business object such that $E = \{<x> \in V \mid \exists op \in OP$: (x operates on its business state with *op* operation)$\}$.

In Figure 2, there are four business activities, each one of which can be represented using BOIGs, for example, BA_1 and BA_2:

- G_{BA1}(V, E) represents the business activity BA_1, where V = {BO_1, BO_4} and E = {<BO_1, BO_4>}.
- G_{BA2}(V, E) represents the business activity BA_2, where V = {BO_4, BO_5} and E = {<BO_4, BO_5>}.

Every business activity consists of at least one interaction. Further, a business activity should not be isolated and should have at least one interaction with the outside world. Hence, |E| is greater than zero.

Business Activity Identification and Measurement

An associated business object must be identified explicitly by the business activities to which it should belong. An identified business activity is viewed as a modular unit with respect to an atomic software component. To be considered as a component, the functionality of a respective business activity should be in correspondence with requirements for high cohesion within the component and low coupling with the rest of components of the system (Chidamber & Kemerer, 1994).

Definition 2: Coupling between Business Objects-Business Object Interaction (CBO-BOI) is the number of interactions between two business objects, which operate on their business state with various kinds of operations.

Let a conceptual *BA* involve *n* business objects, $BO_{BA}=\{BO_1, BO_2, \ldots, BO_n\}$. For BO_i and $BO_j \in BO_{BA}$, it will *belong* to the *BA* if CBO-BOI(*BA*) = $|E_1| + |E_J| > 0$ such that

1. $E_1 = \{<BO_i, BO_j> \in V \times V \mid (\exists op \in OP)$: ($BO_i$ operates on its own business state with *C* operation) ^ (BO_j operates on its own business state with *op* operation)} and
2. $E_J = \{<BO_i> \in V$: (BO_i operates on its own business state with *C* operation)}.

More specifically, the business objects with a high degree of interactions should be encapsulated in a module to reduce an interbusiness object coupling. If CBO-BOI(*BA*) = *0*, there is no business object belonging to the *BA*. This *BA* is called an independent business activity. On the other hand, a business activity that requires business objects to operate on business states with the *C* operation is the main business activity of a particular business process. These business objects, therefore, are represented as the core business objects of the business process.

Definition 3: Coupling between Business Activity-Interaction-based (CBA-I) of an identified business activity is the proportion of the number of internal interactions (II) and the number of external interactions (EI).

Let an identified *BA* consist of *n* member business objects, $BO_{BA}=\{BO_1, BO_2, \ldots, BO_n\}$. Given a $BO_i \in BO_{BA}$,

1. BO_i interacts with other business objects belonging to the same *BA* (II) if there exist edges E_{II} ($|E_{II}| > 0$) such that $E_{II} = \{<BO_i, BO_j> \in V \times V \mid (\exists BO_j \in BO_{BA})(\exists i \in IP)$: ($BO_i$ and BO_j interact through an *i* interaction)$\}$. The greater $|E_{II}|$ implies the higher quality of the internal relationships in the *BA*.
2. BO_i interacts with other business objects, which belong to different business activities if there exist edges E_{EI} ($|E_{EI}| > 0$) such that $E_{EI} = \{<BO_i, BO_j> \in V \times V \mid (\exists BO_j \notin BO_{BA})(\exists i \in IP)$: ($BO_i$ and BO_j interact through an *i* interaction)$\}$. The more $|E_{EI}|$ implies the higher coupling of the *BA* with other business activities.

The CBA-I(*BA*) value is calculated in interval [0, 1]:

$$\text{CBA-I}(BA) = \frac{|E_{EI}|}{|E_{EI}| + |E_{II}|}$$

The higher the CBA-I(*BA*) value, the higher coupling of the *BA*. The numerical value *n* in interval [0, 1] is the critical point specified for determining the low or high coupling. The exact value is left for developers, who model the business components, which are flexible, varying from business system to business system.

For example, if the critical point is set to 0.5, the business activity *BA* regarding the CBA-I metric is expected to gain the number of internal interactions more than or equal to the number of external interactions. Hence,

- If CBA-I(*BA*) > 0.5, there exist more external than internal interactions. This implies that the *BA* should be split into two or more business activities. A technique for breaking up is to split the functionality and their responsive business objects.
- In contrast, if $|E_{EI}| < |E_{II}|$ or the CBA-I(*BA*) is close to *0.0*, there is a large number of internal interactions that need not be split up.

Definition 4: Lack of Cohesion of Business Activity (LCBA). In theory, for any identified business activity, which has been constructed as an independent component, the number of internal method invocations should represent the degree of cohesion. In practice, however, these interactions may enable business object "clusters," operating on disjointed sets of business objects.

The LCBA measure is a refinement of the notion of the Lack of Cohesion in Methods (LCOM) (Chidamber & Kemerer, 1994) to determine the number of business-object clusters. Let *BA* denote an identified business activity and BO_{BA} the set of member business objects. Let $G_{BA}(V, E)$ be an undirected graph with $V = BO_{BA}$.

To obtain an inverse measure of cohesion, LCBA(BA) = $|E_c|$ is defined as the number of cohesion clusters of business objects within the same BA so that:

$$E_c = \{<x, y> \in VxV \mid \forall z \in BO_{BA} : \neg((\exists <x, z> \in VxV) \vee (\exists <y, z> \in VxV))\}.$$

If LCBA(BA)>n, then this implies that BA should be split into n sub-business activities, each containing a cluster of business objects and their responsive functionality. On the other hand, if LCBA(BA)=*0*, then BA does not need to be split.

In summary, an identified BA should satisfy both criteria so that CBA-I(BA)<0.5 and LCBA(BA)=*0*.

Business Component Identification and Measurement

A business component is composed of identified business activities that satisfy both coupling and cohesion measurements. The first step of the technique is to combine a set of identified business activities with high coupling into a single business component with the objective of reducing coupling between identified business components. Then, the identified business component is evaluated for possible splitting into more business components based on the cohesion measurement.

Definition 5: Coupling between Business Activity-Business Object Interaction (CBA-BOI) is the number of interactions between business objects of the two business activities.

Let BO_{BA1} and BO_{BA2} be the set of business objects belonging to two identified business activities BA_1 and BA_2, respectively, and $BO_i \in BO_{BA1}$ and $BO_j \in BO_{BA2}$. There are two kinds of interactions between two business activities, direct and indirect interactions.

1. There is a direct interaction between BA_1 and BA_2, if there exists E_D such that $E_D = \{<BO_i, BO_j> \in VxV \mid \exists k \in IP$: ($BO_i$ and BO_j interact through a k interaction)$\}$.
2. There is an indirect interaction between BA_1 and BA_2, if there exists E_{ID} such that $E_{ID} = \{<BO_i, BO_j> \in VxV \mid (\exists BO_p \in BO_{BAX})$ such that $|<BO_i, BO_p>| > 0$ and $|<BO_p, BO_j>| > 0\}$. That is, there is a business object BO_p, which enables two interactions BO_i and BO_p, and BO_p and BO_j.

The CBA-BOI measure is used to identify business components as follows. Let IP_{CC}={C-C} be the highest coupling degree of interaction patterns. Two identified business activities, BA_1 and BA_2, should be encapsulated as an identified business component, if CBA-BOI(BA_1, BA_2) > 0, such that:

$$\text{CBA-BOI}(BA_1, BA_2) = |E_D| + |E_{ID}|$$

where $|E_D| > 0$, if $\exists k \in IP_{CC}$ and $|E_{ID}| > 0$, if $\exists l, m \in IP_{CC}$.

From the underlying object-oriented perspective, we can define coupling and cohesion measurements of business components as they are defined for business activities.

Definition 6: Coupling between Business Component-Interaction-based (CBC-I) of an identified business component (BC) is the proportion of the number of II and the number of EI.

By using the same criterion as CBA-I(*BA*), we can define CBC-I(*BC*), a larger scope, to measure the coupling between business components for creating independent business components.

Let an identified *BC* consist of *n* member business objects, $BO_{BC}=\{BO_1, BO_2, \ldots, BO_n\}$. Given a $BO_i \in BO_{BC}$,

1. BO_i interacts with other business objects belonging to the same *BC* (II), if there exist edges E_{II} ($|E_{II}| > 0$), such that
   $E_{II} = \{<BO_i, BO_j> \in VxV \mid (\exists i \in IP)$: ($BO_i$ and BO_j interact through an *i* interaction)$\}$.
   The more $|E_{II}|$ implies the higher quality of the internal relationships in the *BC*.
2. BO_i interacts with other business objects, which belong to different business activities, if there exist edges E_{EI} ($|E_{EI}| > 0$), such that
   $E_{EI} = \{<BO_i, BO_j> \in VxV \mid (\exists BO_j \notin BO_{BC})(\exists i \in IP)$: ($BO_i$ and BO_j interact through an *i* interaction)$\}$.
   The more $|E_{EI}|$ implies the higher coupling of the *BC* with other business activities.

The CBC-I(*BC*) value is calculated in interval [0, 1]:

$$\text{CBC-I}(BC) = \frac{|E_{EI}|}{|E_{EI}| + |E_{II}|}$$

The higher CBC-I(*BC*) value, then the higher coupling of the *BC*. The interpretation of an arbitrary critical point value is describable as expressed in the Metric 2 (CBA-I). For example, if the critical point is 0.5, the interpretation is described as follows:

- If CBC-I(*BC*) > 0.5, then there exist more external than internal interactions. This implies that the *BC* should be split into two or more business activities. A technique for breaking up is to split the functionality and their responsive business objects, which enables the maximum external relationships to be a new business activity.
- In contrast, if $|E_{EI}| < |E_{II}|$ or the CBC-I(*BC*) is close to *0.0*, then there is a large number of internal interactions, which need not be split.

Since business components can be assembled into a larger grained component, the properties of an ideal business component are low coupling and high cohesion, using the previous defined measurements. More specifically, a required business component should satisfy both criteria so that CBC-I(*BC*)<0.5 and LCBC(*BC*)=*0.*

Definition 7: Lack of Cohesion of Business Component (LCBC). In addition to the LCBC(*BC*), it can also be defined by extending scope of the LCBA(*BA*) for measuring lack of cohesion in business components.

Let *BC* denote an identified business component, BO_{BC}, the set of member business objects. Let $G_{BC}(V, E)$ be an undirected graph with $V = BO_{BC}$.

To obtain an inverse measure of cohesion, LCBC(*BC*) = $|E_C|$ is defined as the number of cohesion clusters of business objects within the same *BC,* such that:

$$E_C = \{<x, y> \in VxV \mid \forall z \in BO_{BC}: \neg((\exists <x, z> \in VxV) \vee (\exists <y, z> \in VxV))\}.$$

If LCBC(*BC*)>*n*, then *BC* should be split into *n* new business components, each containing a cluster of business objects and their responsive functionality. On the other hand, if LCBC(*BC*)=*0*, then *BC* does not need to be split.

Regarding business component-level metrics, an expected business component should satisfy both criteria so that CBC-I(*BC*)<*n*, where *n* is a critical point value and LCBC(*BC*)=*0.*

Business Component Services

Another consideration of modeling is the architecture. Layering the architecture is a design technique used to classify business component functions that allows each layer to do specific tasks and provides services for upper layers.

Business components are classified based on the notion of service-based measurement (Briand & Morasca, 1999) into two layers: common business component (CBC) layer, which provides general purposes services, and application business component (ABC) layer that does more specific tasks. The service-based measurement measures a general software module (procedure-based software unit) in terms of export services.

Export services are represented using EI with ordered pair interactions, *R-C, R-U, R-R* and *U-U.* For example, *R-C* interaction provides a service that forces a business object within the business component to operate on a business state with *R* operation, and another business object of the request service operates on its business state with *C* operation.

Representation of the Business Component Model

The primary elements of a business component are business activities that specify required business objects. Moreover, a business component may provide

services for other components so that there is an optional component element — interface.

Since XML is a standard language, XML-based representation is helpful and meaningful for exchanging design information with other developers on different platforms and with varying development tools. Moreover, an XML-based model provides information about a business component that is easy to bundle and deploy in a particular commercial business components environment, such as Enterprise JavaBeans (Sun Microsystems, 2000). The business component model is represented by using an XML-based description in order to be applied efficiently to Web computing environments. The respective XML DTD is shown in Figure 3.

Figure 3. XML DTD of Business Components

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE    BusinessComponent[
<!ELEMENT    BusinessComponent     (BusinessObject+, BusinessActivity+, Interface*)>
<!ATTLIST    BusinessComponent     name    ID      #REQUIRED>
<!ELEMENT    BusinessObject        (Structure, Behavior)+>
<!ATTLIST    BusinessObject        id      ID      #REQUIRED>
<!ELEMENT    Structure             (Attribute)+>
<!ELEMENT    Behavior              (Method)+>
<!ELEMENT    Attribute             #PCDATA>
<!ATTLIST    Attribute             id      ID              #REQUIRED
                                   type    CDATA           #REQUIRED>
<!ELEMENT    Method                #PCDATA>
<!ATTLIST    Method                name    CDATA           #REQUIRED
                                   input   ((BusinessDataUnit*, Attribute*)*)
                                   output  ((BusinessDataUnit*, Attribute*)*)>
<!ELEMENT    BusinessActivity      (BusinessObjectList, BusinessDataUnit+, Interaction+)
<!ATTLIST    BusinessActivity      id      ID      #REQUIRED>
<!ELEMENT    BusinessObjectList    (BusinessObjectRef)+>
<!ATTLIST    BusinessObjectRef     BC_name         IDREF   #REQUIRED
                                   BO_id   IDREF   #REQUIRED >
<!ELEMENT    BusinessObjectRef     #PCDATA>
<!ELEMENT    BusinessDataUnit      (AttributeId)+>
<!ATTLIST    BusinessDataUnit      id              ID      #REQUIRED
                                   BO_id   IDREF   #REQUIRED >
<!ELEMENT    AttributeId           #PCDATA>
<!ATTLIST    AttributeId           id              IDREF   #REQUIRED
<!ELEMENT    Interaction           (Method1, Method2*)+>
<!ATTLIST    Interaction           name            ID      #REQUIRED>
<!ELEMENT    Method1               #PCDATA>
<!ATTLIST    Method1               name            IDREF   #REQUIRED>
<!ELEMENT    Method2               #PCDATA>
<!ATTLIST    Method2               name            IDREF   #REQUIRED>
<!ELEMENT    Interface             (BusinessActivityList*,  PublicMethodList*)*>
<!ELEMENT    BusinessActivityList  (BusinessActivityId)*>
<!ELEMENT    PublicMethodList      (MethodName)*>
<!ELEMENT    BusinessActivityId    #PCDATA>
<!ATTLIST    BusinessActivityId    name            IDREF   #REQUIRED
<!ELEMENT    MethodName            #PCDATA>
<!ATTLIST    MethodName            name            IDREF   #REQUIRED
]>
```

In addition to requiring that business activities be processed correctly in a well-defined sequence, every business component should have one controller called a business process controller (BPC) to control and manage the collaboration of its business activities. More specifically, a set of main business activities has to be controlled by the BPC for collaborating in the required sequences.

WEB IMPLEMENTATION MODELING

Based on the MVC architecture, the elements of the business component model can be mapped into the elements of the MVC architecture as below:

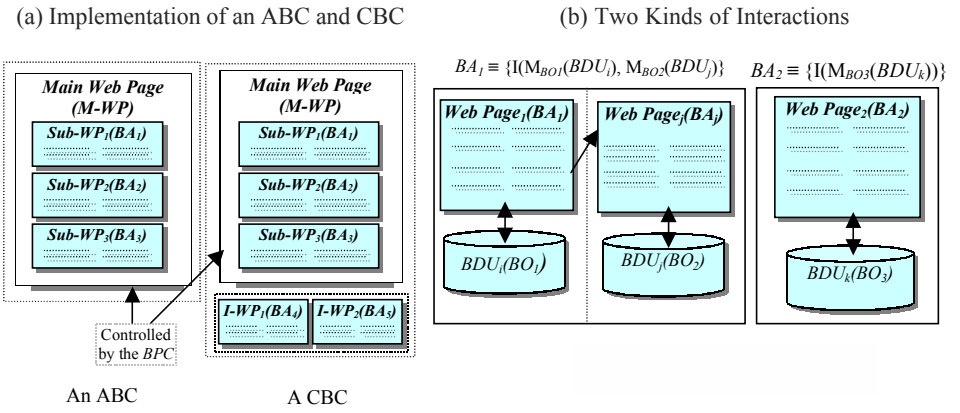
- Business activities and business objects are represented as models in the architecture. The model can be shared across all clients of the application. It should be consistent regardless of the type of client accessing the data.
- A BPC is represented as a controller using a main-SPT (M-SPT) Page. Business activities (business method invocations) represent a part of application behavior that is implemented as a subcontroller (sub SPT pages) within the M-SPT using elements of SPT page.
- A Business Data Unit is represented as a view embodied with its functions through HTML/XML interface (a Web interface). This interface is generated dynamically using SPT page.

The MVC abstraction prevents Web-page designers from interacting with an application developer's code and vice versa. Furthermore, the same model (BA&BO) can be used with other types of clients, for example, a CORBA client, with very few changes to the component. The following describes how the proposed business component model can be modeled regarding the Web-application systems.

The primitive elements of Web-application systems are Web pages derived from business data units of business objects, which are the dynamic contents of the Web pages. The links (navigations) among these Web pages are derived using the sequence of interactions between these business objects. A sequence of these links is governed and implemented using BPC. The implementation of primary component elements of the model on the Web follows the model described below.

- Each business component has only one main Web page (M-WP) containing a set of sub-Web pages (sub-WPs), which represent the main business activities of the business component. In addition to the CBC, the independent Web pages (I-WPs), which implement the independent business activities, can be constructed and deployed independently of the M-WP and will be linked to other Web pages (see Figure 4(a)).
- Every business activity can be represented with a set of interactions involving one or more business objects. Such interactions are implemented with different component elements of Web applications and suppose that:

Figure 4. Implementation of (a) ABC and CBC and (b) Interactions



$BA_1 \equiv \{I(BO_1, BO_2)\} \equiv \{I(M_{BO1}(BDU_i), M_{BO2}(BDU_j)\}$ and
$BA_2 \equiv \{I(BO_3)\} \equiv \{I(M_{BO3}(BDU_k)\}$.

These two business activities can be implemented by Web-application components that support different tasks (see Figure 4(b)). BA_1, for example, requires the interaction that involves BO_1 and BO_2. Since BO_1 and BO_2 operating with their respective BDUs enable creation of their own Web pages, the interaction between BO_1 and BO_2 within the BA_1 represents an interaction between business activities. However an interaction between two business components may have no user interfaces (Web pages).

CONCLUSION

The primary contributions of this chapter are the business component modeling technique and the measures defined using graph theory. The former enhances the advantages of both process- and data-oriented modeling methods. The proposed modeling method can capture both business processes and business data by using a uniform concept driven by the business processes in a more natural approach. The basic idea is that components relate primarily to business processes and activities, instead of to objects as constituent parts. As a result, the business components modeled using such a method can precisely represent business requirements. Secondly, the measures defined formally using graph theory can not only be used to ensure the business components satisfy the low coupling and high cohesion requirements, but also allow development tools to measure the coupling and cohesion automatically.

XML with its metadata description capabilities is used for representation of and working with the suitable business component model to support the software

development processes. Such a representation allows convenient and functionally flexible adaptation to a Web-based computing environment for developing multitier distributed applications.

In accordance with enormous efforts toward transition to a new generation Web, we accept the proposed XML-based model description as a good basis for extension to not only structural but semantic representation and processing of business components.

REFERENCES

Agarwal, R., De, P., & Sinha, A.P. (1999). Comprehending object and process models: an empirical study. *IEEE Trans. Software Eng., 25*(4), 541-555.

Bieman, J.M., & Kang, B-K. (1998). Measuring design-level cohesion. *IEEE Trans. Software Eng., 24*(2), 111-124.

Briand, L.C., & Morasca, S. (1999). Defining and validating measures for object-based high-level design. *IEEE Trans. Software Eng., 25*(5), 722-743.

Booch, G. (1994). *Object-oriented analysis and design with applications* (2nd ed.). Redwood City, CA: Benjamin/Cummings.

Brown, A.W., & Wallnau, K.C. (1998, September/October). The current state of CBSE. *IEEE Software,* 15(5), 37-46.

Chen, J.Q., & Heath, R.D. (2001, Winter). Building web applications: Challenges, architectures, and methods. *Information Systems Management,* 68-79.

Chidamber, S.R., & Kemerer, C.F. (1994). A metric suite for object oriented design. *IEEE Trans. Software Eng., 20*(6), 476-493.

Digre, T. (1998). Business object component architecture. *IEEE Trans. Software, 15*(5), 60-69.

Frolund, S., & Guerraoui, R. (2002). E-transaction: End-to-end reliability for three-tier architectures. *IEEE Trans. Software Eng., 28*(4), 378-395.

Grasner, G., & Pope, S. (1988). A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. *Journal of Object-Oriented Programming, 1*(3).

Herzum, P., & Sims, O. (2000). Business component factory: A comprehensive overview of component-based development for the enterprise. NJ: John Wiley & Sons, Inc.

Hitz, M., & Montazeri, B. (1996). Correspondence of Chidamber and Kemerer's metric suite: A measurement theory perspective. *IEEE Trans. Software Eng., 22*(4), 267-271.

Huang, G. Q., & Mak, K. L. (2001). Issues in the development and implementation of Web applications for product design and manufacture. *International Journal Computer Integrated Manufacturing, 14*(1), 125-135.

Jacobson, I., et al. (1995). *Object-oriented software engineering (OOSE): A use case driven approach (Rev. ed.)*. Boston, MA: Addison-Wesley Publishing Company.

Jacobson, I., Ericsson, M., & Jacobson, A. (1994). *THE OBJECT ADVANTAGE: Business process reengineering with object technology*. Boston, MA: Addison-Wesley Publishing Company.

Jacobson, I., Griss, M., & Jonsson, P. (1997). *Software reuse: Architecture, process and organization for business success*. Boston, MA: Addison-Wesley Publishing Company.

Kenneth, H. R. (1999). *Discrete mathematics and its applications* (4th ed). McGraw-Hill International.

Kocharekar, R. (2001). K-commerce: knowledge-based commerce architecture with convergence of e-commerce and knowledge management. *Information Systems Management, Spring*.

OMG (1999). *Business object concepts* (white paper, OMG document no. bom/99-01-01). (Object Management Group) OMG-Business Object Domain Task Force.

Puc-Rio & Rossi, G. (2000). Web Engineering: Introduction to Minitrack. *Proceedings of the 33rd Hawaii International Conference on System Sciencem*, 6, (January 4-7, p. 6065).

Snoeck-S, M., & Dedene, G. (2000). *An architecture for bridging OO and Business Process Modelling*. IEEE Int'l Conference on Technology of Object-Oriented Language and System (TOOLS'00).

Sun Microsystems. (2000). *Enterprise JavaBeans specification version 1.1*. Sun Microsystems Inc. Found at: http://java.sun.com/products/ejb/docs.html.

W3C-XML (2000, October 6). *Extensible markup language (XML) 1.0 (2nd ed.), W3C recommendation,* Retrieved from http://www.w3.org/TR/2000/REC-xml-20001006.

W3C-XSL (2000, November 21). *Extensible stylesheet language (XSL) version 1.0. W3C candidate recommendation.* Retrieved from http://www.w3.org/TR/xsl/.

ENDNOTES

1 This included D operation.

2 C-U interaction pattern covers U-R pattern.

3 C-R interaction pattern covers R-C pattern.

4 U-R interaction pattern covers R-U pattern.

Chapter XIV

An OO Methodology Based on the Unified Process for GIS Application Development

Jesús D. Garcia-Consuegra
Universidad de Castilla-La Mancha, Spain

ABSTRACT

This chapter introduces an object-oriented methodology for Geographical Information Systems (GIS) development. It argues that a COTS-based development methodology combined with the UML, can be extended to support the spatiotemporal peculiarities that characterize GIS applications. The author hopes that by typifying both enterprises and developments, and, with a thorough knowledge of the software component granularity in the GIS domain, it will be possible to extend and adapt the proposed COTS-based methodologies to cover the full lifecycle. Moreover, some recommendations are outlined to translate the methodology to the commercial iCASE Rational Suite Enterprise and its relationships with tool kits proposed by some GIS COTS vendors.

INTRODUCTION

As in other domains, the development of GIS follows the new trends in technology (i.e., Microsoft .NET, WEB services, etc.) and in methodology (Component Based Development (CBD)). The object-oriented paradigm has also been adopted in the GIS developments. For example, object-oriented modeling languages, such as UML, have been extended to support spatiotemporal characteristics. New proposals to extend, adapt or complement previous solutions in areas, such as requirements engineering, modeling languages, methodologies, patterns, and so on, are continually appearing in the scientific community. However, from the enterprise point of view, the use of these proposals is not a straightforward matter.

On the other hand, the market offers new alternative software products or new versions, with the interoperable capability based on the current standards (Opengis, FGDC, ISO, etc.). However the capability to integrate these new software products at the same rate as they emerge is often lacking, when the methodology used does not contemplate this kind of change. So GIS application developments, particularly those carried out in small and medium enterprises, face the same risks or problems as the software developed in other domains:

- inaccurate understanding of end-user needs;
- inability to deal with changing requirements;
- modules that do not fit together;
- software that is hard to maintain or extend;
- late discovery of serious project flaws;
- poor software quality;
- unacceptable software performance;
- team members are in each other's way, making it impossible to reconstruct who changed what, when, where and why;
- an untrustworthy build-and-release process;
- businesses' demand for increased productivity, improved quality with faster development and deployment and the building of software in a repeatable and predictable fashion.

In this chapter, we outline a methodology that covers the full lifecycle of the software engineering process in the GIS domain. Software engineering addresses a wide diversity of domains (e.g., banking, transportation, manufacturing), tasks (e.g., administrative support, decision support, process control) and environments (e.g., human organisations, physical phenomena). The scientific community agrees that a specific domain/task/environment may require some specific focus and dedicated techniques. This is the case with GIS due to its spatiotemporal peculiarities. In particular, activities like domain analysis as well as requirement elicitation and specification are the main topics of research carried out in the GIS domain. This

methodology attempts to integrate the currently available methodologies with the trends, capabilities and limitations imposed by the technology.

In particular, this chapter focuses on those developments based on COTS components.In order to make an OO Methodology proposal to solve the aforementioned problems, GIS enterprises and their developments have been studied. Those OO methodologies proposed by software engineering as a solution to software development processes have also been identified. Finally, some modifications and extensions needed to cover the GIS development have been proposed. Thus, our methodology makes full use of the best ideas currently available in software engineering as a foundation, extending them to take into account the peculiarity of the GIS domain.

BACKGROUND

GIS

Currently a great number of applications need modeling capabilities of spatial, temporal and spatiotemporal data for the complex object description. Solutions to these requirements can be found in GIS, in some database management systems with spatial extensions (such as Oracle, IBM, etc.) or in research forums.

The space and time combination is implicit in a wide range of needs. One of these is to observe a moving object and to predict its future locations. For example, the location of a moving car changes over time, and its location must be captured at given instants. Another need is to observe objects that do not change their position, but whose attributes change over time. Finally, there are objects in which both attributes and spatial characteristics can change over time (Worboy, 1994).

GIS applications fall into the "data intensive applications" category, often referred to as "non-standard," including, among others, multimedia and VLSI design. The peculiarities of this kind of spatiotemporal application are centred on the support of complex objects and relationships between them and long transactions. Furthermore, spatiotemporal applications deal with three types of objects: those whose position in space can change in time; those whose characteristics and position (shape change) can change in time; and both of the above, simultaneously. These matters have to be dealt with in several stages of the software lifecycle: in the analysis (modeling the business logic and the user interface), requirements specifications, etc.

GIS Enterprise and Application Classification

In the last few years, the software enterprise has been moving very swiftly towards component-based development. This trend, far from being unified and coherent, is the result of many trends, some of them complementary, others oriented to reinforce forgotten aspects and still others that are totally independent.

For this reason, this study distinguishes between methodologies oriented towards the development of components (custom components or in-home components) and those involved in the integration of pre-existing software components (COTS). This approach allows us to make a good approximation to the typology of developments in GIS domains, shown below. Although both are component-based systems, the development process in the first case is called Component Based Development (CBD), and in the second is called COTS Based Development (COTSBD) throughout this work. As in the majority of domains, the software developments in GIS can be divided into two categories:

Software Component Development. ESRI (ESRI) and Intergraph (INTERGRAPH) would be two enterprise examples. They develop generic software components (COTS). Software engineering (SE) proposes methodologies like Catalysis (D'Souza & Wills, 1999), Business Component Factory (Herzum & Sims, 2000) or Unified Process (Kruchten, 2000).

Developments based on previous software components. This category can be divided into two new subcategories:

- Developments that basically focus on component integration.
- Developments that focus on building new components and are highly dependent on previous COTS, either because they are not on the market or they are their business case.

In Spain, TAO, GIM or Absys are good examples of enterprises involved in this kind of development. Methodologies, such as Off-The-Shelf Option (OTSO) (Kontio, 1995), COTS-based Integrated Systems Development (COTS CISD) (Tran & Liud, 1997), Infrastructure Incremental Development Approach (IIDA) (Fox, Marcom, & Lantner, 1997), IusWare (Morision, Seaman, Basili, Parra, Kraft & Condon, in press), Procurement-Oriented-Requirements Engineering (PORE) (Ncube, 2000) or COTS-Aware Requirements Engineering Technique (CARE) (Chung, Cooper & Huynh, 2001), provide them with a solution.

OOMGIS

When the challenge to identify a methodology for fast application development in the GIS domainn was faced, several important questions arose. As the fast development of GIS applications was the major goal, the COTSBD approach was selected as paradigm for the application construction. COTSBD gives the chance to develop applications using pre-existing software components, thereby reducing cost and delivery time. The application construction is mainly reduced to selected software components integration, previously looked up (using repositories or traders) and purchased. This is, moreover, the current situation in the GIS domain, where a group of enterprises is offering components that support the basic functionality with

architectures given, as shown in the previous section. Furthermore, our intention is that applications that are developed should not be attached to specific components, to a specific technology and to a given enterprise. This means that the possibility of selecting components according to the requirements of every system has to be considered.

On the other hand, a methodology must treat the root causes of the problems of software development in order to eliminate these symptoms. That is, the methodology must be a conjunction of the best software practices. This involves developing software iteratively, managing requirements, using component-based architectures, visually modeling software, continuously verifying software quality, controlling changes to software, etc.

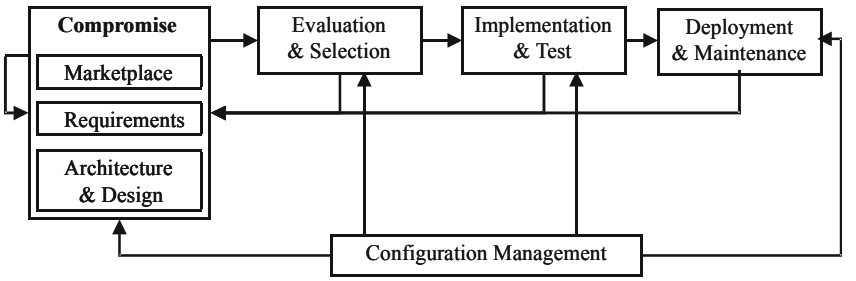
As can be seen in Figure 1, OOMGIS is an iterative and evolutionary process that takes the best ideas of COTSBD methodologies [such as, CARE (Ncube, 2000) and PORE (Chung et al., 2001)], as a foundation. OOMGIS plays particular attention to those matters relating to the iterative Requirements Acquisition and Product Evaluation/Selection process. Furthermore, those OOMGIS parts that have to be adapted to support the peculiarities of the GIS domain (The European Commission, 2000), like requirement specification and system modeling, have been identified.

There is a lot of research taking place in the GIS domain. In general, the research is focused on providing solutions for particular problems or limitations, instead of proposing a full lifecycle software engineering process. OOMGIS tries to cover this gap. Thus, OOMGIS covers the basic activities associated with a COSTBD methodology.

The requirements, marketplace and architecture and design activity sets operate concurrently and in co-operation with one another in order to obtain a compromise in the COTS selection.

The requirements activity set defines, prioritizes, and constrains the CBS to be fielded, accounting for functional and nonfunctional requirements, end-user processes, business drivers, the operational environment and constraints (such as, policies, schedules and budgets). This consists of requirements gathering and

Figure 1. The OO Methodology (OOMGIS) for COTS-Based Development in the GIS Domain



analysis. The initial requirements must be gathered with a high-abstraction level. The objective is to define the context in which the system will operate and the main functions the system will provide. In this phase, the requirements, which will be used to establish the bases for selecting the suitable COTS candidates, are determined. Likewise, in this phase the following activities will be performed: to determine and prioritize (correct) the negotiable or non-negotiable elements of the requirements; to understand the essential elements of the user processes before studying the COTS market; to modify user processes as needed after the available products are known; to negotiate requirement changes and to restudy COTS products in order to optimize user processes.

The marketplace activity set bounds the COTS marketplace elements. Here, a search in the COTS market must be carried out to determine what components are suitable for the proposed requirement and architecture. The following tasks are performed: creation and maintenance of our knowledge of the actual (and emerging) market possibilities for the system and market re-exploration; the definition of new or modified requirements from the selected components in the next iteration and alerting the development team to promising new technologies. As in the previous activity, this activity takes place simultaneously with the requirements and architecture definition. Those products whose capabilities best cover the needs established by requirements and the architecture are the candidate products. It is also possible not to find suitable COTS in the market, such as a number of iterations requirements can render the candidate product unsatisfactory. Then, the make vs. buy decision — the best candidate or a great effort in coding and integrating — must be reconsidered.

The architecture and design activity set captures decisions about the structure of the components, interfaces and relationships of a system's components and the principles and guidelines governing their design and evolution. The architecture must allow efficient evolution of the system, although this is almost always a topic related to the selected COTS. Other aspects to be considered are its compatibility, configurability and integrability. Although some parts of the design are performed in the architecture definition, here these activities are repeated at a much greater level of detail, taking into account the product variants selected. This phase is also concerned with the definition of the integration of COTS and newly developed software. This activity is especially important due to the fact that in COTSBD the effort is shifted from the programming software to the composition (assembling of components) (Voas, 1998). Several COTS can be involved, each one with possibly different architectural styles, constraints, different vendors, and so on.

The evaluation and selection activity set examines and selects COTS products and technologies. The selection of the "best" from among the packages available depends on the assessment of their compatibility with the requirements specification and the prioritization of these requirements. Selection is not static. The purpose of evaluation is to examine COTS products and technologies, to gather information

about them and to appraise them in support of making COTS-based system decisions.

Its activities are, among others: to plan and to design the evaluation, to locate potentially relevant candidates, to perform appropriate analyses for selection of appropriate technologies or products and to document and to share acquired information for use in decision making.

The implementation and test activity set addresses implementation of custom components, COTS integration (parameterizing and writing glue and wrapper code) and system integration and test. This stage composes and integrates a COTS-based system from available parts.

The deployment and maintenance-activity set encompasses initial and continuing delivery of a COTS-based system to end users and system maintenance.

The configuration management activity set establishes and maintains system artefact integrity and traceability throughout the CBS's lifetime.

There are several proposals to define the software component definition and its granularity. According to Herzum and Sims (2000), these can be classified into:

- **Distributed component:** This is the lowest granularity of a component. It fits the usual concept of component in industry. (For example, it may be implemented as an Enterprise JavaBean, as a CORBA or a DCOM component.) It is normally, but not necessarily, built using an object-oriented programming language (OOPL).
- **Business component:** This implements a single autonomous business concept. It usually consists of one or more distributed components, which cover its distribution aspects. That is, the business component is a single artefact that may physically be distributed across two or more computers. The business component is built using the software component technology, like a composition of distributed components. It is a component, not only at deployment and runtime, but also during the development life cycle. It is a medium-grained component. It represents those that are relatively autonomous in the problem space.
- **Business component system:** This makes up the largest grained component. It corresponds to an information system or "application" (for example, an e-mail system or a cadastral GIS application). The business component system concept can be defined as: "a set of co-operating business components assembled together to deliver a solution to a specific business problem."

In the GIS domain, there are two main vendors (ESRI and Intergraph) whose components are the most frequently used in desktop and Web-application developments. ESRI provides two options: MapObjects and ArcObjects, the last and more complete collection of software components. On the other hand, Intergraph has Geomedia Objects, a collection consisting of 17 ActiveX controls (OCX), nine data

servers and more than 90 programmable ActiveX Automation objects. One of the major capabilities of Geomedia 4 is its interoperability with other systems. Both ArcObjects and Geomedia Objects have been developed with Microsoft technology based on the COM protocol. Basically, the foundations for their development have been the same. They are fully incorporated into the standardization train imposed by the several international organizations (Opengis, ISO, FGDC, etc.). Both allow customization using the built-in VBA scripting capabilities or a COM-compliant programming language, such as Visual Basic, Visual C++ or Delphi. Their differences are centred on their capabilities and their object models.

These technologies provide a set of classes that implement spatial characteristics, which can be extended to integrate new behavior or new attributes to the geographical entities modeled. Temporal properties are usually implemented by the capabilities of the DBMS used in each case.

Thus, we can conclude that the granularity of the software components offered in the GIS marketplace is equivalent to the distributed and business components (Herzum & Sims, 2000). Furthermore, considering our market analysis and studying the pros and cons of both, the granularity suitable for COTSBD must be thought of in terms of a business component, since using their distributed components involves a high integration effort. Other problems solved by this selection are those associated with evolution. To integrate distributed software components from different vendors requires a great effort. In this sense, the marketplace must consider the creation and maintenance of a sound knowledge of the components to provide an accurate foundation for candidate component selection and foreign requirements definition (Chung et al., 2001), to understand the impact on the architecture and design and, finally, to implement and configure the component management. Furthermore, a team must be well trained in all selectable components. Emergent components, new versions or a change of components can involve serious modifications mainly in the architecture and design as well as in the implementation activities.

At present, the iCASE tool that supports a methodology is as important as the methodology itself. So, when we were working on the definition of a methodology to develop GIS applications, we also considered developing an iCASE or adapting an existing iCASE. iCASE complexity demands a great effort for its development and maintenance. The extension or adaptation of an existing iCASE is the most suitable option. In this case, the Rational Enterprise Suite has been selected to implement OOMGIS. This selection takes advantage of the Software Component Development, which is covered due to Rational Enterprise Suite implements Rational Unified Process (RUP) (Kruchten, 2000). It provides the mechanisms to extend and modify the process. It supports UML as the modeling language as well as the mechanisms to extend it. Finally, like every commercial package, its evolution involves OOMGIS evolution as well.

To carry out the OOMGIS implementation on Rational Suite Enterprise, methodologies like Model-Base Architecting and Software Engineering (MBASE)

(Boehm, Port, Abi-Antoun & Egyed, 1998) or Information Technology Solutions Evolution Process (ITSEP) (Albert, 2002) are taken into account. The MBASE approach is a unified approach that considers four types of models for a system: success, process, product and property. MBASE extends the spiral model for the integrated product and process development. It uses four guiding principles to develop value-driven, shared-vision-driven, change-driven and risk-driven requirements. MBASE is compatible with Integrated Capability Maturity Model (CMMI) and RUP. ITSEP relies on the A/A PCS Process, a framework for COTS-based engineering and on the RUP. The A/A PCS Process supports simultaneous trade offs among requirements, preferences, marketplace offerings, architecture, design and business processes. The process is a spiral approach where the incremental accumulation of knowledge is used to reduce the risk.

In addition to a methodology, a modeling language with a sufficiently large expressive wealth for supporting the model elements is needed. For the chosen environment (OO paradigm and Rational Suite Enterprise), UML is the most suitable modeling language. As proposed in Faria, Medeiros and Nascimento (1998), using the core constructs of UML, spatial and temporal properties can be added to an object class definition by associating it with temporal and spatial object classes. To accomplish this task, each spatiotemporal attribute of a class is promoted to a separate, but associated, class with the appropriate time stamps and spatial extents as attributes. Temporal classes and associations are treated similarly, by adding timestamp attributes to the class or to the association class, respectively, in the latter case after promoting the association to an association class. In Price, Tryfona and Jensen (1999), this approach is reported as not suitable for representing temporal or spatial variation at the attribute level as the timestamp and spatial locations are defined only at the object-component level. Furthermore, these diagrams, when presented in this fashion, are visually highly complex, which gives them an unnatural feel and makes them difficult to follow.

The alternative solution is to extend the UML-class diagram in order to model the spatiotemporal characteristics without discarding the simplicity of diagrams. UML provides the "stereotypes" as the usual way to make domain-specific extensions to its core constructs, which can be used as if they were of UML's original metamodel or definition.

In the scientific community, different temporal, spatial and spatiotemporal models can be found: TEMPOS (TEMPOS, 1999), MADS (Parent, Spaccapietra & Zimanyi, 1999), TOOBIS (TOOBIS, 1999), RDNSAAE (Böhleny, Jensen & Skjellaug, 1998), ScanGIS (Renolen, 1997), EGSV (Erwing, Güting, Schneider & Vazirgiannis, 1999), STER (Hadzilacos & Tryfona, 1998), GeoOOA (Kösters, Pagel & Six, 1997), Fuzzy-UML (Yazici, Zhu & Sun, 2001) and ST-UML (Price et al., 1999). The majority of these were proposed from the database perspective. The ST-UML is the best model to cover GIS needs, since it integrates the spatial and temporal aspects. However, for our purposes, ST-UML suffers from certain limitations:

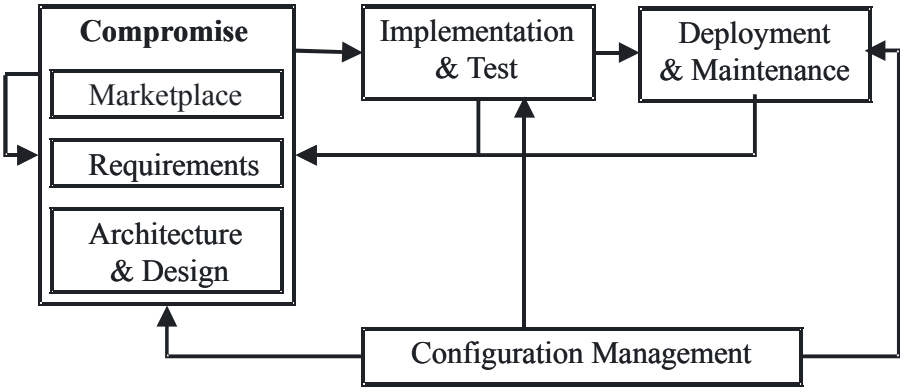
- It does not cover its implementation. This stage can follow the EGSV or TOOBIS models.
- Temporal aspects are not completely supported as in TEMPOS or TOOBIS.
- Its syntax can, in some models, be complex.
- It only covers the class diagram.
- The UML extension in Rational Rose that implements ST-UML is not straightforward.

In Rational Rose, add-ins allow customizations and automation of several Rose features included in one package through the Rose Extensibility Interface (REI). An add-in is a collection of certain characteristics. In this case, our interest is focused on the following characteristics:

- **Properties:** Rose model properties allow us to extend Rose model elements through additional properties and their values. This characteristic is used to implement the Specification Box (Price et al., 1999), since it involves a set of properties to be added to the elements of the model.
- **Stereotypes:** Rose stereotypes allow us to customize the look of different model elements according to the requirements of your add-in. This is the natural way to add the characteristics proposed in ST-UML.
- **Functionality**: Some code can be programmed in order to provide the dialog boxes and other functionality desired. It is used to adapt the code to cover our interests.

In the COTS and CBD methodologies, a technology change (for example, a ESRI COTS new version or a change to Intergraph technology) involves a high cost (model evolution and/or increment of the glue or wrap code) in the maintenance of the models. The way in which this extension is carried out gives it an important advantage when dealing with this problem. It can provide the mechanisms through which the model may achieve maximum independence from its implementation with a specific technology. At present, following ST-UML, our proposal consists of extending the UML Class Diagram by defining a set of class stereotypes with their corresponding tag values. New stereotypes to model specific application domains can be defined as subclasses. Once the model is complete, the stereotypes are translated into a standard UML Class Diagram, following the guidelines and observing the restrictions of each technology. The resulting standard UML Class Diagram can be processed with the specific tools provided by each vendor (i.e., ArcGIS). Thus, modeling is carried out in a canonical way, regardless of the final technology to be used. At this point, we are engaged in defining stereotypes for covering the spatial, temporal and spatiotemporal GIS peculiarities.

Figure 2. The Reduced OO Methodology



RESULTS

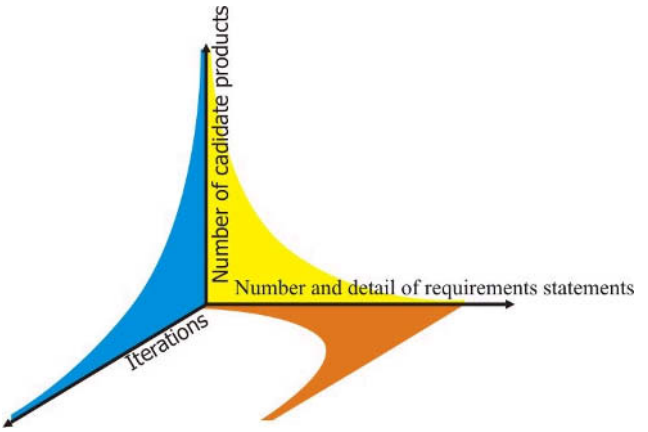
Our experience shows that most of the small and medium enterprises involved in GIS applications development cannot pay for the costs arising from the software component study in marketplace activity. Thus, they introduce a simplification in the software development process defined (see Figure 2).

- The compromise activity remains as is. Nevertheless, the marketplace is reduced to creation and maintenance of the knowledge of the chosen COTS; the definition of new or modified requirements to fit the capabilities of the COTS used and alerting the development team to new versions.
- The evaluation and selection activity is erased since only a business component is considered.

In the COTSBD process, one of the key lifecycle aspects is the early evaluation and selection of the candidate COTS products. Its success depends, principally, on the correct understanding of the capabilities and constraints shown by individual product candidates. In order to achieve success, the evaluation and selection of software product candidates must start at the same time as the client requirements elicitation activity. To obtain a reasonable level of security from the product evaluation and selection, rigorous methodologies are needed to guide this activity and the client requirements elicitation activity (Ncube, 2000).

An erroneous requirements identification and acquisition can seriously affect the resulting system. Furthermore, in COTSBD, the successful selection of candidate products also largely depends on requirements. These products are selected according to their degree of compliance to customer requirements. Most current

Figure 3. The Iterative Process of the Requirement Acquisition and the Product Evaluation/Selection Activities



research focuses on integration of selected COTS products (Shaw & Garlan, 1996), selection and integration processes and, in the last few years, also on how requirements must be acquired to make the COTS evaluation and selection processes easier.

In the requirements elicitation phase in the GIS domain, user requirements, GIS peculiarities, capabilities offered by and the inherent constrains on the selected, or selectable COTS must be taken into account. The latter is frequently forgotten, involving high wrapper-code developments. The impact of the use of COTS on the requirements specification has established the distinction between native and foreign requirements (Chung et al., 2001). The former refer to the set of requirements defined regardless of whether the development is based on COTS components or not. COTS capabilities make up the foreign requirement. So, in a COTSBD, the requirements for the system are composed of a mix of foreign and native requirements. Likewise, RE must consider the dependencies, overlaps and associations between products in the market, user requirements and the system architecture, because they influence each other (Thomas, 1999).

In that sense, there are some guidelines, such as those provided by the European Commission (ESPRIT Programs) that can be consulted in Best-GIS The European Commission. They should be used in the requirement specification phase of any COTS-based development methodology in the GIS domain (as in this case). These requirements are opened for possible extensions as experience dictates. The user interface has great importance in the GIS domain. Requirements elicitation and analysis ought to take them into account, considering proposals such as those in Lozano, González & Ramos, 2001).

COSTBD is an iterative process with the compromise and evaluation/selection activities making up the main loop (Figure 3). In order to reduce the development cost and time, thereby increasing productivity, the number of iterations must be reduced

as far as possible. At the same time, mechanisms to automate the compromise activity must be defined or even implemented as tools to contribute to the previous aims. Possible fields of research are:

- to automate COTS candidate impact on architecture and foreign requirements specifications, model definition in a UML spatiotemporal extension and test cases definition, paying particular attention to the patterns;
- to define user interfaces and automate their implementation;
- to define and use metrics in order to evaluate quality aspects;
- to make reuse aspects easier.

CONCLUSIONS

GIS applications are sufficiently large, complex and important to warrant SE skills being set to work on their development. In this chapter, research in progress to develop an OOMGIS for COTSBD in the GIS domain, has been discussed. OOMGIS makes the maximum possible use of the best ideas of currently available methodologies, especially those based on COTSBD, as a foundation. This chapter also outlines some research fields that can contribute to reducing the development cost and time in this domain. On the other hand, the extension of a commercial iCASE like Rational Suite Enterprise and UML to support the methodology and the spatiotemporal GIS peculiarities, respectively, are also depicted. The selection of Rational Suite Enterprise is based on its support of Unified Process and their capabilities for adaptation, as well as their strong component in SQA, especially in the management of configuration, a critical part of the COTSBD.

REFERENCES

Albert, C. (2002, February). Meeting the challenges of commercial off-the-shelf (COTS) products: The information technology solutions evolution process (ITSEP). International Conference on COTS-Based Software Systems ICCBSS2002, Orlando, FL, USA, (February, 2002).

Boehm, B., Port, D., Abi-Antoun, M., & Egyed, A. (1998). *Guidelines for the life cycle objectives (LCO) and the life cycle architecture (LCA) deliverables for model-based architecting and software engineering (MBASE)* (Tech. rep. USC-CSE-98-519). Los Angeles, CA: USC-Center for Software Engineering.

Böhleny, M., Jensen, C. S., & Skjellaug, B. (1998). Spatio-temporal database support for legacy applications. *Proceedings Of the ACM symposium on Applied Computing, Atlanta, GA* (pp. 226-234) New York.

Chung, L., Cooper K., & Huynh, D. T. (2001). COTS-aware requirements engineering techniques. *Proceedings of The 2001 Workshop on Embedded*

Software Technology (WEST'01). Found at http://www.utdallas.edu/~kcooper/research/WEST.pdf.

D'Souza, D. F., & Wills, A. (1999). *Objects, components and frameworks with UML*. Addison-Wesley.

Erwing, M., Güting, R. H., Schneider, M., & Vazirgiannis, M. (1999). Spatio-temporal data types: An approach to modelling and querying moving objects in databases. *GeoInformatica 3*(3), 269-296.

ESRI (2003). Retrieved from http://www.esri.com.

The European Commission. (2000). BEST-GIS: best practice in software engineering and methodologies for developing GIS applications (Project no. 21580). *ESPRIT Programme.* Retrieved January 2003 from http://www.gisig.it/best-gis/guides/main.htm.

Faria, G., Medeiros, C. B., & Nascimento, N. A. (1998). An extensible framework for spatio-temporal database applications (Tech. rep. TR-27, pp. 1-15). Time Center. Found at www.cs.auc.dk/research/DP/tdb/TimeCenter/TimeCenterPublications/TR-27.ps.gz.

Fox, G., Marcom, S., & Lantner, K. (1997). A software development process for COTS-based information system infrastructure. *Proceedings of the 5th International Symposium on Assessment of Software Tools and Technologies (SAST'97),* (pp. 133-143).

Hadzilacos, T., & Tryfona, N. (1998, September). Evaluation of database modelling methods for Geographic Information Systems. *Australian Journal of Information Systems 6*(1), 15-26.

Herzum, P., & Sims, O. (2000). *Business component factory*. New York: John Wiley & Sons Inc.

INTERGRAPH, Retrieved from http://www.intergraph.com/gis.

Kontio, J. (1995). *OTSO: A systematic process for reusable software component selection* (Tech. rep. CS-TR-3478). College Park, MD: University of Maryland.

Kösters, G., Pagel, B.-U., & Six, H.-W. (1997). GIS-application development with GeooAA. *International Journal Geographical Information Science 11*(4), 307.

Kruchten, P. (2000). *The rational unified process an introduction* (2nd. Ed.). Addison-Wesley.

Lozano, M., González P., & Ramos, I. (2001). User interface generation: current trends. Informatik/informatique, No. 2. Swiss Informatics Society, Abril.

Morision, M., Seaman, C. B., Basili, C. R., Parra, A. T., Kraft, S. E., & Condon, S. E. (in press). COTS-based software development: Processes and open issues. *Journal of Systems and Software*.

Ncube, C. (2000). A requirements engineering method for COTS-Based systems development. Centre for Human-Computer Interaction Design, Londres.

Parent, C., Spaccapietra, S., & Zimanyi, E. (1999) Spatio-temporal conceptual models: data structures + space + time. Proceedings of the *7th ACM Symposium on Advances in GIS, Kansas City, Kansas*.

Price, R., Tryfona, N., & Jensen, C. S. (1999). *A conceptual modelling language for spatio-temporal applications* (Tech. rep. CH-99-20).

Renolen, A. (1997). Conceptual modelling and spatio-temporal information systems: How to model the real world. ScanGIS'97 (June 1-3, 1997).

Shaw, M., & Garlan, D. (1996). *Software architecture—perspectives on an emerging discipline.* Prentice Hall Inc.

Temporal Object-Oriented Databases in Information Systems (TOOBIS). (1999, June). EEC-funded project of the ESPRIT-IV framework. Retrieved 2002 from http://www.mm.di.uoa.gr/~toobis/.

TEMPOS. (1999). A temporal database model seamlessly extending ODMG. Retrieved from http://citeseer.nj.nec.com/21283.html.

Thomas, B. (1999). Meeting the challenges of requirements engineering, spotlight. *SEI Interactive, 2*(1).

Tran, V., & Liud, D. (1997) A procurement-centric model for engineering component-based software systems. *Proceedings of the 5th International Symposium on Assessment of Software Tools and Technologies (SAST'97)* (pp. 70-80).

Voas, J. (1998, July/August). Maintaining component-based systems. *IEEE Software, 15*(14), 22-27.

Worboy, M. S. (1994). A unified model for spatial and temporal information. *The Computer Journal, 37,* 26-34.

Yazici, A., Zhu, O., & Sun, N. (2001, July). Semantic data modelling of spatio-temporal database applications. *International Journal of Intelligent Systems 16*(7), 881-904.

Chapter XV

A Framework for Intelligent Service Discovery

Robert Bram
Monash University, Australia

Jana Dospisil
Monash University, Australia

ABSTRACT

The claim of improved efficiency and reliability of networking technology provides for a framework of service discovery, where clients connect to services over the network based on a comparison of the client's requirements with the advertised capabilities of those services. Many service directory technologies exist to provide this middleware functionality; each with their own default set of service attributes that may be used for comparison and each with their own default search algorithms. Because the most expressive search ability might not be as important as robustness for directory services, the search algorithms provided are usually limited when compared to a service devoted entirely to intelligent service discovery.

This paper proposes a framework of intelligent service discovery running alongside a service directory that allows the search service to have available a range of search algorithms. The most appropriate algorithm may be chosen for a search according to the data types found in the search criteria. A specific implementation of this framework will be presented as a Jini service, using a constraint satisfaction problem-solving architecture that allows different algorithms to be used as library components.

INTRODUCTION

The availability, speed and reliability of networking technology validates a service discovery framework, where clients connect to services over the network based on the advertised capability descriptions (Pascoe, 2000) of those services. Services in this framework are computer-based interfaces to devices, applications, objects or resources that a client uses.

The key challenge for such systems is to enable clients to locate the service that best suits their needs, where best will be defined by the client using infrastructure provided by the framework's implementation. The client's requirement description must be compared with the advertised capability descriptions of the services to find the best match (Pascoe, 2000).

Comparison of a client's requirement description with a service's capability description requires an infrastructure with a language for describing service attributes. Descriptions can be formed using this language and used as search criteria. Descriptions might include service attributes, such as type of service, or quality of service in terms of cost, speed or accuracy of results.

Service discovery frameworks may be classified according to where the comparison takes place. The locality could be: at the client site, at the server site or at a third party search service called a lookup. Lookup is typically a directory-based process of locating or looking up a specific service or activating an agent capable of doing the job (McGrath, 2000). Each of these styles has a different profile in terms of network traffic and a combination of all three is possible (Pascoe, 2000).

When using a lookup, the client must be able to provide the lookup process with enough detail for the service to be located. This detail may be a specific address or identification or it may be data to form some matching criteria with which the lookup process may search and build a satisfier set of services that match the search criteria.

This paper begins by giving a useful definition of a directory service with an attribute language and search language. Most directory services are either distributed file system or networked file sharing mechanisms. This paper then expands upon that definition to form the concept of a service directory that allows sharing of electronic services by electronic clients. Java's Jini technology is explored as a perfect example of a service directory that uses capability descriptions and requirement descriptions with exact pattern matching as part of its search language.

Exact pattern matching is found to be insufficient for some advanced search types that could be modeled as constraint satisfaction problems (CSP), and a definition of a CSP is given as part of a linear system and then defined as an object model. A process description is then given, showing at a high level of detail the general pattern of usage this system of intelligent searching is expected to fall under. Finally a conclusion and indication of future work are given.

DIRECTORY SERVICES

Directory services allow electronic resources to be shared, usually across homogenous networks. Directories may be subdivided into smaller lists or categories and may even be indexed by categories — subdirectories are the easiest way to achieve categorization. An operating system uses a directory structure to organize its resources into a single directory hierarchy. The hierarchy provides a reference to enable users and applications to access resources in the file system (Webopaedia.com, 2000). A directory is any searchable list of entries containing at least two things:

- A reference to a resource of some type that a client searcher may use to access that resource. The reference may be an identification, such as a name, or it may be a path or address formatted to the appropriate protocol, perhaps a network or file system protocol.
- A set of attributes that describe the resource in some way. If the entry contains a reference to a service, then the set of attributes will form the service's capability description.

The list exhibits properties that define how it may be manipulated through the infrastructure provided. These properties can include:

- whether elements of the list may be duplicated;
- whether elements of the list are naturally ordered. Possible ordering systems are: alphabetic, chronological and hierarchical;
- whether elements of the list are categorized. Categorization systems could be based upon: access rights, service type and any ordering system that allows grouping of multiple references;
- whether the list is indexed. Indexing allows elements in the list to be accessed arbitrarily, rather than having to traverse the list to find the desired element. A list may have more than one index.

All directory services maintain some form of descriptive attribute language to name services and/or define the attributes of services. The level of detail provided for by this language is reflected often in the expressive power offered by the search language allowed for by the directory. It is in the best interest of a service directory

to limit the expressive power of a search to provide better performance (Newmarch, 2001b).

Simple attribute languages provide only a small set of predefined attributes that cannot be changed with values from a limited type system: character, numeric or Boolean. Complex attribute languages allow new attributes to be created from a variety of types, even object types, that may exhibit behavior as well as store data. Simple search languages provide exact pattern matching or the selection of different indexes in a GUI. Complex search languages will provide mechanisms such as regular expression searches and comparative pattern matching.

Distributed file systems or peer-to-peer file sharing utilities (Newmarch, 2001b) usually have simple attribute language, defining a static set of file attributes to be searched with a substring search and allowing the results to be ordered by attribute values. Another type of directory service is one with resources that are other services.

SERVICE DIRECTORIES

Services find directories useful because it increases their exposure to potential clients. A directory service may increase the expressive power granted to services in the formulation of capability descriptions by the provision of a complex attribute language. The more complex an attribute language is, then the more detailed, varied and potentially useful information a service can provide about itself.

Clients find directories useful because they can search many references from one list, choosing the best reference for their purpose, according to the attributes for each reference recorded by the directory. A directory service may increase the expressive power granted to clients in the formulation of requirement descriptions by the provision of a complex search language. The search language allows clients to specify rules or relationships between attributes that must be satisfied for a search to succeed.

The service providers who wish to "sell" their services must register the services with the directory. The client searches available directories to find a set of suitable services by comparing their requirements with details about each service provided by the directory. The references registered with the service provides a pointer to the service or a proxy.

At a minimum, the client provides nothing more than a single-string service name or description, an object reference or some form of unique service identification (RMI). On the other end, a client can provide an open set of attribute values to be mapped to whatever services meet the description (Jini). This could still involve nothing more than a pattern match between service attributes in a capability description and attribute values provided in a requirements description; or it might involve the formulation of some mathematical model to be solved in order find a satisfier set.

JINI: A JAVA DIRECTORY SERVICE

In a peer-to-peer environment, directories are central repositories for information. Directory information software can be scalable, granular and independent of location and able to present and store metadata (Bowstreet, 2001).

Jini is a set of Java API's defining a directory service listing references to electronic services. There are four main components in a typical Jini scenario: three actors connected via a network as illustrated in Figure 1.

In Jini terminology, a directory service is a lookup (or registry). References stored in the directory are service items registered with the lookup that generally include proxies for computer-based service providers. Clients are computer-based entities who wish to access the services listed in the Jini lookup. They must search the Jini lookup for the service or services they want

There is a general pattern for how clients and service providers use the Jini service directory, which is shown below.

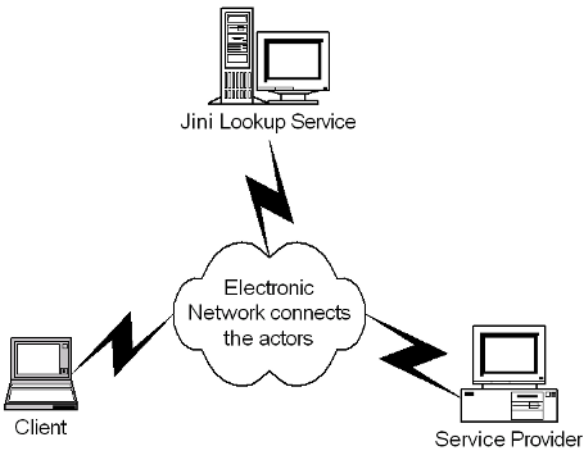
Services providers

1. Discover the Jini lookup directory. This makes use of the Jini discovery protocols.
2. Register a service item with the directory. This makes use of the Jini join protocols.

Clients

1. Discover the Jini lookup directory. This makes use of the Jini discovery protocols.
2. Provide a service template to the directory, which represents a clients' search criteria. This makes use of the Jini join protocols.

Figure 1. Actors in a Jini Scenario



3. Clients receive a satisfier set containing all service items that matched the search criteria contained in the service template.
4. Assuming the satisfier set returned to the client is not empty, the client chooses the most appropriate service item to make use of.
5. The client accesses the service item to perform whatever task they need done.

CAPABILITY DESCRIPTIONS AND REQUIREMENT DESCRIPTIONS

When service providers register with the lookup service, their capability descriptions are provided as ServiceItem objects sent to the lookup. When clients look up a service, their requirement descriptions are provided as ServiceTemplate objects sent to the lookup.

ServiceTemplates provide matching criteria for each field that appears in a ServiceItem. ServiceID is guaranteed to be globally unique for each Jini registered service. Clients may search for a specific service ID. The service object is an instance of a Java type and clients may search for a service object whose type list includes all elements of a particular array of Java types. Entry attributes describe the service in some way. At their most basic, Entry attributes represent name-value pairs; clients may search for services that are described by a specific set of attributes.

Table 1 shows the contents of a service provider's capability description as a ServiceItem against the content of a client's requirements description as a ServiceTemplate.

When the client submits a ServiceTemplate to the lookup service, a search is performed on all three criteria, then an exact pattern match will be performed first on the ServiceID, then on the object types and lastly on the Entry attributes.

The service item includes a service object. This object may be a proxy for the service provider or it may be self-contained. The Jini specification does not define any of this detail, including how the client and service provider must communicate if the service object is a proxy.

Entry Attributes

Entry objects describe the service object. Clients search by attributes that a service provider has registered with the lookup service along with the service object itself.

Table 1. Capability and Requirement Descriptions

ServiceItem	*ServiceTemplate*
ServiceID	ServiceID
service object	array of object types
array of Entry objects	array of Entry objects

There are a number of ready-made Entry objects, all of which extend AbstractEntry, which implements Entry. Entry objects cannot hold primitive types.

Direct Known Subclasses of Entry are:

- Address;
- Comment;
- Location;
- Name;
- ServiceInfo;
- ServiceType;
- Status.

Objects of these types will be used most frequently to describe Jini standard service objects. However Entry objects can easily be extended to include GUI components and, as this work shows, dynamic attributes.

Entry objects are specialized serial objects designed to be compared with each other in a very simple way. They are stored and compared in serial form. An Entry object cannot have any of the primitive types as fields; they must be contained in an object wrapper class, if needed. This makes it easier to compare Entry objects. A null field matches any value as a wildcard, while a non-null value must match exactly. Additionally, Entry classes must have a no argument constructor and ensure that the only fields that will be compared must be public, nonstatic, nontransient and nonfinal (Newmarch, 2001a).

Exact Pattern Matching

An attribute in the ServiceTemplate is matched by an attribute in the ServiceItem, if all fields of the attribute that are non-null match exactly their corresponding field in the ServiceItem. A null field in the ServiceTemplate will match any value of the corresponding attribute in the ServiceItem. This is the exact pattern match implemented by Jini (Edwards, 2001). The attributes are kept serialized even for comparison. The value of two fields are considered matched, if they have the same sequence of bytes according to the Java serialization scheme (Edwards, 2001). This form of pattern matching keeps the mechanism quite simple. It also means that code need not be deserialized to allow for custom compare functions (Venners, 2000).

If a client wishes to conduct a more advanced search, they need to define a ServiceTemplate that will match the broadest category of services they are interested in and implement an advanced search on that set. This scenario does not represent a good separation of concerns. Clients should be free to perform the tasks they are assigned without having to manage specialized service discovery code that could involve a great deal of calculation to find the correct service, such as a client searching for the Solution Engine for a Constraint Satisfaction Problem (Tsang). This specialized code belongs in a service of its own and for use by clients in need of the same.

INTELLIGENT SEARCH SERVICE FRAMEWORK

An intelligent search service should be able to conduct a comparative search upon the terms handed to it. For example, the service could be designed to accept a predicate logic expression (Hughes, Barbeau & Bordeleau, 2001) or predicate object that could be evaluated to form a satisfier set (Hughes et al., 2001) for the client to choose from.

Constraint Satisfaction as a Search

When a lookup service is given a ServiceTemplate, it attempts service discovery on behalf of the client by searching a domain for a solution. The domain of the search is the list stored by the directory. The solution is a satisfier set — a set of references that have satisfied the search criteria. This suggests that searching for a service could be described as a Constraint Satisfaction Problem (CSP). A CSP is any problem that can (Crescenzi, 2000):

a. be defined by
   - a set of variables: $V = \{v_1, v_2, \ldots v_m\}$,
   - a set of domains that define what values each variable can take: $D = \{d_1, d_2, \ldots d_m\}$,
   - a set of constraints defining all relationships between variables: $C = \{c_1, c_2, \ldots c_n\}$;
b. be solved by determining a set of variable-value pairs satisfying all constraints (a solution).

The domains may not be continuous; they may not even be numeric. This depends on the expressive power of the problem description language. The solution is the set of values for each variable where all constraints are satisfied. A solution set may consist of:

- just one solution, arbitrarily chosen;
- all possible solutions;
- an optimal solution given some objective function that maps to a variable which can be minimized or maximized;
- an empty set, if no solution can be found.

Computing a solution to a CSP means finding a set of value assignments for each variable such that all constraints are satisfied, and any objective function is achieved. The basic process is iterative; instantiate variables and check if all constraints are satisfied. If they are, you have a solution. If they are not, instantiate a different set of values and try again until either a solution is found, or it can be concluded that there

is no solution. Decisions are made by the heuristics of an algorithm. These decisions include: what variable is to be instantiated next or what constraint should be evaluated next.

Intelligent Attributes Measure and Describe

How can Jini, as a service directory, provide for the requirements of a more advanced search service? It can by providing a standard method of service discovery that can be expanded to include attributes that measure a service and, thus, be used in an intelligent search service that uses a wide variety of search techniques, such as constraint satisfaction through ILOG — a high level mathematical solver.

Attributes of a service are by definition descriptive. The standard set of attributes mentioned in the section on Entry attributes contains descriptive string information about a service, but they are only used in exact pattern matching. Comparative pattern matching could use descriptive measures of a service. Any measurable quality can be compared with the inequality operators, < and > as well as the equality operator, =, and can be used to build expressions, such as constraints and an objective.

Potential candidates for intelligent attributes include:

- size of the service proxy to be downloaded;
- bandwidth of the service provider;
- expected minimum, maximum and median processing time needed to run service (or maybe a formula for the calculation of expected processing time);
- queue length for service;
- cost of the service.

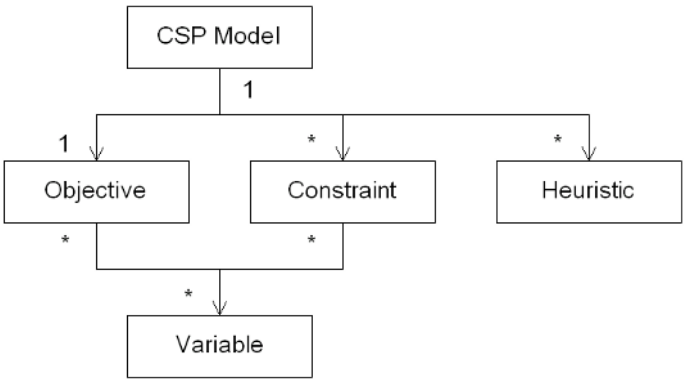
Finding a solution to a CSP means finding a set of value assignments for each variable such that all constraints are satisfied, and the objective function is achieved. From this description of a CSP provided above, any search implemented as a CSP must model variables, domains, constraints, objectives and heuristics.

Search criteria in the Entry array are variables; object data members of the ServiceTemplate. Java primitive types have a primary domain automatically specified by type, and Java object types may use code to specify a more complex domain via the inclusion of a method that outputs the next value in the domain's sequence. A collection of Java primitive and object types can, thus, form a set of variables with domains.

Constraint Satisfaction Problem — Object Model

In Java, new data types can be written to provide a framework that can be used in many situations. A constraint type will represent a relation between variables and have a method to return Boolean true, if the constraint is satisfied; false otherwise. An objective type may also represent a relation between variables, defining the

Figure 2. Constraint Satisfaction Problem Object Model



expression that must be minimized or maximized. A CSP type should also exist, maintaining references to the constraints, objective and any other supporting classes, such as heuristics and expression objects. An example of such a model is shown in Figure 2. Importantly, shows that variables may participate in multiple relationships, which is needed in CSPs.

A single CSP model should contain any number of constraint and heuristic objects and only one objective. Variable objects are implicitly part of the model through the objective and constraints, but the model must have a way to access the set of all variables.

Heuristics should be able to control the order in which constraints are evaluated. If each constraint is modeled as an individual object, the set of all constraints may be stored in an array and ordered to suit. The processing of a set of heuristics in between iterations of a constraint problem-solving exercise involves modifying the order in which variables are assigned values and constraints are evaluated. The collections of variables and constraints can be stored in their own arrays, made available to a Java object encapsulating heuristic logic, which can order the variables and constraints as it sees fit, and ready for the next iteration.

The following is a specification of the methods required for a variable object that also acts as an Entry object. This specification deals with Java double values but could be implemented as integer values, if needed. The public double data item allows the variable to be matched by Jini's exact pattern matching.

```
public interface Variable implements net.jini.core.Entry
{
   // Public object wrapped by implementing class
   public Double value;
   // Get name of this variable
```

```
    public String getName();
    // Change state to first value in domain. Return new value.
    public double first();
    // Change state to previous value in domain. Return new value.
    public double previous();
    // Get current value
    public double currentValue();
    // Change state to next value in domain. Return new value.
    public double next();
    // Change state to last value in domain. Return new value.
    public double last();
}
```

A similar format can be applied to another class for constant values.

```
public interface Constant implements net.jini.core.Entry
{
    // Public object wrapped by implementing class
    public Double value;
    // Get name of this variable
    public String getName();
    // Get value
    public double getValue();
}
```

The specification for constraints, objective and heuristics are quite open. They do not need to inherit the Entry interface, because they will not be searched for; only variables need be searched for.

```
interface Constraint
{
    public boolean satisfied();
}

interface Objective
{
    public double value();
}


interface Heuristic
{
```

```
    public void processHeuristics
        (Variable [] attributes, Constraint [] constraints);
}
```

Expressions

Constraints and objective functions represent relationships between variables. There must be control over what sort of relations are accepted. Variables and constants should be added in some ordered way to an expression object capable of enforcing an accepted grammar. Many CSP-solving engines are capable of solving linear problems. A linear expression class could be used as a base with extensions available to cover more advanced relationships, if a solving engine can work with them.

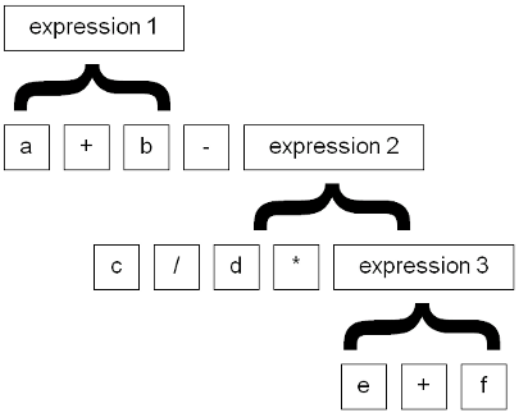
Characteristics of a Linear Integer Expression

A linear expression could be built, using the basic arithmetic operators, and allow for bracketed expressions to be "nested expressions." This grammar could allow division by constant values, but not of variables, to keep the degree of the expression as one. Such an expression has:

- **Width:** the number of operands in the first level of the expression, where operands may be expressions or nonexpressions;
- **Maximum width:** the maximum number of operands in the first level of an expression;
- **Depth:** the number of levels of nested expressions;
- **Maximum depth:** the maximum number of levels of nested expressions. This limits the number of expressions that may be used as operands;
- **Population:** the number of nonexpression operands in the entire expression and all child expressions;
- **Maximum population:** the maximum number of nonexpression operands in the entire expression and all child expressions;
- **Expression population:** the number of expression operands in all levels of the expression;
- **Base operand:** the base operand is the first operand in the expression. It is special because no operator may be applied to it, although an implicit addition operator may be assumed;
- **Solution:** any expression can be solved for its current set of values. An expression is not an equality or inequality and, thus, does not consist of $=, \leq$ or $\geq$. The solution to an expression as defined by this grammar might also be noninteger, since it is possible to divide by constants.

The example expression shown in Figure 3 will serve to illustrate these characteristics.

Figure 3. An Example Expression



Assume these values for demonstration purposes: a = 1, b = 2, c = 3, d = 4, e = 5 and f = 6. Table 2 shows the characteristics of the expression shown in with these values.

The expression object needs to make sure a number of rules are enforced, such as nonduplication of an expression within itself, nondivision by a variable as well as the limits of width, depth and population.

The revised object model would be as shown in Figure 4.

Each expression object may only be part of one constraint or objective, but variables may be shared between any of them. An implementation detail is how to model expressions and variables. A binary or n-ary tree may be the most suitable.

PROCESS DESCRIPTION

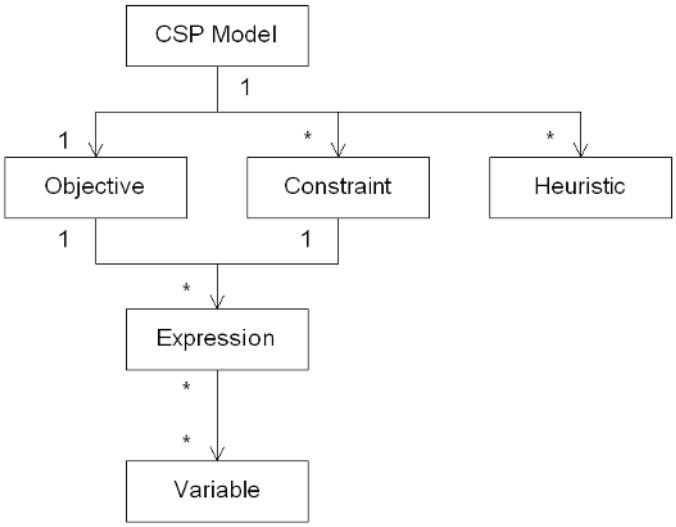
This section shall provide a process overview at a relatively high level of detail, describing the chronological order of events between entities in the system modeled by this work.

The discovery and registration stages are shown in Figure 5.

Table 2. Expression Characteristics

Expression	*Base Operand*	*Width*	*Depth*	*Population*	*Expression Population*	*Solution*
expression 1	a	3	3	6	2	-5.25
expression 2	c	3	2	4	1	8.25
expression 3	e	2	1	2	0	11

Figure 4. CSP Object Model



One

All service providers must discover the Jini Service Directory and then register their capability descriptions with it.

Two

The intelligent search service (ISS) must discover the Jini Service Directory and register its capability description with it.

Three

The client must discover the Jini Service Directory and perform a search to find the ISS.

Four

The client will receive a satisfier set from the Jini lookup service. It will either contain reference to an ISS or contain nothing. If it does contain reference to an ISS, the client will move on the next step, shown in Figure 6.

Five

The client will initiate a second search, this time on the ISS. The client actually calls a method on the ISS proxy it received from the Jini lookup. This proxy object uses RMI to forward the request to a remote service object on the ISS machine that also implements the proxy's interface. The proxy will do nothing more than return to the client whatever the remote object returns to it.

Figure 5. Discovery and Registration

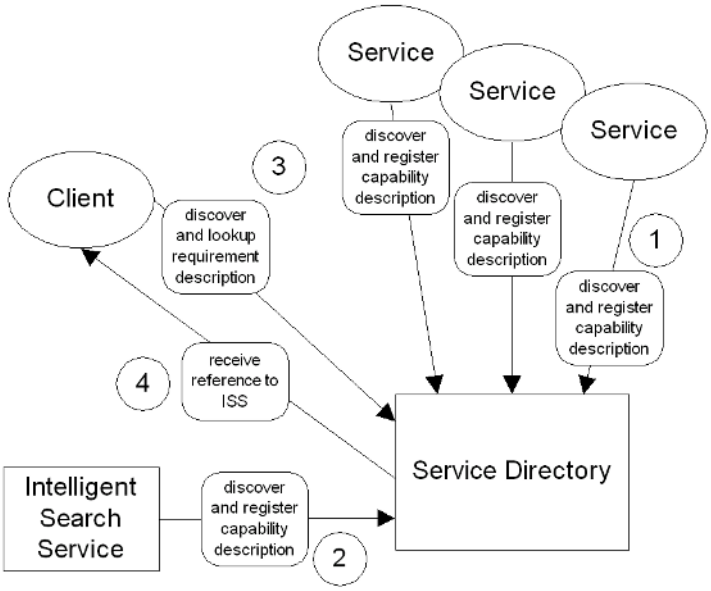
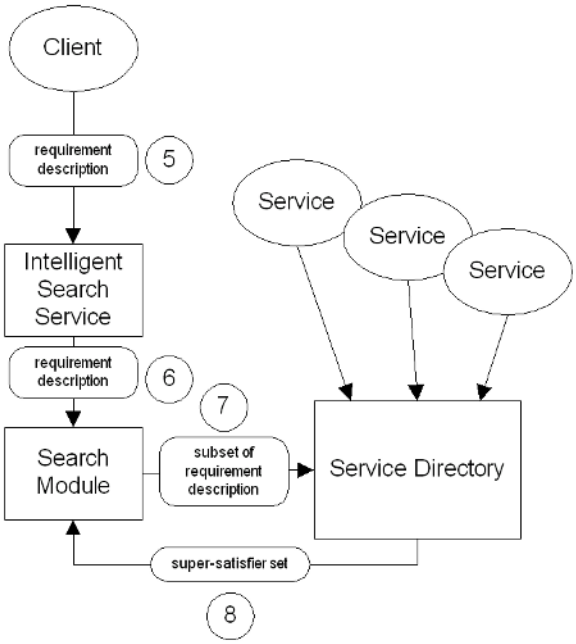


Figure 6. Client — ISS Search Process



A decomposition of this process is shown in Figure 7.

The client actually sends the requirement description to the local proxy object it received from the Jini lookup's ServiceRegistrar object. The proxy forwards the requirement description to the remote service object, which examines it. Upon finding a scenario object, it forwards the requirement description to the module capable of dealing with the given scenario.

Six

The ISS, having received the requirement description from a client, will examine that description to decide which search module to activate. If a capable module is found, it will be activated and passed the client's requirement description.

Seven

The search module processes the requirement descriptions and will use a subset of the client's requirement description to form a new requirement description. The new requirement description is sent to the Jini lookup as the broadest possible search criteria, forming a superset of all services that have the correct attributes.

Eight

The ISS will receive a satisfier superset from the Jini lookup service. It will either contain references to a number of service providers or contain nothing. If it does contain references to services, the ISS will have a superset of capability descriptions for all services that have the required attributes. The ISS will then move on the next step, shown in Figure 8.

Figure 7. Decomposition of Client to ISS Requirement Description

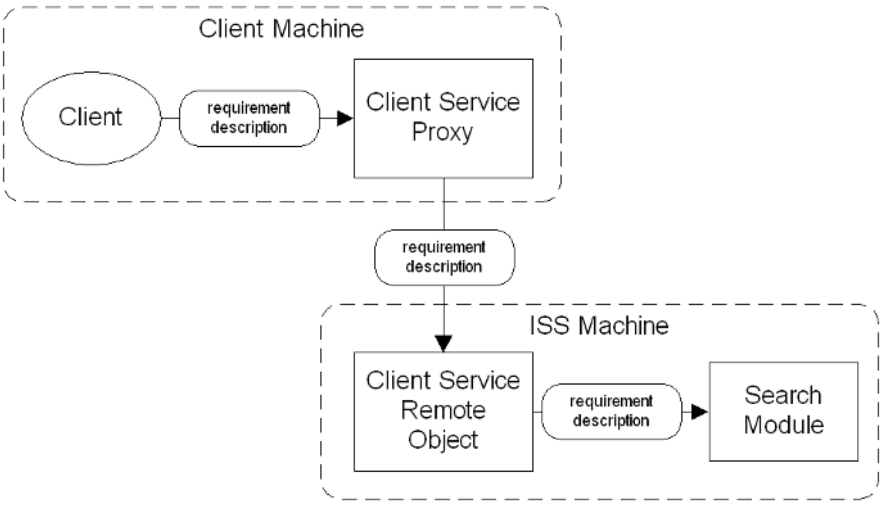
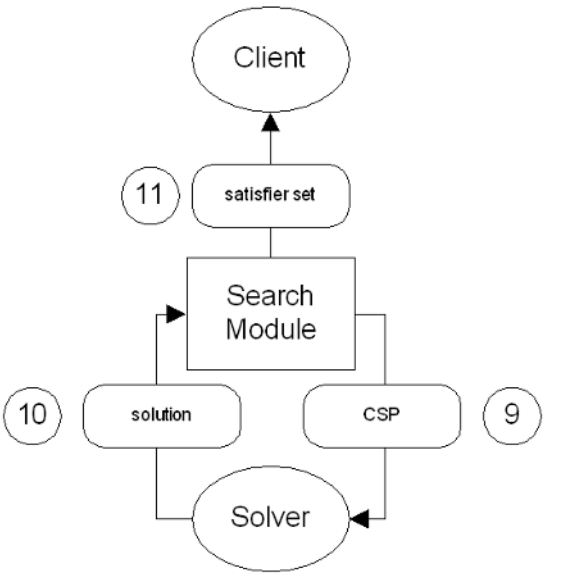


Figure 8. Solve and Return Process



Nine

The ISS now computes a CSP model. This model will be independent of any particular solver but will consist of the elements detailed in the framework. This ISS search module will now choose a solver-specific module, one specialized for a particular solving engine. The general CSP model will be sent to it and shall convert the generic CSP model, shown in Figure 8, into the same thing but specialized for the solving engine.

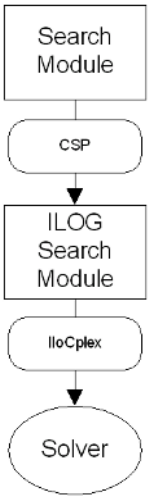
A decomposition of this process is shown in Figure 9, where ILOG is assumed to be the solving engine of choice.

The CSP created by the Search Module is not specific to any particular solver, meaning that the solving service could be brokered to any solving engine capable of doing the job. The price of this independence is a measure of double handling. The generic model must be specialized to a solver at some point. This process shows that an ILOG-specific search module is responsible for this job. It turns the model into an IloCplex object, which is solved by the ILOG solver.

Ten

The specific solving engine will return a solution to the search module. The search module must map this solution onto the superset of references obtained in step eight.

Figure 9. Decomposition of ILOG Solving Process



Eleven

If the solver return shows that no result has been found, a null satisfier set shall be built. Otherwise a satisfier set shall be built consisting of those services the solver found to match the client's requirement description.

ServiceTemplate for Simple and Complex Searches

The search service will receive a ServiceTemplate and should be able to discriminate between attributes to be used in the search for a superset of services and those that should not. Two arrays of variables could be sent or some could be marked in some other way.

A Generic Framework of Service Discovery

The application will provide a search service for other Jini services, but there is no reason why it cannot be designed to facilitate a client connecting directly to the search service, without the knowledge of Jini.

The most important aspect is to ensure that a communication protocol is in place, using either XML or Java objects, to contain the service template and results. Wrappers could be written for RMI or SOAP, with the wrapping class incorporating the appropriate transmission protocol, such as sockets.

CONCLUSION AND FUTURE WORK

This paper represents the current state of research. A framework of intelligent service discovery has been outlined using a generic constraint satisfaction problem-

solving architecture that allows different algorithms to be used as library components. A prototype has been built to these specifications and is being tested.

The addition of ISS would undoubtedly increase the complexity of an already complex notion. The benefits of such a search service are that clients in a large system would have the power to select the service that is right for them based on attributes that describe measurable aspects of the service, rather than just describe the service (i.e., the basic Jini attributes act more as keywords for a service). This power to discriminate is useful in any scenario, where there are a large number of services with similar attributes. An example is a market place driven by competition for client patronage or a pool of solvers that all solve CSPs but with different specialist algorithms.

The client must be sure such a service is worth their while. A search service would undoubtedly increase the time it takes to link the client to a service, so it must be proportionately important to the client to find the right service.

In addition to increasing the client's time to connect to a search service, there is the possibility that bottlenecks will hamper the performance of any central directory service. However, ISS splits the work of service discovery into two tiers: simple discovery and complex discovery. The search service, being specialized, will not receive lookup requests that Jini's native lookup service can handle.

Another load balancing issue has to do with dynamic attributes, and the fact that service discovery is not an atomic action. The time delay in using a search service could mean that a critical variable, used to make a decision based on its current value, could have quite a different value when the client finally connects to the service. This warning applies to any dynamic attribute. One method of dealing with dynamic or volatile attributes, whose values change often, is to include a field rating of the volatility of an attribute. Highly volatile attributes might be considered to have a lower priority than nonvolatile attributes. These functions can be relegated to the reasoning implemented by a Heuristic object.

Backtracking and constraint propagation are the two main categories of algorithms used to solve CSPs (Crescenzi, 2000). An important question that needs to be answered by this investigation is: will the framework outlined in this paper be capable of allowing a wide variety of established CSP algorithms to run?

Future work will involve further completion of the test bed and development of an extended suite of standard classes to allow a variety of search algorithms to be used "out of the box," without the client having to get their hands dirty, except for filling in a few values. The ISS will be evaluated according to how successfully it discovers the best services for a client as defined by the collection of CSP objects.

REFERENCES

Bowstreet. (2001). *Directory services markup language*. Retrieved October 3, 2001, from http://www.dsml.org/about.html.

Crescenzi, P. (2000). *On the hamming distance of constraint satisfaction problems*. Retrieved from the Università di Firenze Web site: http://gdn.dsi.unifi.it/~rossig/Papers/TCS1/tcs.html.

Edwards, K. (2001). *Core Jini* (2nd ed.). Upper Saddle River, NJ: Prentice Hall PTR.

Hughes, E., Barbeau, M., & Bordeleau, F. (2001, June). *Service recommendation using SLP (Service Location Protocol)*. Paper presented at the meeting of the IEEE International Conference on Telecommunications (ICT), Bucharest, Hungary.

McGrath, E. R. (2000, April 5). *Discovery and its discontents: Discovery protocols for ubiquitous computing*. Retrieved October 1, 2001, from http://www.ncsa.uiuc.edu/People/mcgrath/Discovery/dp.html.

Newmarch, J. (2001a, June 12). *Jan Newmarch's guide to JINI technologies, v 2.08*. Retrieved May 20, 2002, from http://jan.netcomp.monash.edu.au/java/jini/tutorial/Jini.xml.

Newmarch, J. (2001b, September). *A Survey of some recent Internet services*. Retrieved April 7, 2002, from: http://jan.netcomp.monash.edu.au/services/presentation.html.

Pascoe, R. (2000). *Dynamic networking requires comprehensive service discovery*. Retrieved October 1, 2001, from: http://www.serverworldmagazine.com/hpchronicle/2000/10/discovery.shtml.

Tsang, E. (1996). *Mapping constraint satisfaction problems to algorithms and Heuristics.* Department of Computer Science, University of Essex.

Venners, B. (2000, February). *Finding services with the Jini lookup service—discover the power and limitations of the ServiceRegistrar interface*. Retrieved from the JavaWorld, a division of Web Publishing, Inc., Web site: http://www.javaworld.com/javaworld/jw-02-2000/jw-02-jiniology.html.

Webopaedia.com. (2000, May 30). *Definition of directory*. [Webpage] Retrieved April 7, 2002, from the Internet.com Web site: http://www.webopaedia.com/TERM/d/directory.html.

Chapter XVI

A Service-Based Approach to Components for Effective Business-IT Alignment

Zoran Stojanovic
Delft University of Technology, The Netherlands

Ajantha Dahanayake
Delft University of Technology, The Netherlands

ABSTRACT

Although Component-Based Development (CBD) platforms and technologies, such as CORBA, COM+/.NET and Enterprise Java Beans (EJB), are now de facto standards for implementation and deployment of complex enterprise distributed systems, the full benefit of the component way of thinking has not yet been gained. Current CBD approaches and methods treat components mainly as binary-code implementation packages or as larger grained business objects in system analysis and design. Little attention has been paid to the potential of the component way of thinking in filling the gap between business and information technology (IT) issues. This chapter proposes a service-based approach to the component concept representing the point of convergence of business and technology concerns. The approach defines components as the main building blocks of business-driven service-based system architecture that provides effective business-IT alignment.

INTRODUCTION

The main challenges enterprises face today are how to manage complexity of systems being developed, effectively utilize the power of the Internet and be able to rapidly adapt to changes in both technology and business. The new paradigm of CBD has been introduced as an excellent solution for building complex Internet-enabled enterprise information systems (Szyperski, 1998; Brown & Wallnau, 1998). The basic idea of CBD originates from the strategy successfully applied in other engineering disciplines that a system developed from components is more flexible and easier to develop. CBD provides higher productivity in system development through reusability, more effective system maintenance, higher quality of solutions and the possibility for parallel work. Moreover, it provides better system adaptability through replaceability of parts, localization and better control of changes, system scalability and the possibility of using legacy assets.

The CBD paradigm has often been presented as a new silver bullet for complex, enterprise-scale system development in the Internet age (Udell, 1994). However CBD inherits many concepts and ideas from the earlier encapsulation and modularization — "divide-and-conquer" initiatives in information technology (IT). The NATO Conference in 1968 recognized that producing software systems should be treated as an engineering discipline providing system assembling from software components (McIlroy, 1968). Parnas (1972) defines concepts and requirements for decomposing system into modules. These principles of separation of concerns, encapsulation and plug-and-play building blocks have been applied in different ways through the concepts of functions, subroutines, modules, units, packages, subsystems, objects and now components.

The CBD paradigm was been first introduced at the level of implementation and deployment. CBD middleware technologies, such as CORBA Components (Siegel, 2000), Enterprise Java Beans (Sun Microsystems, 2002), and COM+/.NET (Microsoft, 2002), are now used as standards for the development of complex enterprise-distributed systems. While the technology solutions are necessary in building the system, one cannot simply program and deploy components using a component middleware, without any prior plan to follow from business requirements towards implementation. For the effective use of the CBD paradigm and in order to gain real benefits of it, the component way of thinking must be applied in earlier phases of the development lifecycle, such as system analysis and design. CBD methods and approaches proposed so far do not provide a complete and consistent support for various component concepts. Components are often treated as implementation concepts — packages of binary or source code that can be deployed over the network nodes. During the system analysis and design, components, if used, are often represented as larger grained business objects. This suggests using components mainly at the system implementation and deployment as software code packages, while still following the principles of object-oriented (OO) modeling, analysis and design.

At the same time, the role and usefulness of the component concept as a bridge between business and technology issues have not been truly recognized. Components can be identified very early in the system lifecycle, namely derived from business requirements, and then used as central artefacts of the whole development process. In this way, the whole process would be structured and organized around the same set of component concepts. That can provide a necessary alignment and traceability from business services to implementation assets. The benefit from the separation of concerns using components will be gained at all levels of system development.

This chapter proposes a service-based approach to components that provides a consistent and integrated support for the model-driven system development, using components as the central concept. The approach provides that the same component way of thinking and the same consistent set of technology-independent component concepts can be effectively applied in different aspects and phases of enterprise systems development, from autonomous business services to distributed software components. In this way, the service-based component concept represents the point of integration of business and technology concerns, the common ground between them and the effective way for business-IT alignment. The service-based approach proposed here can be equally applied in modeling and specifying the system developed to use Web services over the Internet.

In the remainder of the chapter, the state-of-the-art of CBD is presented. Various approaches to components and component-based design and development proposed so far are outlined. In describing the most relevant CBD approaches, we mainly focus on the way they define, identify and handle components. In the next section, we propose a new view on the component concept, defining important component properties and elements, as well as different component granularity levels. In the sequel, we present the way of identifying components and briefly illustrate it using the example of the Internet-based travel agency. In the following section of the chapter, we propose the way of specifying single business components as well as complete business-driven service-based component architecture of the system. This architecture can serve as a point of negotiation between different actors in the component-based development process.

THE STATE-OF-THE-ART OF COMPONENT-BASED DEVELOPMENT

Component technologies are now widely used in the development of complex Internet-enabled systems. First, VBX controls, DCOM/COM, CORBA and Java Beans, and now COM+/.NET (Microsoft, 2002), CORBA Components (Siegel, 2000) and Enterprise Java Beans (EJB) (Sun Microsystems, 2002) represent the standard component-based implementation solutions. Based on them, component-

based frameworks and architectures have been developed to speed-up the application development, such as IBM San Francisco framework based on EJB (Carey, Carlson & Graser, 2000) and the TINA architecture for building telecommunication systems based on CORBA (TINA, 2002).

The physical perspective on components as binary packages of software is still predominant. The standard Unified Modeling Language (UML) treats components as packages of binary code and uses them in describing system implementation through component and deployment diagrams (Booch, Rumbaugh & Jacobson, 1999). Components in UML represent physical things that can be deployed over network nodes. The Catalysis approach defines a component as a package of software code as well as other software artefacts (D'Souza & Wills, 1999). In Szyperski (1998), a software component is a unit of composition with contractually specified interfaces and explicit context dependencies. A software component can be deployed independently and is subject to composition by third parties. Gartner Group (1997) defines a runtime software component as a dynamically bindable package of one or more programs managed as a unit and accessed through documented interfaces that can be discovered at runtime.

Definition of a business-oriented component concept can be found in Herzum and Sims (2000), where a business component is the software implementation of an autonomous business concept or business process, and in Accenture (1998), where a business component is a means for modeling real-world concepts in the business domain. When introducing components, questions about similarities and differences between objects and components naturally arise. In Udell (1994), components represent a new silver bullet for system development in the Internet age, while objects have failed to provide higher level of reusability. In the UML, components are nothing other than larger grained objects deployed on the network nodes. In Szyperski (1998), a component comes to life through objects and, therefore, it would normally contain one or more classes, as well as traditional procedures and even global variables. In a debate over this topic (Henderson-Sellers, Szyperski, Taivalsaari & Wills, 1999), granularity has been seen as the main issue in distinguishing components and objects. By Catalysis, components are often larger grained than traditional objects and can be implemented as multiple objects of different classes. Components can use persistent storage, while objects typically work only within the main memory.

Academia and industry have just started to recognize the importance of new CBD methods, processes, techniques and guidelines. The methods and approaches are often greatly influenced by the OO concepts, constructs and principles, dictated by the use of the standard UML. The Rational Unified Process (RUP) (Jacobson, Booch & Rumbaugh, 1999), Catalysis and the Select Perspective (Allen & Frost, 1998) can be considered as the first generation of the CBD methods (Stojanovic, Dahanayake & Sol, 2001a). RUP does not specifically target component-based development. Rather it offers a general framework for OO design and construction. Components are implementation units that package other software artefacts and are

deployed over network nodes. The Select Perspective was originally an OO method with the later addition of component - modeling principles. The Select defines a component as a service or implementation packages. The method is not that sophisticated; rather it combines best-of-breed techniques and tools. The Catalysis approach originates from several OO methods and approaches, with the component concepts added afterwards. Therefore, object and component concepts are interleaved in Catalysis to some extent. The Catalysis provides remarkable CBD support and a lot of valuable concepts and principles for understanding components but not a systematic roadmap and ease of use.

The Business Component Factory (BCF) (Herzum & Sims, 2000), the UML Components approach (Cheesman & Daniels, 2000) and the KobrA approach (Atkinson et al., 2001) represent the second generation of the CBD methods. These methods are more focused on components concepts than previous ones. They provide a comprehensive support to CBD throughout the system lifecycle and represent remarkable achievements in the field. On the other hand, there are certain shortcomings. The BCF approach defines business components as representation of autonomous business concepts and business processes in the domain. By separating entities and behavior, this approach does not provide a uniform view on components. On the other hand, the role and importance of service-based interfaces are diminished. The UML components approach does not take into account potentially different levels of component granularity and importance of using the separation of concerns in defining them. The approach proposes mainly a data-driven way of component identifying that does not fit well into the advanced service-driven computing initiatives. The KobrA method describes components by using UML class and object diagrams without extensions. A KobrA component has, at the same time, properties of a class and a package. On the other hand, the role of component interface is not emphasized enough. The composition of components is defined mainly through containment trees, instead of collaboration between component interfaces.

One of the most important activities in practicing component-oriented development is how to start with components, i.e., how to identify components and place them properly in the development lifecycle. The way of identifying components is closely related to how components are defined and treated. In Booch et al. (1999), components are identified during the implementation and deployment phases as a way of packaging and deploying software code artefacts. In the Business Component Factory, business components are identified as important business concepts that are relatively autonomous in the problem space. In the UML Components approach, components are identified through the core business types. These core types result in business component interfaces that manage instances of those types. In addition system interfaces interacting with those components are derived from use cases. In RUP components are represented as subsystems in component-based design but without further details about their identification. There is no strict prescription for identifying components in Catalysis. The emphasis in Catalysis is on component

collaboration in the form of the framework, and components are represented through the type analysis. The KobrA approach does not offer strict rules about how to identify components (Atkinson et al., 2001). Rather the approach treats important business domain concepts as components and follows OO analysis and design on them. In Jain, Chalimeda, Ivaturi and Reddy (2001), an algorithmic approach is proposed for treating components as containers of classes. It uses a clustering algorithm on the domain model that represents significant domain classes, together with the sets of managerial and technical goals in component-based system design.

A NEW VIEW ON COMPONENTS

The use of components as implementation artefacts is already well established in the enterprise system development practice. Software systems are built from components in easier and more effective way. Components can be built in-house from scratch or by wrapping existing legacy assets. Furthermore, components can be bought as Commercial Off-The-Shelf (COTS) components from the component marketplace, such as ComponentSource (ComponentSource, 2002), or invoked over the Internet in the form of Web services (IBM, 2002).

At the same time little attention has been paid to applying the component concepts and component way of thinking in earlier phases of the system lifecycle, i.e., system analysis and design. In our opinion, the component concept becomes more useful when used as an architectural-level artefact to model the logical architecture of the technical or business/domain infrastructures. In this chapter, we are interested in how components can help in more effective building of flexible, business-driven system architecture. After the complete distributed system architecture is precisely specified in the component-oriented manner, we can decide on concrete realization of specified components in one of the ways mentioned above. In this way the precise system specification is a durable result than can be implemented afterwards in different ways, using different algorithms, platforms and technologies. This strategy is now the mainstream of the Object Management Group's (OMG) Model-Driven Architecture (MDA) (OMG-MDA, 2002). MDA proposes first designing a Platform Independent Model (PIM), using modeling standards, such as the UML, Meta Object Facility (MOF) and Common Warehouse Metamodel (CWM), then specifying the Platform Specific Model (PSM) of the system using, for example, the UML Profiles for EJB or CORBA, and finally implement the system in the target middleware technology.

Our main goal is to propose an approach to defining business-driven, truly component-oriented PIM that will rapidly respond to changes in the business environment and, at the same time, will be easily transferable to code and middleware. The component concept is the focus and the main artefact of this PIM. In this way, the component architecture specified by the PIM can provide a bridge between business and implementation issues.

Basic Component Concepts and Definitions

For years system developers and integrators have struggled to translate and interpret business requirements into system implementation that fulfill business goals, delineate business structure and provide efficient development of adaptable solutions. In our opinion, component-based development can represent a link between different perspectives on the system, making a common ground for business and technical concerns. Regardless of the context of usage, the essence of the component-based approach is the explicit separation between the outside and the inside of the concepts being addressed. This means that only the question what is considered (what useful services are provided by the particular building block to the context of its existence), not the how (how these services are actually implemented). Since components can be of different forms — granularity and nature — any attempt to provide a single, general definition of a component covering all its possible aspects may be insufficient. Examining essential properties of a component can be more appropriate.

In order to use the consistent component way of thinking at different levels of abstraction, from business to system distribution, we propose a general, implementation-independent concept of the component. In our opinion, a component defined mainly as an autonomous provider of necessary services to its environment represents the point of integration of business and system concerns (Stojanovic & Dahanayake, 2002). The concept of services is equally useful and precise in both business and technical terms. Instead of providing a "one-size-fits-all" definition of a component, we will define a set of essential component concepts and properties uniformly applicable for all component types and variants:

- A component represents a self-contained concept in the context of its existence. A component is a part of the context but as independent as possible from the rest. It must be possible to precisely define what a component can offer (promise to do) to its context, under what constraints and rules and what kind of support a component can expect from the context in order to behave properly.
- A component represents an autonomous concept but does not exist in isolation. It always participates in a composition with other components to form a higher-level component. At the same time every component can be represented as a composition of lower level components. This recursive composition is an important property of the component concept.
- A component is an encapsulated unit with a completely hidden interior behind the interface. The interface provides an explicit separation between the outside and the inside of the component, answering the question what but not how. The component interface, precisely defined in a contract-based manner, allows for the use of the component services without knowing and taking care how they are actually realized.

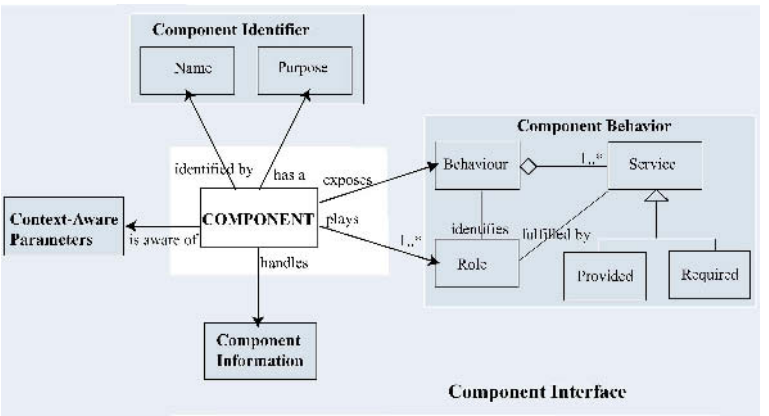
In order to use the component, it is necessary to know its interface. All the component shows to its context is its interface. The interface is usually represented as a set of operations that the component provides and requires (Siegel, 2000). Here we propose a wider definition of the interface representing all the necessary information about the component that its context should know about and rely on in order to collaborate with the component properly. We divide the component interface into the following parts (Figure 1):

Component identifier: A component is identified in the context by its unique name (or identification) in the naming space and its purpose. The purpose of a component is the intended effect of the component in the context or an aim of its existence. The purpose of a component determines component behavior in order to reach that goal.

Component behavior: A component is a behavioral unit. It is mainly a provider of services. It plays certain role(s) in the given context. According to its roles, a component exposes certain behavior through providing and requiring services to/from its environment according to certain constraints and conditions. A component often needs some services, called required services, from other components in order to behave properly according to its contract. The behavior of component services is specified using preconditions, postconditions and guarantees.

Component information: A component must handle, use, create or simply be aware of certain information in order to provide its services properly. The interface must define what types of information are of interest to the component, as well as constraints and rules on them. This does not necessarily mean that the component owns or contains that information; it only defines what information the component needs for exposing proper behavior. In its simplest form, that information can be considered as input/output parameters of component services. The behavior of defined information is specified using invariants — conditions that must always be true.

Figure 1. Component and its Interface

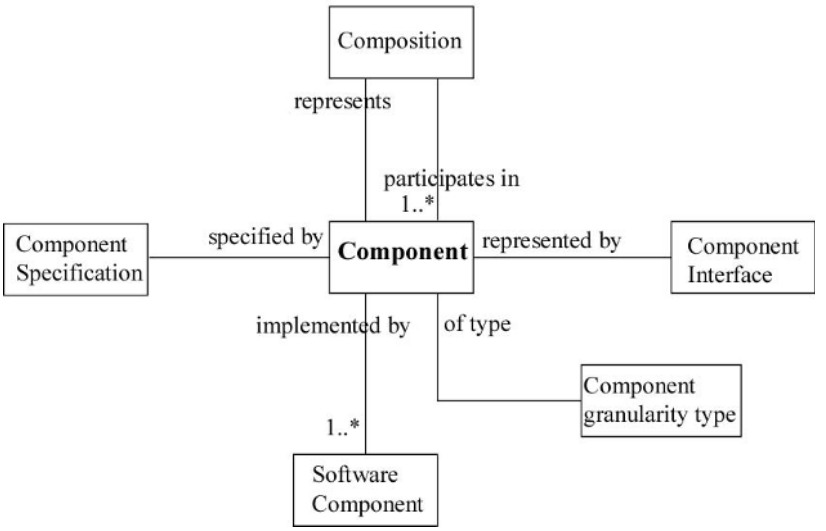


Context-aware parameters: A component is often dependent on the context of its existence. Beside the services a component needs from other components, in order to provide flexible services and adapt to certain changes in the context, the component interface can define some context-aware parameters, be able to reason about them and adapt its services to varying requirements. These parameters can be, for example: different required Quality-of-Service (QoS), necessity for synchronization with other services in the context, different profiles of component consumers, as well as different component locations in time and space. By defining these parameters, the component is able to "survive" ever-changing requirements that come from the context and provide appropriate services in a dynamic way.

Context-aware parameters can influence preconditions, postconditions and invariants on services and information types, respectively, by relaxing or modifying their criteria. The reason for introducing context-aware parameters into the specification of the component interface is that interfaces capable of supporting many different levels of services are more flexible than interfaces that only support a single strict service level. Components capable of providing services at many different levels can be reused in many different contexts. Systems built using such interfaces and components are inherently more adaptable.

Defined elements of the component interface are mutually related in the following manner. The name and the purpose of the component denotes its roles and behavior, i.e., what services the component should provide to fulfill its mission in the context of existence. Services, as a part of component behavior, use component information as parameters. Invariants, static and dynamic conditions defined by information, based on the business rules, put particular constraints on the component

Figure 2. Metamodel of the Component Concepts



behavior. Preconditions and postconditions defined by services cross-reference information types. Context-aware parameters adapt both component information and component behavior according to the changing requirements of the context.

All defined segments of the component interface must be fully and precisely defined using concepts of the contract-based theory (Meyer, 1997). Such contractually defined interface actually forms a contract between a component (provider of services) and components (consumers of those services). Precise and formal specification of the component and its interface is used to properly assemble and integrate the component into an overall solution for a given problem, to substitute the component with a compatible component, if necessary, to browse the component catalog in order to find a component that matches the needed specification or to reuse the component in a different context. A fully specified component at the level of specification architecture is normally implemented by one or many software components at the implementation level (e.g., CORBA, COM+ components or EJB). A metamodel of the main component concepts is presented in Figure 2.

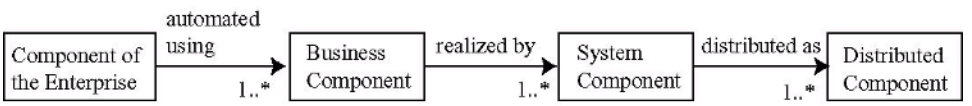
Component Granularity

In the enterprise system development, depending on the level of abstraction (from business to system distribution), several levels of component granularity can be defined (Figure 3). All of them follow the general set of component concepts and properties defined above.

At the highest level of abstraction, the whole enterprise can be considered as a collaboration of components. Components of the enterprise are business actors (people, systems, departments, companies, etc.) in the enterprise that have goals and responsibilities, fulfill roles, provide services, handle necessary information, and work in collaboration to achieve the common business goals. All of these actors actually have a component-like structure. Their interior is hidden and is not important as long as they provide contractually specified services to others, according to their roles in the enterprise. One of the components of the enterprise is the IT system being developed, which normally represents a composition of lower level components that we call business components.

Business Components: Business components are defined by considering business processes and tasks for which automated system support is required. They provide business and/or business-oriented technical services for the business processes in which the system participates, i.e., they add perceived and measurable value to the business actors interacting with the system. At this level, it is not

Figure 3. Different Levels of Component Granularity



important how these services of the system are implemented as long as they, in collaboration, provide expected behavior of the system in the enterprise. A business component represents a composition of system components.

System Components: Opposite of business components, system components provide lower grained services, which have more technical than business meaning and therefore do not add meaningful business value to the actors around the system. Several system components in collaboration actually provide services defined by business components. A system component can be represented by one or by the composition of many distributed components.

Distributed Components: Distributed components are actually distribution-aware system components. These components in collaboration provide necessary business and technical services, but, at the same time, depending on their roles in collaboration, they can be placed on particular tiers of the multitier system architecture. In this way, some distributed components support users in working with the system; some components are responsible for managing permanent data storage, while others represent the realization of the business logic.

The concept of components of the enterprise can be used in enterprise modeling to provide better adaptability and management of the complexity at the business process level, but it is out of scope of this chapter. The concepts of business components, system components and distributed components represent fundamental specification elements of the system at different levels of abstraction and will be further elaborated in the sequel of the chapter. Depending on the scale of the problem being solved, inside each of these basic levels of component granularity particular sublevels can be defined, for example, higher grained and lower grained business components or higher and lower grained system components. Defined specification components as building blocks of the system architecture are normally implemented using one or many software components (COM+/.NET components, CORBA components or EJBs). A software component is a unit of software that is independently developed, deployed and maintained.

Component Modeling and Specification Notation

Regarding the notation for modeling components, the natural approach is the use of the standard UML as a standard OO-modeling notation. The UML provides component notation only at the physical level through implementation and deployment diagrams and does not provide a support for various component concepts at the logical level. Therefore, the component at more conceptual level can be represented as a UML package, the <<subsystem>> stereotype of the package, or a particular stereotypes of a class, for example, <<business component>>. For representing the interface, the standard lollipop notation can be used as well as the extended version through the <<interface>> stereotype of a class. Information used and handled by the component can be represented using particular stereotypes of a class, such as <<business concept>> or <<information type>>. Collaboration among components

can be represented using sequence or collaboration diagrams. We hope that the next major version of the UML 2.0 will provide a more comprehensive support to various component concepts.

For more formal, contract-based specification of components of different granularity, collaboration and complete system architecture, a dedicated Component Specification Language (CSL) can be defined. In order represent sufficient all component concepts presented above, the CSL should represent a superset of the following:

- Interface Definition Language (IDL) (Siegel, 2000) for specifying services and functions;
- Object Constraint Language (OCL) (Warmer & Kleppe, 1999) for specifying constraints on services and information;
- Object Definition Language (ODL) (Cattell et al., 2000) for specifying information types or XML Schema Definition (W3C, 2002);
- A specification constructs for defining context-aware component parameters and rules for adapting component behavior based on them.

IDENTIFYING AND SPECIFYING COMPONENTS

One of the crucial decisions in practicing component-based development is how to start the process of using components, i.e., how to identify components and define them through different granularity levels. The way and time of identifying components depend a lot on the way of defining and treating components. In order to provide smooth transition of business requirements into the system solution, first-cut business components should be defined as early as possible in the development lifecycle. This strategy ensures that the whole development process is structured in a component-oriented manner and organized around the same set of component concepts.

Use Case Driven Component Identifying

Components by our definition are primarily behavioral, service-based building blocks, providing useful services to its environment. Considering the system in the given business context, the main task is to define and encapsulate the behavior of the system that supports business processes (or their parts) in which the system participates. Therefore, the right place for starting with components is, in our opinion, use case analysis. While moving from the use case model to the class/object system model is not a straightforward and easy process (Dobing & Parsons, 2000; Cockburn, 2001; Jacobson, Christerson, Jonsson & Overgaard, 1992), the use case model represents the right starting point for identifying components. Use case analysis should help in defining the place of the system inside the business domain, actors using the system as well as business and technical services realized by the

system. The participation of the system in existing business processes is defined through one or more use cases that the system provides.

Our basic idea is to identify and define components as building blocks of the system responsible for the realization of particular use cases. In this way our component concept corresponds to the concept of collaboration as defined by the UML (Booch et al., 1999). The further consideration of semantic relationships between the component concept defined here and the UML collaboration will be presented in another paper. Following the given assumption, components should provide services to fulfill business and/or technical goals defined by use cases. The following important characteristics of use cases justify our choice of using use cases for identifying components. Use cases capture the intended behavior of the system without having to specify how that behavior is implemented (Jacobson et al., 1992). Use cases are technology independent; they are not tied to any technology and any methodology including object-orientation.

For the purpose of identifying components we use a goal-oriented approach to use case analysis proposed by Cockburn (2001). In this approach, a use case is described in terms of a goal-oriented sequence of business or work steps. Both the goals and the interactions in a use case scenario can be unfolded into finer and finer grained goals and interactions. Cockburn (2001) defines three levels of use cases as three named goal levels:

- User goal is the goal the primary actor has in trying to get work done or the one the user has in using the system, e.g., Place Order, Create Invoice, Pay Bill, etc.
- Summary-level goals involve multiple user goals. They represent higher level goals that show the context in which the user goals operate, e.g., Order, Advertise, Rent, etc.
- Subfunction-level goals are those that carry out user goals. They represent lower level goals that participate in fulfilling user goals, e.g., Find a Product, Compose a Query, Save a Change, and Browse a Catalog, etc.

Furthermore, we use white-box use case description that, besides the activities directly visible to the external actor, includes some details of the behavior of the system in responding to the actor's activities and requests. For example, the white-box description of the "customer withdraw money" use case includes the following: the account is accessed to determine the balance, the money is debited from the account, the report about the transaction details is issued, and so on. We also adopt some principles of the robustness analysis, as defined by Jacobson et al. (1992). Robustness analysis is used to identify a first-guess set of objects and classify them into boundary objects (which actors use in communicating with the system), entity objects (usually objects from the domain model) and control objects (which embody much of the application logic).

Based on the goal-oriented use case analysis, the white-box representation of use cases and the robustness analysis, we propose an algorithm for identifying

components at different levels of granularity. The following algorithm steps are defined:

- Based on defined primary-goals use cases, high-level business components are identified as parts of the system that support the realization of these primary goals.
- Business components, as the main building blocks of the business-driven component architecture, are identified as service providers responsible for the realization of user-goals use cases.
- In defining a business component through a given use case we can define several types of component services responsible for realizing different aspects of the use case:
  - services that support the user in interacting with the system while performing the use case;
  - services that perform different computing as a part of the business logic related to the use case;
  - services that manage the information handled in different ways during performance of the use case.

  These services are actually responsible for realizing the steps of the user-goal use case and can be mapped to system components.
- Realization of the steps of "user goal" use cases is under the responsibility of system components. Some system components support interactions of the user with the system, some of them perform particular processing, while some of them provide managing of data and data transactions.
- After taking distribution settings into account, we can place particular system components in different tiers of a distributed multitier architecture and define distributed components accordingly. In this way, we have components attached to the user-tier, components providing business logic, and components responsible for managing data and permanent storage.
- If one use case extends the other use case, the component responsible for that extension can be considered a subcomponent of the component realizing the main use case. The "extension" component is obviously lower grained than the main one, and it belongs completely to the main component.
- If one use case includes the other use case, this can be considered as collaboration between components, where the first component uses services of the other one. Usually, the used component is lower grained and exists independently, since it can be used by many higher grained components of the system. That component can be considered as a "subroutine" of some larger grained components that need its services.
- If one use case is a generalization/specialization of the other use case, this denotes that the first use case represents a generalization/specialization of the later one.

This approach for identifying components is actually top down, starting from business requirements and use cases. Instead of treating components as static data containers, i.e., business entity objects that offer CRUD operations, we define components basically as service providers. The approach preserves the unity of data and behavior but in a slightly different way and at the higher level of abstraction. In identifying classes/objects in object orientation, first an important domain entity is identified and represented by a class (object) through the list of attributes, and then operations for managing these attributes are defined as methods of the class. By our component approach, a component is identified as a provider of services that support particular business needs. Then, the information needed by these services in order to perform their tasks properly is specified.

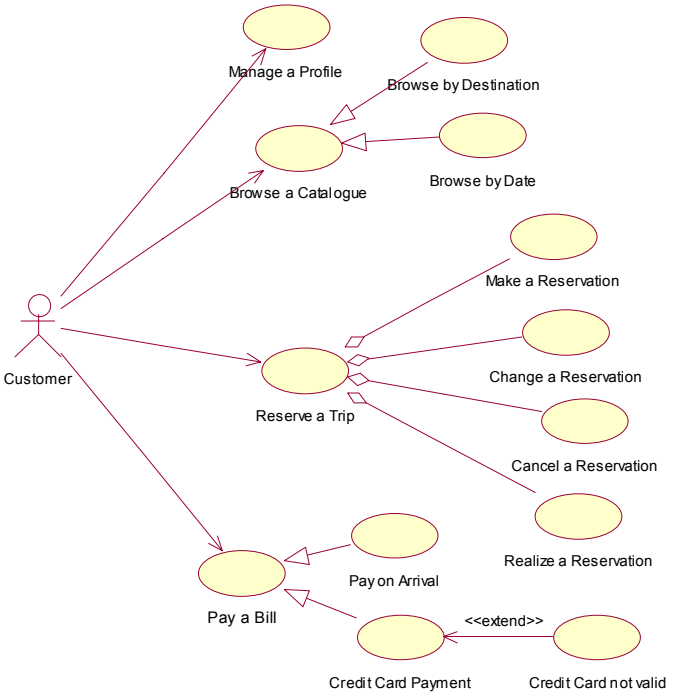
At the conceptual level, it is only important what information types a component should be aware of, assuming that instances of these types will be available to the component during the run time. The decision whether the component should own the instances of particular information types and in what way the permanent storage should be organized, is related to the detailed system design. Generally, if the component is supposed to create, update or delete information type instances, it should own that information type, or better said it is responsible for managing all aspects of the information type. If the component only reads the information type instances, then it should only "know" how to find and use that information, when necessary. An additional solution is the introduction of the Data Manager component that is responsible for delivering information type instances to components when they need it. These are already an implementation decision and out of scope of the chapter.

This approach to components actually closes the known semantic gap between use cases as requirement analysis tool on one side, and objects and classes in OO system analysis, design and implementation, on the other side (Dobing & Parsons, 2000; Meyer, 1997). Components are now directly identified by use-case analysis; and, after components and their collaboration at all granularity levels are specified, objects and classes can be used for their interior realization. The approach defines the component concept at the higher level of abstraction than the concept of object in object orientation and places components between use cases and classes/objects.

Travel Agency Example

The approach for identifying and defining components of different granularity will be illustrated in this section by the example of the Internet-based travel agency. The agency offers travel arrangements through its Web site. The customer can create his profile and further manage it using the facilities of the Web site. The customer can browse the catalog of trip offerings, examine detailed information about them, check their availability and prices and choose what he wants. The customer then can make a reservation for the chosen trip. He can afterwards change the reservation or cancel it, if necessary. The customer can realize the reservation

Figure 4. The Use Case Diagram of the System



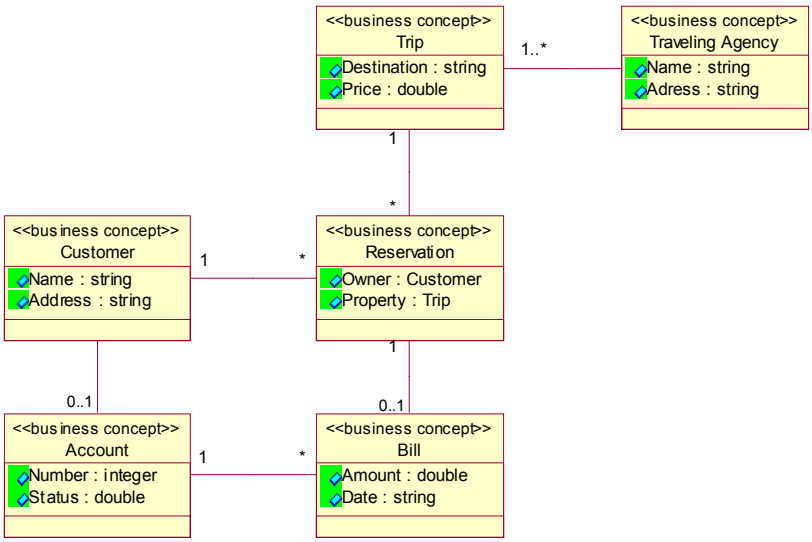
by paying it with the credit card via the Web site, or, on arrival, in cash. Based on the given problem description and the typical scenario of system usage, the use-case diagram presented in Figure 4 can be defined.

The conceptual information model of the system is shown in Figure 5. This model contains the main information types that represent the concepts of the business domain that should be handled by the system. Each information type has the list of attributes and constraints and invariants defined on them.

Following the algorithm for identifying components based on the use-case analysis presented above, we can define the first-cut business components of the system. These components are responsible for supporting the realization of the main use cases of the system. They are:

- **Profile Manager:** to control log in and password information, enable creating and modifying user profiles, and ensure security;
- **Catalog Browser:** to provide browsing and querying of trip catalog information based on different criteria;
- **Reservation Manager:** to manage activities relating to trip reservations, from making a reservation to confirming a reservation;
- **Payment Manager:** to support the customer payment.

Figure 5. Conceptual Information Model

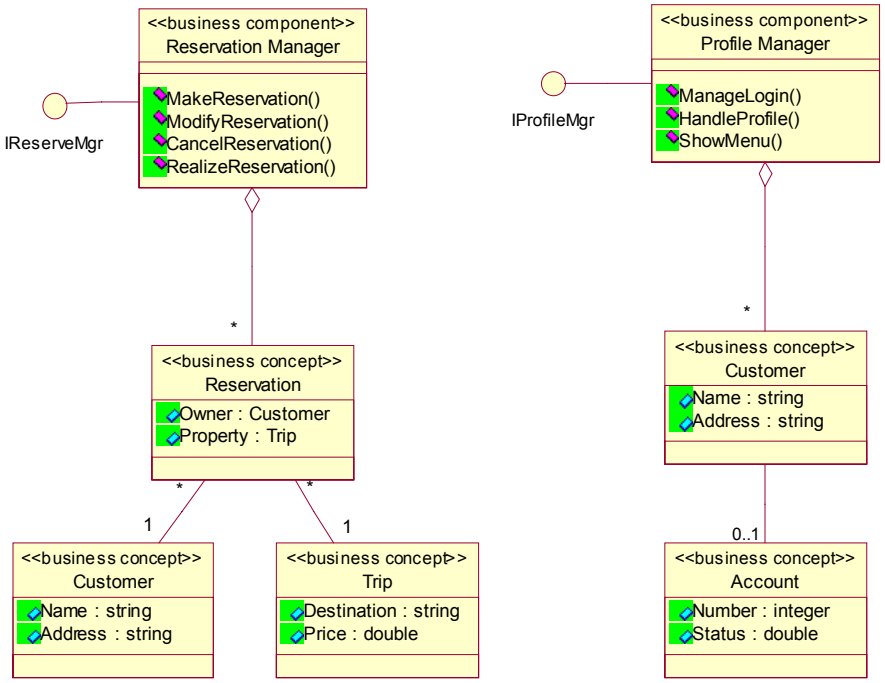


Some of these components can be further specialized into more concrete components. For example, Catalog Browser can be specialized into the Browser by Destination component and the Browser by Date component, based on the given selection criteria. Similarly, the Payment Manager component can be further specialized into the Payment by Credit Card component and the Payment on Arrival component depending on the type of payment. On the other hand, the component Reservation Manager represents an aggregation of lower level components responsible for different activities related to the process of the reservation.

Business services defined for each business component now cross-reference the given information model to decide what information types are needed by a particular component to provide its services, and how on these services are defined using information types. By following given business rules, services that each component should offer are precisely specified using preconditions and postconditions and defined constraints on information types. These services correspond to steps in use cases that were previously used for identifying higher level business components. Each of the steps by Cockburn has its own lower level goal, and the goals of system components are to provide realization of these lower level use-case goals. The provisional specification of two components of the system, Reservation Manager and Profile Manager, together with information types they should handle, is shown in Figure 6.

Services of business components are actually realized by particular system components (or, in the case, of a large-scale application with several component recursive levels, by lower level business components). Again, the main characteristic

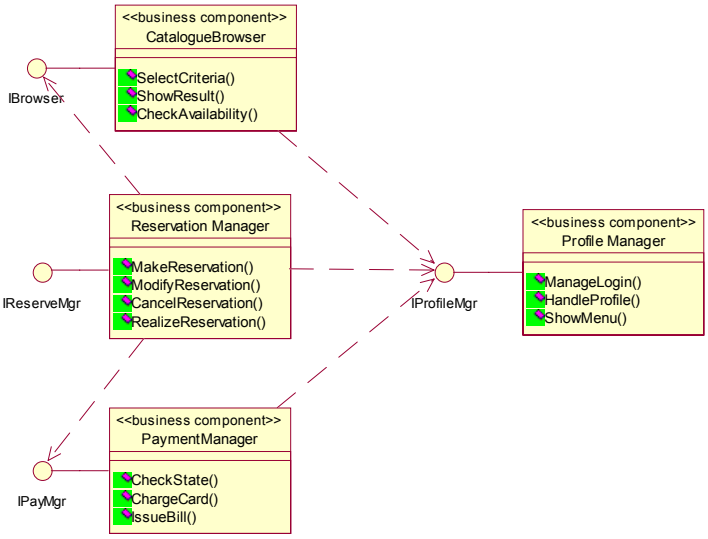
Figure 6. Specification of the Reservation Manager and Profile Manager Components



of a business component that distinguishes it from a system component is that a business component provides some meaningful services, i.e., it adds some value to the user of the system. In the case of the Reservation Manager component, we can define the following lower level business components: Reservation Maker, Reservation Canceller, Reservation Modifier and Reservation Realization Manager. Furthermore, the Reservation Maker component can be represented as collaboration of the following system components: Reservation Composer, Reservation Processor and Result Presenter. Similarly, in the case of the component Profile Manager, we can define the following system components that belong to it: Login & Password Manager, Profile Handler and Menu Manager. System components use or are simply aware of the same types of information (or their subset) as business components they belong to.

When system distribution and multitier system architecture are taken into account, defined system components can be transferred to corresponding distributed components. Some system components are related to the user interface (UI) tier, some to the business logic tier and some to the data storage tier. For example, Reservation Composer and Result Presenter are attached to the UI tier; Reservation Processor is attached to the business logic tier, and Reservation Storage Handler is placed in the data storage tier.

Figure 7. Initial Component-Oriented System Architecture



Initial component-oriented architecture of the Internet travel agency system representing the main business components of the system and dependencies between them is shown in Figure 7. Further steps in the approach include precise specification of all business, system and distributed components using the approach that will be described in the sequel of the chapter. The complete system architecture specification can then be realized using particular component implementation.

BUSINESS-DRIVEN SERVICE-BASED COMPONENT ARCHITECTURE

In this section we will focus on the way to provide complete component-oriented specification for the system architecture, based on the given business requirements. Enterprise distributed systems being developed are potentially very complex and demanding. That raises the need for using some form of separation of concerns in specifying system architecture as an effective general strategy for managing the problem of complexity.

Separation of Concerns

In order to manage complexity and ensure completeness in the specification of components and component-oriented system architecture, we use the ISO standard Reference Model of Open Distributed Processing (RM-ODP) as an underlying idea (ODP, 1996). RM-ODP defines a framework for specifying architectures for distribution, interoperability and portability of applications based on OO technology.

It is widely recognized as offering the most complete and internally consistent specification framework. The RM-ODP specification of a system consists of five different specifications, corresponding to five separate, but related and consistent viewpoints: enterprise, information, computational, engineering and technology.

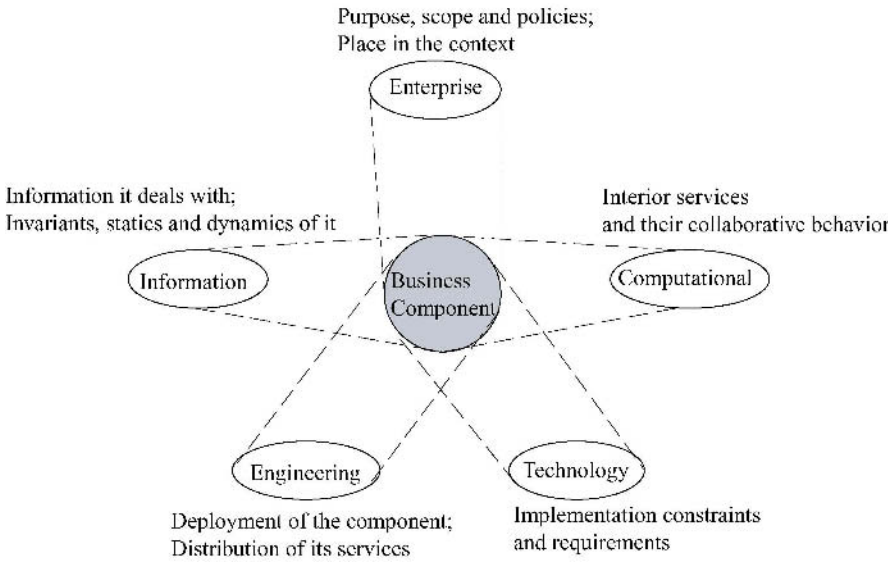
We use the concepts and principles of the RM-ODP specification framework at two different levels:

1. for complete specification of a single business component (possible, but less important for a system component);
2. for complete specification of the whole component architecture.

The single business component is specified through the five viewpoints in the following way (Figure 8).

- The enterprise viewpoint specification of the business component defines purpose, scope and policies of the component in the context of its existence. The exact place of the component in the context is defined as the component's goal, the way it communicates with other components towards a common goal, as well as the set of rules, constraints and policies applicable to the component.
- The information viewpoint specification of the business component defines types, constraints and semantics of information handled by the component, as well as the possible ways of information processing. The information specification of the business component is a model and dynamics of the information that the business component holds, uses or remembers.

Figure 8. Specification of the Business Component Using the RM-ODP Viewpoints



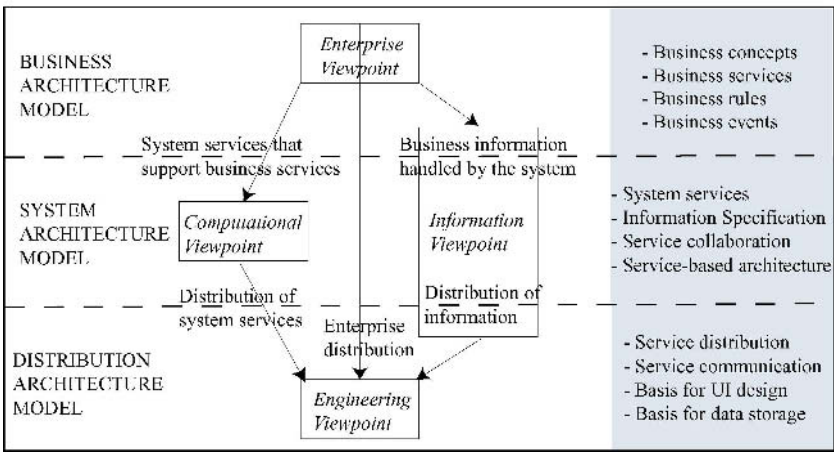
- The computational viewpoint specification of the business component specifies the services of the component and how they collaborate to produce a cohesive set of functionality provided by that component. Based on this, subcomponents of the business component and their collaboration are specified as well as what system services are available through the interface of the business component and what services are used for internal collaboration.
- The engineering viewpoint specification of the business component specifies in details how services of the business component (or better said its subcomponents) are placed on the network tiers, when distribution aspects are taken into account.
- The technology viewpoint specification of the business components defines some technology constraints and requirements, regarding the future implementation of the business component.

At the next level, we use the RM-ODP as an underlying framework for the complete specification of component-oriented system architecture. Based on ODP viewpoints, we propose the following three architectural models:

- Business Architecture Model (BAM);
- System Architecture Model (SAM);
- Distribution Architecture Model (DAM).

These different, but consistent and correlated architectural models, provide us a way of separating of concerns through different architectural views (Stojanovic, Dahanayake & Sol, 2000; Stojanovic, Dahanayake & Sol, 2001b). We use the RM-ODP viewpoints as a basis for defining our architectural models. The relation between our architectural models and the RM-ODP viewpoints as well as the main

Figure 9. Proposed Architecture Models and RM-ODP Viewpoints



relations between the models are shown in Figure 9. Our main idea is to incorporate the component concept into each of the RM-ODP viewpoints (Stojanovic et al., 2000). In this way we can define component-oriented viewpoints on the system, completely focused on the component concept and organized around it, as powerful means for specifying component architecture. The same consistent set of component concepts used by different viewpoints represents the point of consistency and integration between them (Stojanovic et al., 2001b).

Proposed models are defined as following:

BAM specifies the behavior of the system in the context of the business for which it is implemented. It represents business context and processes in which the system should participate, services that must be offered by the system to fulfill business purposes, business domain concepts and information that must be handled by the system, and business rules that must be satisfied and supported by the system. Defined services are the basis for identification of higher level business components as we have seen in the previous section. Each business component, as well as their collaboration, must be specified in the business architecture model, through the common set of component properties. Business domain conceptual information should be defined and cross-referenced by the specification of business services offered by components. This shows how conceptual information types are used and handled by particular business components in providing services. Each business component is typically a collaboration of several system components. This means that identified business components are used as an entry for defining business-driven system architecture and correspondent system components.

SAM defines the structure of the system in terms of configurations of system services and the interactions among them to provide necessary business services related to the business requirements on the system. This model defines information that should be handled by the system including rules for information dynamics, and information processing that should be performed by the system. SAM takes as an entry the higher level components identified in the previous model. This model further specifies system components that in collaboration realize required business services. Components of different granularity are defined and fully specified. This model also specifies information managed by the system, i.e., by each of its components. Collaboration of components inside the system architecture is specified as a potential for future distribution of components. System components provide lower grained services that in collaboration actually realize business services. Each system service component actually represents one, or a collaboration of several, distributed components.

DAM specifies a distributed infrastructure of the system in terms of the distribution of system services, their allocation and communication over the tiers in the n-tier architecture. This model defines in what way information is distributed over the network nodes, as a basis for future data storage and user-interface design. It describes the distribution of the enterprise, for example, so-called virtual enterprises.

DAM uses collaboration of components inside the specified architecture as a starting point, and uses particular distributed settings to precisely define complete component-based multitier system architecture. Distributed components are actually system components in the distribution context. Typically, business and system components can have user interface, business logic and permanent storage facets (one, two or all three of them), so that they can be represented as a collaboration of their subcomponents (i.e., distributed components) attached to different architecture tiers.

All three models are consistent and mutually integrated, providing a complete business-driven system architecture specification. The specification of the models should not be performed in a simple sequential order, rather in incremental and iterative way. This approach will ensure completeness and consistency of the models. The traceability between the architectural models and the three types of components are shown in Figure 10. The specification process takes as an entry the business (process) modeling (the top of Figure 10) and ends in the specification of the complete component-oriented system architecture (bottom-right corner of Figure 10). This specification actually represents a component-oriented PIM that can be mapped to a PSM and realized using particular component implementation technology.

Negotiation Architecture

Defined component architecture actually represents a point of negotiation between different actors in component-based system development (Figure 11). It provides a clear connection to the business processes and business domain that are

Figure 10. Types of Components and Traceability Between Them

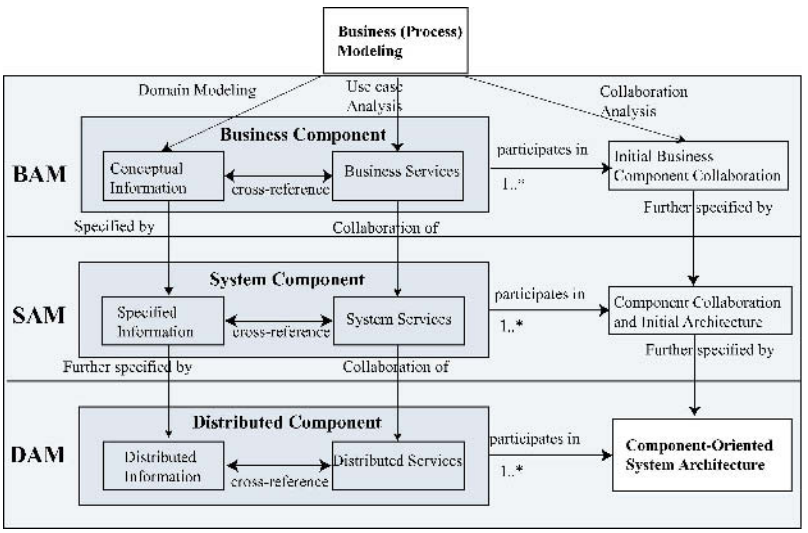
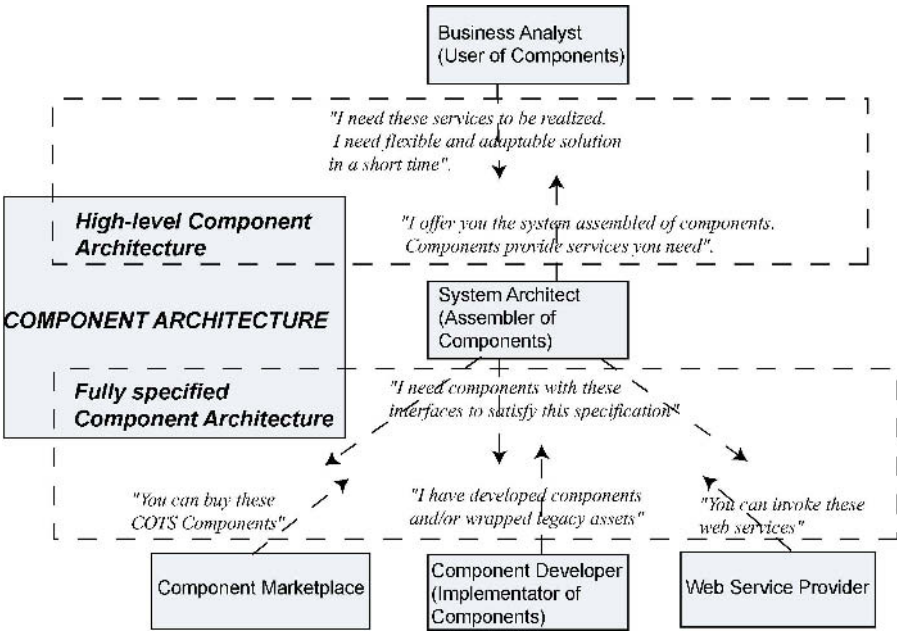


Figure 11. A Role of Component Architecture in a Dialogue Between Actor Roles



the basis for the requirements of the system being developed. At the same time, this architecture provides an easy and effective mapping into appropriate implementation architecture of the system.

The higher level component architecture, not overloaded with unnecessary details, can be used for the purpose of negotiation between the business user as the user of the component system and the system architect as the assembler of components. At this level, it should be decided how business requirements should be mapped into the system solution in the best and most effective manner. It is possible to exactly determine what system parts are responsible for satisfying particular business needs, and, therefore, be able to replace/improve/change those parts if the business is changed. Business users can easily recognize and articulate their needs through the business component architecture that hides implementation details from them.

On the other hand, the completely specified component architecture can serve as a point of negotiation between a system architect on one side, and a component developer, a provider of COTS components or a provider of Web services, on the other side. At this level, the decision about an appropriate implementation solutions for components specified in the architecture should be made. The way of concrete realization of the component specification is not important as long as the contract between the component and its context is completely fulfilled.

CONCLUSION AND FUTURE TRENDS

Although component-based platforms and technologies such as CORBA, COM+/.NET and EJB are widely used as a standard for implementation and deployment of complex systems, the component way of thinking is not mature enough. Current CBD best practices, approaches and methods do not provide a full support for various component concepts, and therefore, are not able to provide a full benefit of the CBD paradigm. Handling components mainly at the implementation level and treating them as binary-code packages actually limits the usefulness of the same separation of concerns in requirements elicitation, system analysis and design. At the conceptual level, components are mainly represented as larger grained objects that are more data driven than service driven. The real power of using the service-based component way of thinking as a bridge between business and technology concerns has not been recognized yet.

The main goal of the chapter is to define a service-based approach to components that provides a comprehensive support to a model-driven development of Internet-enabled enterprise systems, from business to implementation. The approach applies the same component way of thinking and the same consistent set of technology-independent component concepts in different aspects and phases of enterprise systems development, from autonomous business services to distributed components. Defined service-based component concepts provide greater ability to model business services and requirements at a higher level, in a domain-specific, but implementation-independent, way. On the other hand, the application developers map business-driven system models into complete applications using advanced component middleware.

In this way, component architecture is flexible enough to be easily adapted according to frequent changes in the business. On the other hand, the architecture is minimally affected by the underlying technology choice, providing a durable solution that can survive the changes in technology. Furthermore, the presented component architecture represents a point of negotiation between different actors in a development process: the user, assembler and provider of component services.

The approach presented in this chapter is in line with the current OMG strategy in establishing the MDA that suggests first creating a high-level UML description of how applications will be structured and integrated, independently of any implementation details (PIM), then moving toward more constrained UML design (PSM), and finally converting it into language code for a specific platform. Our service-based component approach actually aims at fully specified component-oriented PIM that can be easily mapped first to component-oriented PSM and then to a particular component middleware.

By defining components as providers of services, our approach can be effectively applied for modeling, analysis and design of systems, using the new paradigm of Web services (IBM, 2002; Microsoft, 2002). Web services are self-contained, self-describing modular units providing a location independent of business

or technical services that can be published, located and invoked across the Web. From a technical perspective, of course, the Web service is essentially an extended and enhanced component interface construct. They are a natural extension of component thinking and further convergence of business and technology. By using the approach presented here, Web services become just one way of possible realization of service-based component specification. In this way, the approach provides a smooth transition from the standard object orientation- to the advanced Web-service way of thinking.

Finally, our approach is in line with the principles of new Agile Modeling and Development initiatives (Ambler & Jeffries, 2002; Cockburn, 2002). The approach is agile, if it is effective, flexible, lightweight, iterative and responds to changes. The approach presented here can be considered as a service-based agile approach for designing component architecture, since it provides an effective way for specifying components and component-based solutions that can be easily adapted to changes in the business environment.

ACKNOWLEDGMENTS

The authors would like to thank Professor Dr. Henk Sol for his support and valuable suggestions in writing this chapter. The authors also gratefully acknowledges the support of the BETADE research project (http://www.betade.tudelft.nl) and its members.

REFERENCES

Accenture (formerly Andersen Consulting Co.). (1998, September 20). *Understanding components* (White paper, Eagle project). Retrieved September 20, 1998 from http://www.ac.com.

Allen, P., & Frost, S. (1998). *Component-based development for enterprise systems: Applying the select perspective.* Cambridge, UK: Cambridge University Press.

Ambler, S. W., & Jeffries, R. (2002). *Agile modeling: Effective practices for extreme programming and the unified process.* New York: John Wiley & Sons.

Atkinson, C., et al. (2001). *Component-based product line engineering with UML.* Boston, MA: Addison-Wesley Publishing.

Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *The unified modeling language user guide.* Boston, MA: Addison-Wesley.

Brown, A. W., & Wallnau, K. C. (1998, September/October). The current state of component-based software engineering. *IEEE Software, 15*(5), 37-47.

Carey, J., Carlson, B., & Graser, T. (2000). *San Francisco design patterns: Blueprints for business software.* Boston, MA: Addison-Wesley.

Cattell, R., et al. (2000). *The object data standard: ODMG 3.0*. San Francisco, CA: Morgan Kaufmann.

Cheesman, J., & Daniels, J. (2000). *UML components: A simple process for specifying component-based software.* Boston, MA: Addison-Wesley.

Cockburn, A. (2001). *Writing effective use cases*. Boston, MA: Addison-Wesley.

Cockburn, A. (2002). *Agile software development.* Boston, MA: Addison-Wesley.

ComponentSource. (2002, July 12). Retrieved July 17, 2002 from http://www.componentsource.com.

D'Souza, D. F., & Wills, A. C. (1999). *Objects, components, and frameworks with UML: The catalysis approach*. Addison-Wesley.

Dobing, B., & Parsons, J. (2000). Understanding the role of use cases in UML: A review and research agenda. *Journal of Database Management, 11*(4), 28-36.

Gartner Group. (1997, December 5). *Componentware: categorization and cataloging* (Research note). Retrieved December 5, 2001 from Applications Development and Management Strategies Web site: http://www.gartnergroup.com.

Henderson-Sellers, B., Szyperski, C., Taivalsaari, A., & Wills, A. (1999). *Are components objects?* OOPSLA'99 - Panel Discussion.

Herzum, P., & Sims, O. (2000). *Business component factory: A comprehensive overview of business component development for the enterprise*. John Wiley & Sons.

IBM. (2002, July 17). *Web services*. Retrieved July 17, 2002 from http://www.ibm/com/webservices.

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The unified software development process*. Reading, MA: Addison-Wesley.

Jacobson, I., Christerson, M., Jonsson, P., & Overgaard, G. (1992). *Object-oriented software engineering: A use case driven approach.* Reading, MA: Addison-Wesley.

Jain, H., Chalimeda, N., Ivaturi, N., & Reddy, B. (2001). Business component identification - a formal approach. *Fifth IEEE International Enterprise Distributed Object Computing Conference (EDOC 2001)* (pp. 183-187) Los Alamitos, CA.

McIlroy, M. D. (1968, October). Mass produced software components. *Report NATO Conference on Software Engineering*.

Meyer, B. (1997). *Object-oriented software construction*. Upper Saddle River, NJ: Prentice Hall.

Microsoft. (2002, July 17). *COM technologies and .NET Web service platform*. Retrieved July 17, 2002 from http://www.microsoft.com/.

Object Management Group (OMG). (2002, July 17). *Object Management Group, the source for UML, CWM, and MOF.* Retrieved from http://www.omg.org.

Object Management Group-Model Driven Architecture (OMG_MDA). (2002, July

17). *Object Management Group, MDA-Model Driven Architecture.* Retrieved July 17, 2002 from http://www.omg.org/mda/.

ODP. (1996). International Standard Organization (ISO). *Reference model of open distributed processing: Overview, foundation, architecture and architecture semantics* (Rep. no. ISO/IEC JTC1/SC07. 10746-1 to 4. ITU-T; recommendations X.901 to 904).

Parnas, D.L. (1972). On the criteria to be used in decomposing systems into modules. *Communication of the ACM, 15*(12), 1053-1058.

Siegel, J. (2000). *CORBA 3: Fundamentals and programming*. OMG Press, John Wiley & Sons.

Sol, H. G. (1988). Information system development: A problem solving approach. *Proceedings of 1988 INTEC Symposium Systems Analysis and Design: A Research Strategy,* Atlanta, GA.

Stojanovic, Z., & Dahanayake, A. N. W. (2002). A new approach to components. *IRMA 2002 International Conference*, Seattle, WA, USA.

Stojanovic, Z., Dahanayake, A. N. W., & Sol, H. G. (2000). *Integrated component-based framework for effective and flexible telematics application development* (Tech. Rep. ISBN: 90-76412-13-8). The Netherlands: Delft University of Technology.

Stojanovic, Z., Dahanayake, A. N. W. & Sol, H. G. (2001a). A methodology framework for component-based system development support. *Sixth Workshop on Evaluation of Modeling Methods in System Analysis and Design (EMMSAD'01*), *Interlaken, Switzerland* (pp. XIX-1 – XIX-14).

Stojanovic, Z., Dahanayake, A. N. W., & Sol, H. G. (2001b). Integration of component-based development concepts and RM-ODP viewpoints. *First Workshop on Open Distributed Processing WOODPECKER 2001, Setubal, Portugal* (pp. 98-109).

Sun Microsystems. (2002, July 17). *Enterprise JavaBeans; The source for Java™ technology.* Retrieved from http://java.sun.com.

Szyperski, C. (1998). *Component software: Beyond object-oriented programming*. ACM Press, Addison-Wesley.

Telecommunications Information Networking Architecture (TINA)(2002, July 17). *Service Architecture 4.0.* Retrieved from TINA Consortium Web site: http://www.tinac.com.

Udell, J. (1994, May). Cover story: Componentware. *Byte Magazine*.

Warmer, J. B., & Kleppe, A. G. (1999). *The object constraint language: Precise modeling with UML.* Reading, MA: Addison-Wesley.

World-Wide-Web Consortium (W3C). (2002, July 17). *eXtensible markup language (XML)*. Retrieved from http://www.w3c.org/xml.

Chapter XVII

One Method for Design of Narrowband Lowpass Filters

Gordana Jovanovic-Dolecek
National Institute of Astrophysics Optics and Electronics (INAOE), Mexico

Javier Diaz-Carmona
Technology Institute of Celaya, Mexico

ABSTRACT

This chapter describes a design of a narrowband lowpass finite impulse response (FIR) filter using a small number of multipliers per output sample (MPS). The method is based on the use of a frequency-improved recursive running sum (RRS), called the sharpening RRS filter, and the interpolated finite impulse response (IFIR) structure. The filter sharpening technique uses multiple copies of the same filter according to an amplitude change function (ACF), which maps a transfer function before sharpening to a desired form after sharpening. Three ACFs are used in the design, as illustrated in the accompanying examples.

INTRODUCTION

The digital filter design problem is concerned with finding a magnitude response (or, equivalently, a gain) that meets the given specifications. These specifications are usually expressed in terms of the desired passband and stopband edge frequencies ω_p and ω_s, the permitted deviations in the passband (passband ripple) R_p, and the desired minimum stopband attenuation A_s (Mitra, 2001). Here we consider the specifications given in decibles (dB). Figure 1 illustrates a typical magnitude specification for a digital lowpass filter.

Due to their complexity, narrowband lowpass FIR filters are difficult and sometimes impossible to implement using conventional structures (Milic & Lutovac, 2002).

The IFIR filter proposed by Neuvo, Cheng and Mitra (1984) is one efficient realization for the design of narrowband FIR filters. The IFIR filter $H(z)$ is a cascade of two filters:

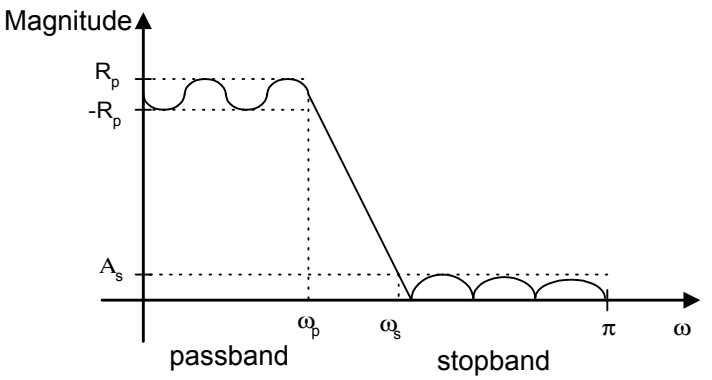
$$H(z) = G(z^M)I(z) \tag{1}$$

where $G(z^M)$ is an expanded shaping or model filter, $I(z)$ is an interpolator or image suppressor, and M is the interpolator factor. In this manner, the narrowband FIR prototype filter $H(z)$ is designed using lower order filters, $G(z)$ and $I(z)$. For more details on the IFIR structure, see Neuvo, Cheng and Mitra (1984).

The interpolation factor M is chosen so that the orders of filters $G(z)$ and $I(z)$ are equal or close to each other.

A linear increase of the interpolation factor results in the exponential growth of the interpolation filter order, as well as in the decrease of the shaping filter order. Our goal is to decrease the shaping filter order as much as possible and to efficiently implement the high-order interpolator filter. To do so, in this chapter, we propose to use RRS filter as an interpolator in the IFIR structure. Similarly, Pang, Ferrari and

Figure 1. Lowpass Filter Magnitude Specification



Sankar (1991) proposed a low order *B*-spline filter as an interpolator filter. More details about RRS filters are given in Chapter XVIII.

The outline of the chapter is as follows. In the first section, a brief review of the filter sharpening technique and the sharpening RRS filters is presented. The following section illustrates the design procedure through three design examples.

SHARPENING RRS FILTER

The sharpening technique, which was first proposed by Kaiser and Hamming (1984), attempts to improve both the passband and stopband of a linear FIR filter by using multiple copies of the same filter. This technique is based on the use of polynomial approximation of a piecewise constant desired amplitude change function (ACF). The ACF maps a transfer function amplitude $H(\omega)$, before sharpening, to an amplitude value after sharpening, $P[H(\omega)]$. The method assumes that $|H(\omega)|$ approximates unity in the passband and zero in the stopband.

Hartnett and Boudreaux (1995) proposed an extension of the method, giving a more direct control over passband and/or stopband improvement. This generalization is achieved by applying the following constraints to the approximating polynomial $P[H(\omega)]$:

- n^{th} order tangency at $\{H(\omega),P[H(\omega)]\}=(0,0)$ to the line of slope δ.
- m^{th} order tangency at $\{H(\omega),P[H(\omega)]\}=(1,1)$ to the line of slope σ.

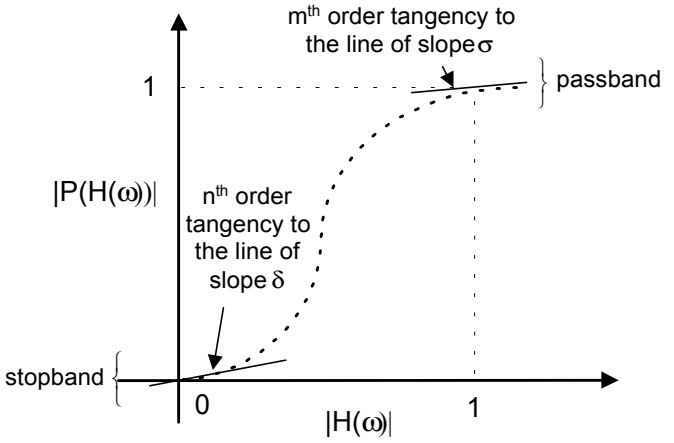
The design parameters are illustrated in Figure 2. Recently, Samadi (2000) derived a closed formula for $P[H(\omega)]$, for arbitrary values of design parameters. It is given by:

$$P[H(\omega)]=\delta H(\omega)+\sum_{j=n+1}^{R}\left(b_{j,0}-\sigma b_{j,1}-\delta b_{j,2}\right)H^{j}(\omega) \tag{2}$$

where

$$\begin{aligned}
b_{j,0} &= \sum_{i=n+1}^{j}(-1)^{j-i}\binom{R}{j}\binom{j}{i}\\
b_{j,1} &= \sum_{i=n+1}^{j}(-1)^{j-i}\binom{R}{j}\binom{j}{i}\left(1-\frac{i}{N}\right)\\
b_{j,2} &= \sum_{i=n+1}^{j}(-1)^{j-i}\binom{R}{j}\binom{j}{i}\frac{i}{N}\\
j &= n+1,....,R\\
R &= n+m+1.
\end{aligned} \tag{3}$$

Figure 2. Filter Sharpening Parameters



As shown in Figure 2, a passband and stopband improvements using the smallest order polynomial are obtained by applying these conditions:

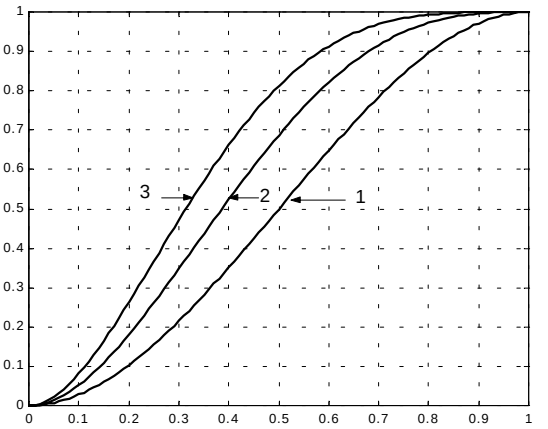
1. Both slopes must be zero (design parameters σ and δ).
2. The tangency order in the point $\{H(\omega),\mathrm{P}[H(\omega)]\}=(0,0)$ must be one (design parameter n).
3. The tangency order in the point $\{H(\omega),\mathrm{P}[H(\omega)]\}=(1,1)$ determines the passband improvement (design parameter m).

Combining these conditions and the Equation (2), improved passband ACF's result, as given in the following expression:

$$
\begin{aligned}
&1)\quad P[H(\omega)] = 3H^2(\omega) - 2H^3(\omega), \\
&\quad (\sigma = 0, \delta = 0, m = 1, n = 1) \\
&2)\quad P[H(\omega)] = 6H^2(\omega) - 8H^3(\omega) + 3H^4(\omega), \\
&\quad (\sigma = 0, \delta = 0, m = 2, n = 1) \\
&3)\quad P[H(\omega)] = 10H^2(\omega) - 20H^3(\omega) + 15H^4(\omega) - 4H^5(\omega), \\
&\quad (\sigma = 0, \delta = 0, m = 3, n = 1)
\end{aligned}
\tag{4}
$$

The plot for each one of these ACFs is shown in Figure 3. Note that the third ACF has the best passband improvement and the smallest stopband attenuation. On the other hand, the first ACF has the worst passband improvement but the best stopband attenuation.

Figure 3. Plot of the Three ACFs



The filter sharpening technique was applied with each one of the three ACFs to a specific RRS filter ($N=8, L=2$). The resulting passband and stopband frequency domain responses are shown in Figure 4 and Figure 5, respectively. One can observe a significant improvement in the RRS magnitude function.

The three sharpened structures are presented in Figure 6, where the resulting number of multipliers per output sample (MPS) is equal to three, four and five for the first, second and third structure, respectively.

DESCRIPTION OF THE DESIGN PROCEDURE

In the previous section, we outlined a sharpening RRS filter technique. We now consider the design of a narrowband lowpass filter using an IFIR structure, where the interpolator filter is the sharpening RRS filter. In order to suppress the mirror

Figure 4. Passband Magnitudes for Sharpening RRS Filters

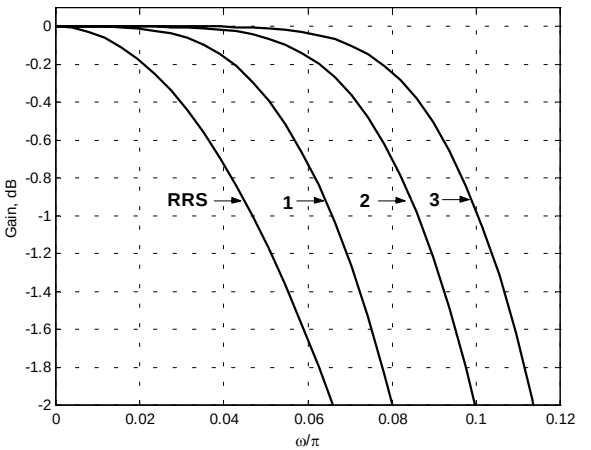
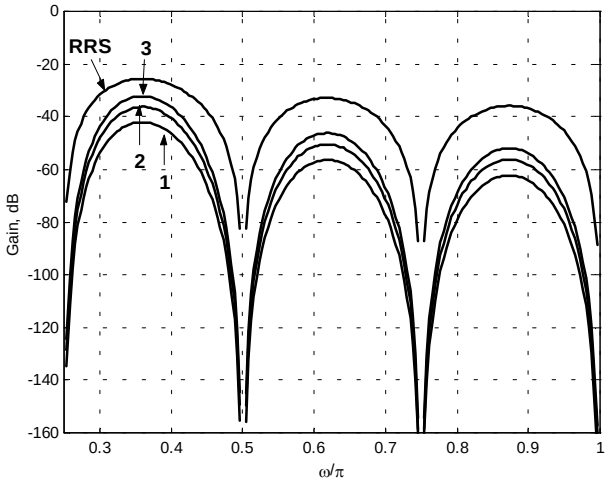


Figure 5. Stopband magnitude for sharpening RRS filters



images of the expanded model filter, the RRS frequency nulls must be centered at each mirror image, as shown in Figure 7. This implies that the length of the RRS filter must be equal to the interpolator factor *M*.

The specifications of the filter to be designed are: normalized passband edge at ω_p, normalized stopband edge at ω_s, passband ripple R_p, and a minimum stopband attenuation A_s in dB. The number of stages in the RRS filter *L* controls the resulting filter stopband attenuation, and the filter sharpening technique controls the filter passband width.

Figure 6. Sharpening Structures

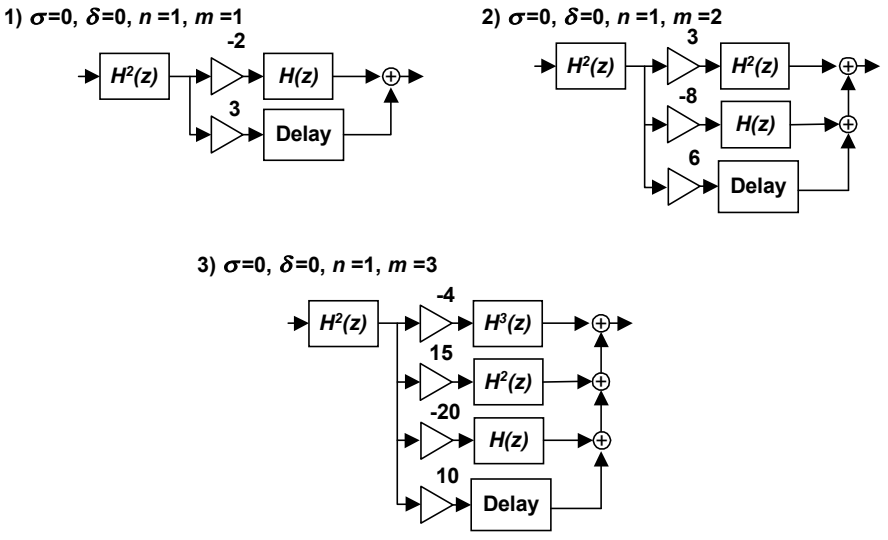
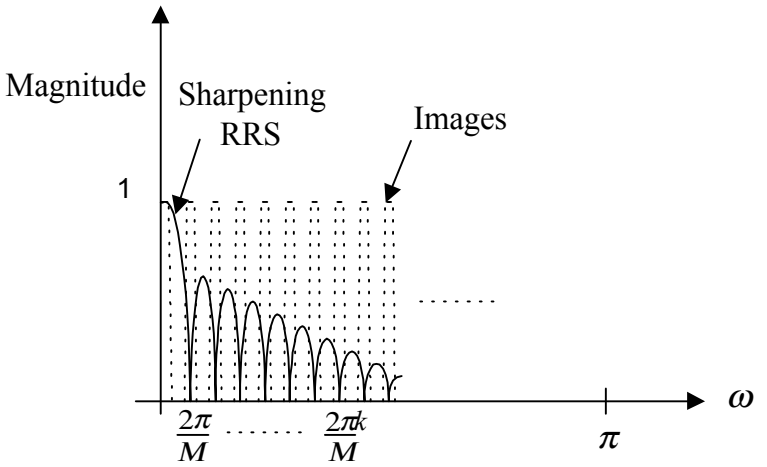


Figure 7. Mirror Images Suppression where k is an Integer $0 \le k \le M-1$



The procedure for the filter design is outlined in the following steps:

1. Choose the interpolation factor M and design the model filter $G(z)$, using the specifications:

$$\omega_p^G = M\omega_p, \qquad \omega_s^G = M\omega_s,$$

$$R_p^G = \frac{R_p}{2}, \qquad A_s^G = A_s \tag{5}$$

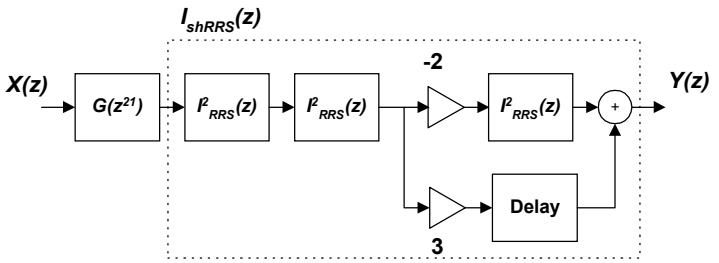
where ω_p^G and ω_s^G are the passband and the stopband edge frequencies, R_p^G is the maximum passband ripple, and A_s^G is the minimum stopband attenuation of the filter $G(z)$.

2. Design the L-stage RRS filter $I_{RRS}(z)$ of the length M.
3. Choose an ACF and apply the filter sharpening to the filter $I_{RRS}(z)$. The resulting filter is denoted as $I_{shRRS}(z)$.
4. Cascade the model filter $G(z^M)$ and the sharpening RRS filter $I_{shRRS}(z)$.
5. If the stopband filter specification A_s is not satisfied, go to Step 2 and increase the number of stages L.
6. If the passband filter deviation is higher than R_p, go to Step 3 and change ACF.

We illustrate the design procedure with three examples.

Example 1: A linear-phase narrowband lowpass FIR filter is designed with the following specifications: normalized passband edge at ω_p=0.01, normalized stopband

Figure 8. Resulting Structure in Example 1 with First ACF



edge at $\omega_s = 0.015$, maximum passband ripple $R_p = 0.086$dB, and minimum stopband attenuation A_s = -60 dB.

The linear-phase FIR filter designed by Parks McClellan method would require an order $N = 1017$. Using $M = 18$ in the IFIR structure, the orders of filters $G(z)$ and $I(z)$, both designed by Parks McClellan method, are 66 and 69, respectively.

We choose the first ACF given in Equation (4), the interpolation factor $M = 21$, and the number of the stages of RRS filter $L = 2$. The order of the filter $G(z)$ is 57. The resulting structure is shown in Figure 8. The magnitude responses of $G(z^{21})$ and $I_{shRRS}(z)$ are shown in Figure 9, where we can see that the mirror images are suppressed by nulls of the sharpening RRS filter. The allband magnitude response for the designed filter and the passband zoom are shown in Figure 10.

If the filter sharpening is applied using the second ACF in Equation (4), it would be possible to use higher values for the interpolation factor M, for example we can choose $M = 31$. In this case the RRS filter has an order 31, and the number of the stages L is 3 (Figure 11).

As a consequence, the order of the filter $G(z)$ is 40. The resulting allband magnitude response as well as the passband zoom for this case are shown in Figure 12.

Figure 9. Allband Magnitude Responses for $G(z^{21})$(Dotted Line) and $I_{shRRS}(z)$(Solid Line)

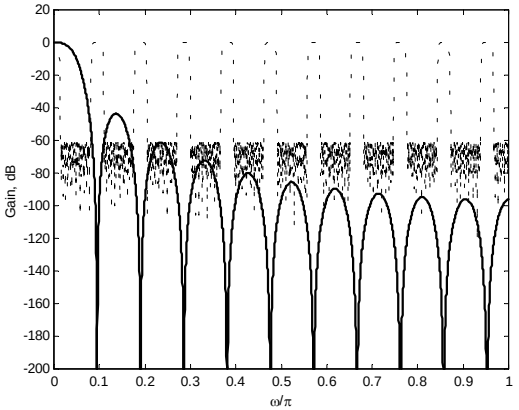
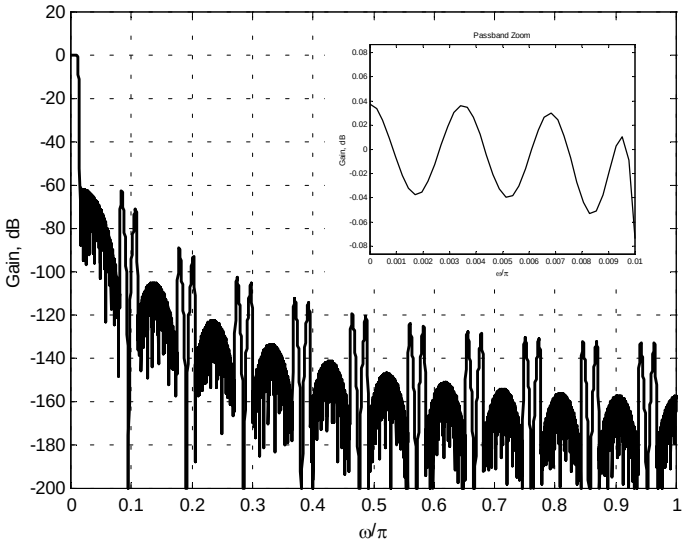


Figure 10. Magnitude Responses for Example 1 and First ACF



A further order reduction of the filter $G(z)$ is obtained using the third ACF in Equation (4). Here the interpolation factor M is 37, and the number of the stages of RRS filter L, is equal to four. As a result, the model filter order has an order 34. The mirror image suppression and the final magnitude response are shown in Figures 13 and 14, respectively.

The design parameters for all three cases and the resulting number of MPS are summarized in Table 1. Note that the number of MPS is decreased as a result of the increase in the complexity of ACF.

Two additional examples are given below. Unlike Example 1, the next example illustrates the design of a filter with a wider bandwidth than one given by Example 1.

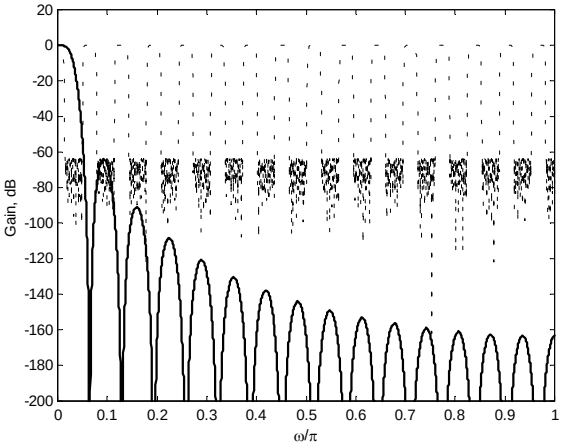
Example 2: Design a linear-phase narrowband lowpass FIR filter in order to meet the following specifications: normalized passband and stopband edges at ω_p = 0.02 and ω_s = 0.03, maximum passband ripple R_p = 0.086dB and minimum stopband attenuation A_s = - 60 dB.

The direct application of the Parks McClellan algorithm results in a filter whose order is equal to 509. Using an interpolator factor M = 11 in the IFIR structure, we

Table 1. Design Parameters and Number of MPS for Example 1

ACF	M	N_G	L	MPS
1	21	57	2	31
2	31	40	3	24
3	37	34	4	22

Figure 11. Magnitude Responses for $G(z^{31})$(Dotted Line) and $I_{shRRS}(z)$(Solid Line)



can obtain the filters $G(z)$ and $I(z)$, both designed by Parks McClellan method, of the orders 55 and 47, respectively. The design results for all three ACFs are summarized in Table 2.

The magnitude responses for the first, second, and third cases are shown in Figures 15, 16 and 17, respectively.

Example 3: This example considers the design of a very narrowband filter having a normalized passband edge at $\omega_p = 0.008$ and a normalized stopband edge at $\omega_s = 0.012$. Passband ripple is $R_p = 0.1728$dB, and the minimum stopband attenuation is $A_s = -65$ dB.

Figure 12. Magnitude Responses from Example 1 and Second ACF

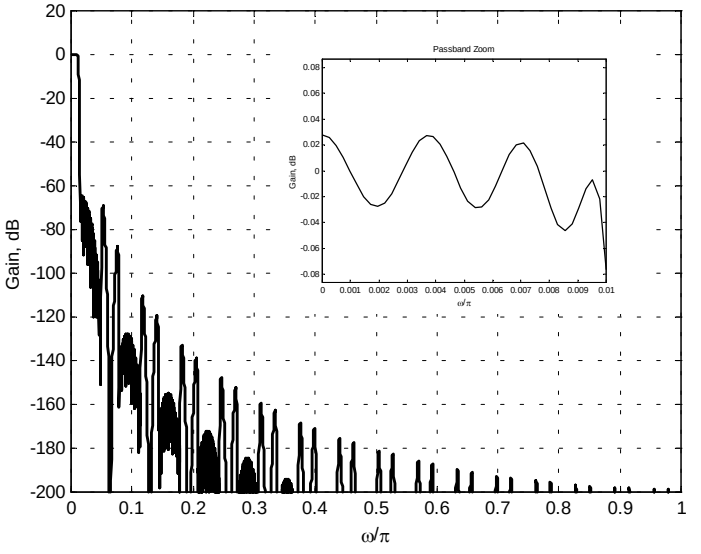


Figure 13. Magnitude Responses for $G(z^{37})$ (Dotted Line) and $I_{shRRS}(z)$ (Solid Line)

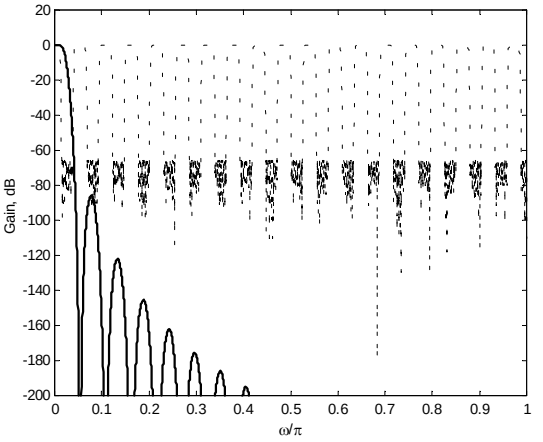


Figure 14. Magnitude Responses for Example 1 and Third ACF

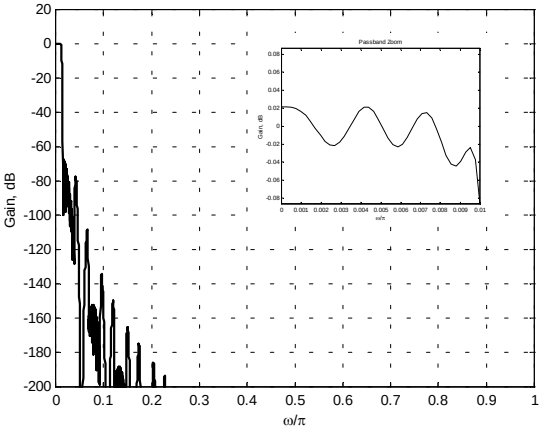


Figure 15. Magnitude Responses for Example 2 with First ACF

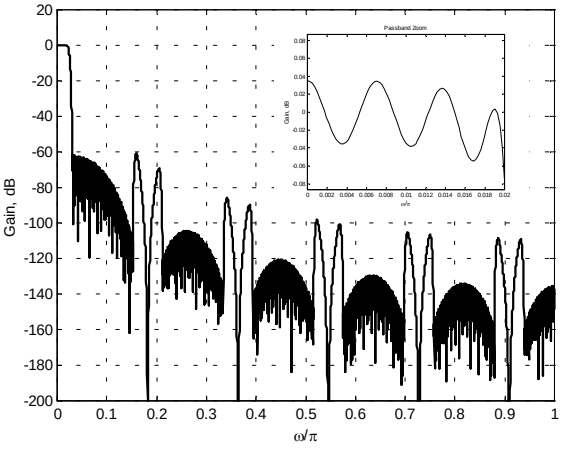


Figure 16. Magnitude Responses for Example 2 with Second ACF

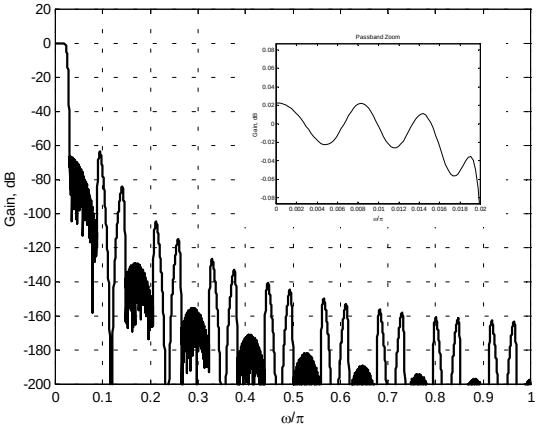


Figure 17. Magnitude Responses for Example 2 Using Third ACF

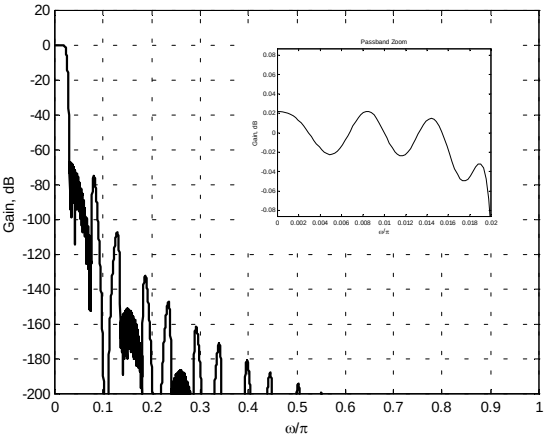


Figure 18. Magnitude Responses for Example 3 with First ACF

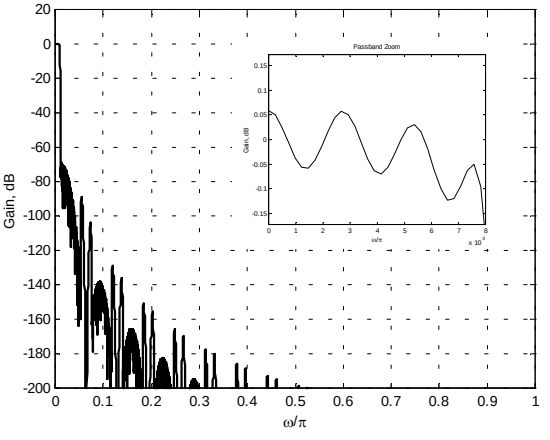


Table 2. Design Parameters and Number of MPS for Example 2

ACF	M	N_G	L	MPS
1	11	55	2	30
2	17	37	3	22
3	19	33	4	21

Table 3. Design Parameters and Number of MPS for Example 3

ACF	M	N_G	L	MPS
1	31	48	3	27
2	41	37	3	22
3	51	31	4	20

Using Parks McClellan method, such specifications require a filter of an order of 1232. However, the model filter and the interpolator in the IFIR structure both have an order of 72, when the interpolation factor M is equal to 20. Table 3 summarizes the value of the parameters needed for each of the three ACFs. The corresponding magnitude responses are plotted in Figures 18, 19, and 20.

CONCLUSION

A narrowband lowpass FIR filter design method with a small number of multipliers per output sample (MPS) is described. The number of multipliers is reduced by using a sharpening RRS filter as an interpolator in the IFIR structure. To this end, three ACFs are obtained with the design parameters σ, δ, n and m. The

Figure 19. Magnitude Responses for Example 3 with Second ACF

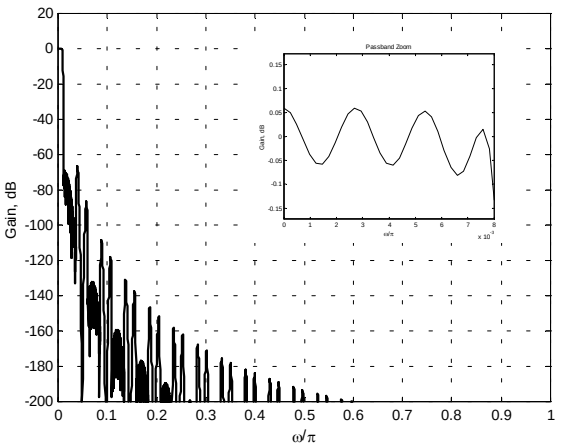
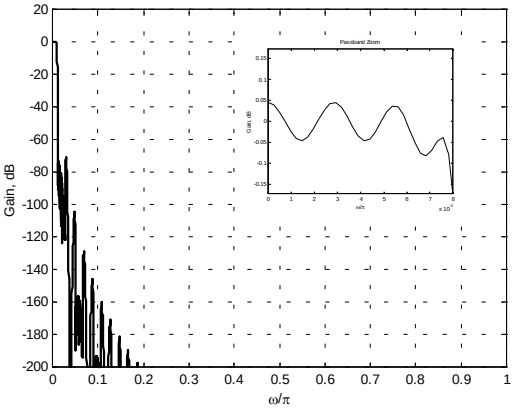


Figure 20. Magnitude Responses for Example 3 with the Third ACF



design control parameters for a given ACF are the interpolation factor *M* and the number of stages *L* of an RRS filter.

As the filter design examples presented in this chapter show, there is a notable reduction in the number of MPS, using the proposed method.

REFERENCES

Hartnett, R. J., & Boudreaux, G. F. (1995, December). Improved filter sharpening. *IEEE Transactions on Signal Processing, 43,* 2805-2810.

Hogenauer, E. B. (1981, April). An economical class of digital filters for decimation and interpolation. *IEEE Transactions on Acoustics Speech and Signal Processing*, *29,* 155-162.

Kaiser, J. F., & Hamming, R. W. (1984, October). Sharpening the response of a symmetric nonrecursive filter by multiple use of the same filter. *IEEE Transactions on Acoustics Speech and Signal Processing, ASSP-25,* 415-422.

Milic L. J., & Lutovac, M. (2002). Efficient multirate filtering (Chapter IV). In G. J. Dolecek (Ed.) *Multirate Systems: Design and Applications*. Hershey, PA: Idea Group Publishing.

Mitra, S. K. (2001). *Digital signal processing: a computer-based approach.* 2nd ed. New York: McGraw-Hill.

Neuvo, Y., Cheng-Yu, D., & Mitra, S. (1984, June). Interpolated finite impulse response filters. *IEEE Transactions on Acoustics Speech and Signal Processing, 32,* 563-570.

Pang, D. L., Ferrari, A., & Sankar, P. V. (1991, September). A unified approach to IFIR filter design using B-spline functions. *IEEE Transactions on Signal Processing, 39,* 2115-2117.

Samadi, S. (2000, October). Explicit formula for improved filter sharpening polynomial. *IEEE Transactions on Signal Processing, 9,* 2957-2959.

Chapter XVIII

Design of Narrowband Highpass FIR Filters Using Sharpening RRS Filter and IFIR Structure

Gordana Jovanovic-Dolecek
National Institute of Astrophysics Optics and Electronics (INAOE), Mexico

ABSTRACT

This chapter presents the design of narrowband highpass linear-phase finite impulse response (FIR) filters using the sharpening recursive running sum (RRS) filter and the interpolated finite impulse response (IFIR) structure. The novelty of this technique is based on the use of sharpening RRS filter as an image suppressor in the IFIR structure. In that way, the total number of multiplications per output sample is considerably reduced.

INTRODUCTION

FIR filters are often preferred to infinite impulse response (IIR) filters because of their attractive properties, such as the linear phase, stability and the absence of the limit cycle (Mitra, 2001). The main disadvantage of FIR filters is that they involve a

higher degree of computational complexity compared to IIR filters with equivalent magnitude response. In the past few years, many design methods have been proposed to reduce the complexity of FIR filters, (Jou, Hsieh & Kou, 1997; Kumar & Kumar, 1999; Webb & Munson, 1996; Lian & Lim, 1998; Bartolo & Clymer, 1996, etc.).

We consider highpass (HP) linear-phase narrowband filters. As it is known, one of the most difficult problems in digital filtering is the implementation of narrowband filters. The difficulty lies in the fact that such filters require high-order design in order to meet the desired frequency response specifications. In return, these high-order filters need a large amount of computation and are difficult to implement. Here we propose an efficient implementation of HP liner-phase narrowband digital filters based on IFIR and RRS filter.

We first consider the transform of a lowpass filter (LP) into its HP equivalent. We then describe the design of HP filters using an IFIR structure, separately considering an even and an odd interpolation factor. We also review the basics of RRS and sharpening RRS filters. Finally, we explain how to design an HP filter, using the sharpening RRS-IFIR structure. Two methods are presented, but the choice between the two depends on the parity of the RRS filter. Several examples accompany the outlined procedures. All filters are designed using MATLAB.

TRANSFORM OF LP INTO HP FILTER

Instead of designing an HP filter by brute force, we can transform an LP filter into an HP one. First we replace the desired cutoff frequencies of the HP filter, ω_p and ω_s, with the corresponding LP specifications as follows:

$$\begin{aligned} \omega_p' &= \pi - \omega_p \\ \omega_s' &= \pi - \omega_s \end{aligned} \tag{1}$$

Given these specifications, an LP-FIR filter can be designed. From this auxiliary LP filter, the desired HP filter can be computed by simply changing the sign of every other impulse response coefficient. This is compactly described in Equation 2:

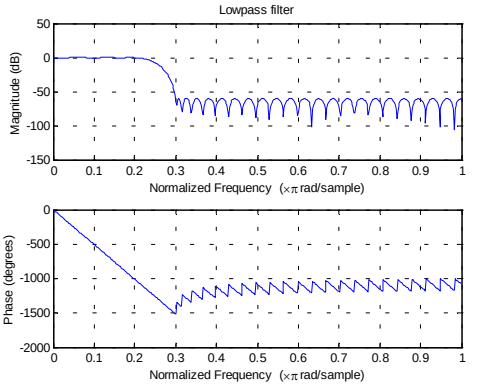
$$h_{HP}(n) = (-1)^n h_{LP}(n) \tag{2}$$

where $h_{HP}(n)$ and $h_{LP}(n)$ are the impulse responses of the HP and the LP filters, respectively.

The following example illustrates the procedure.

Example 1: We consider the design of HP-FIR filter using Parks-McClellan algorithm (Mitra, 2001) with these specifications: normalized passband edge ω_p =

Figure 1. LP Filter



0.8π, normalized stopband edge $\omega_s = 0.7\pi$, passband ripple $Rp = 1$dB, and minimum stopband attenuation A_s = -60 dB. Using (1) we obtain the corresponding LP specifications:

$$\omega_p' = \pi - 0.8\pi = 0.2\pi$$
$$\omega_s' = \pi - 0.7\pi = 0.3\pi \qquad (3)$$

The order of the filter is $N = 57$. The magnitude and phase responses of the designed LP filter are shown in Figure 1. Using Equation 2, we transform the LP filter into the corresponding HP filter. The magnitude and phase responses of the resulting HP filter are shown in Figure 2.

The impulse responses of the LP and the HP filter are shown in Figure 3.

Figure 2. HP Filter

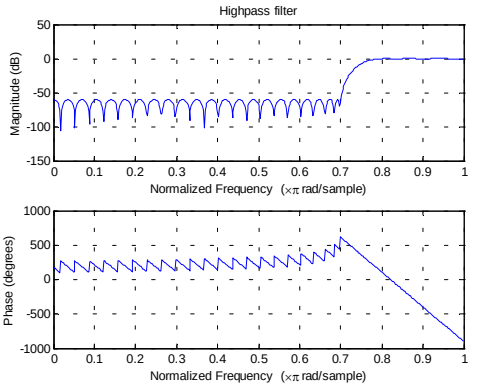
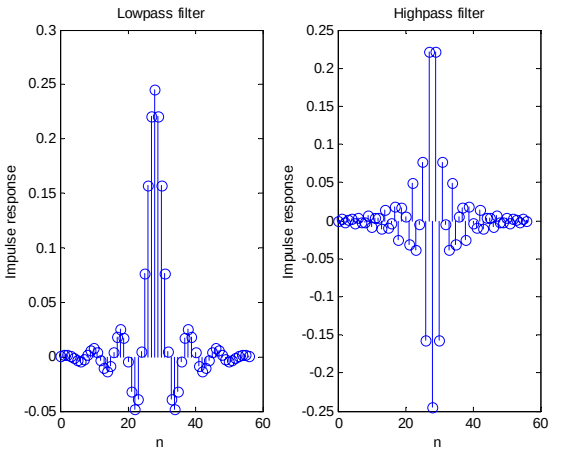


Figure 3. Impulse Responses



(a) (b)

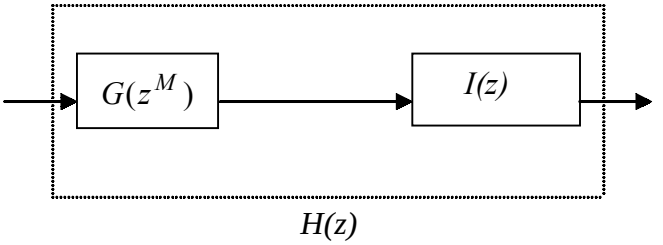
IFIR STRUCTURE

The IFIR structure proposed by Neuevo, Dong and Mitra (1984) is an efficient realization of a high-order liner-phase LP FIR filter. Instead of designing one high-order liner-phase LP filter $H(z)$, two lower order liner-phase LP filters are computed. One of them is called the shaping filter $G(z)$ and the other one is the interpolator filter $I(z)$. Suppose that the specifications of the original filter $H(z)$ are: normalized passband edge ω_p, normalized stopband edge ω_s, passband ripple R_p, and minimum stopband attenuation A_s. The specification of the LP filter $G(z)$ can then be expressed as follows:

$$\begin{aligned} \omega_p^G &= M\omega_p \\ \omega_s^G &= M\omega_s \\ R_p^G &= R_p / 2 \\ A_s^G &= A_s \end{aligned} \tag{4}$$

where M is the interpolation factor and the upper index G denotes for the filter $G(z)$. The filter $G(z)$ is expanded M times, and the filter $G(z^M)$ is obtained by replacing each delay z^{-1} in the filter $G(z)$ with the delay z^{-M}. In the time domain, this is equivalent to inserting $M-1$ zeros between two consecutive samples of the impulse response of the filter $G(z)$. The expansion of the filter $G(z)$ introduces $M-1$ images in the range $[0, 2\pi]$, which have to be eliminated. This is why the filter interpolator $I(z)$ is needed. The general LP IFIR structure is given in Figure 4.

Figure 4. IFIR Structure



Finally, the HP filter can be designed by combining the transformation in Equation (3) and the procedure for designing an LP filter described above.

Interpolation Factor *M* is Even

For even *M* there is an image at high frequency. If all other images, along with the original spectrum are eliminated, the HP filter results. Figure 5 shows the expanded filter for $M = 2, 4, 6, 8$. We notice that we can obtain the desired HP filter by eliminating the original LP spectrum and all images except one at high frequency. That means that the interpolator filter is an HP filter with these specifications:

$$\begin{aligned} \omega_p^I &= \pi - \omega_p^{'} \\ \omega_s^I &= \frac{2\pi}{M}\frac{M-2}{2} + \omega_s^{'} = \frac{\pi(M-2)}{M} + \omega_s^{'} \\ R_p^I &= R_p / 2 \\ A_s^I &= A_s \end{aligned} \tag{5}$$

where upper index *I* stands for the interpolator $I(z)$, and $\omega_p^{'}$ and $\omega_s^{'}$ are the lowpass passband and stopband normalized edges of the filter $G(z)$.

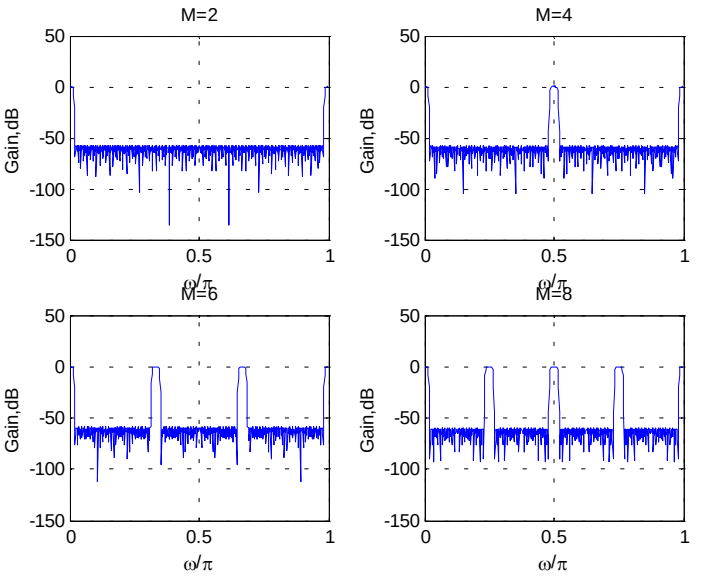
The procedure for the design of a HP-IFIR filter is outlined in the following steps:

1. Design the LP filter $G(z)$ using (4).
2. Expand the filter $G(z)$ M times.
3. Design the HP interpolator $I(z)$ with the specifications in Equation 5.
4. Cascade filters $G(z^M)$ and $I(z)$.

This approach is illustrated in the Example 2.

Example 2: To clarify the design of an HP filter utilizing the IFIR structure with the even interpolation factor *M*, we consider the design of a linear-phase FIR LP filter

Figure 5. Expanded Filter G(z) for Different Values of M



with the same specifications as given in Example 1, i.e., $\omega_p = 0.8\pi$, $\omega_s = 0.7\pi$, $R_p =$ 1 dB, and $A_S =$ -60 dB. The interpolation factor M is 2. The detailed procedure of the design is presented in the following steps:

1. Design an LP filter $G(z)$ with these specifications:

$$\begin{aligned} \omega_p^G &= 2 \cdot 0.2\pi \\ \omega_s^G &= 2 \cdot 0.3\pi \\ R_p &= 0.1/2\text{dB} \\ A_s &= -60\text{dB} \end{aligned} \tag{6}$$

The order of the filter $G(z)$ is $N_G = 32$. The filter is shown in Figure 6(a).

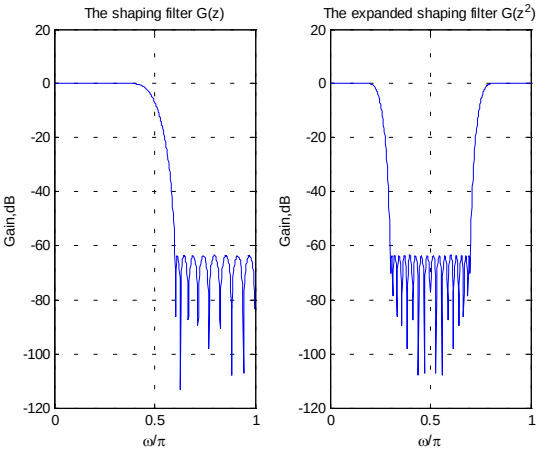
2. We expand the filter $G(z)$, $M = 2$ times. Figure 6(b) depicts the filter.
3. Design an HP interpolator with the specifications:

$$\begin{aligned} \omega_p^I &= \pi - \omega_p^{'} = \pi - 0.2\pi = 0.8\pi \\ \omega_s^I &= \omega_s^{'} = 0.3\pi \end{aligned} \tag{7}$$

The interpolator filter has an order of $N_I = 13$, and it is plotted in Figure 7(a).

4. Cascade the expanded filter and the interpolator. The resulting HP filter is shown in Figure 7(b).

Figure 6. Shaping (a) and Expanded Shaping Filter, M = 2, (b)



As the previous example illustrates, we have designed two filters $G(z)$ and $I(z)$ with the respective orders 32 and 13, instead of designing a prototype filter $H(z)$ with the higher order of 57 (see Example 1).

Interpolation Factor *M* is Odd

This section considers the IFIR design for an odd interpolation factor M. Figure 8 shows the expanded filters $G(z)$ for $M = 3, 5, 7$ and 9.

As Figure 8 shows, there are no images at high frequency. In order to obtain the LP filter, all images have to be eliminated using the interpolator filter, so that only the original spectrum remains.

Figure 7. The Interpolator (a) and the Resulting HP Filter (b)

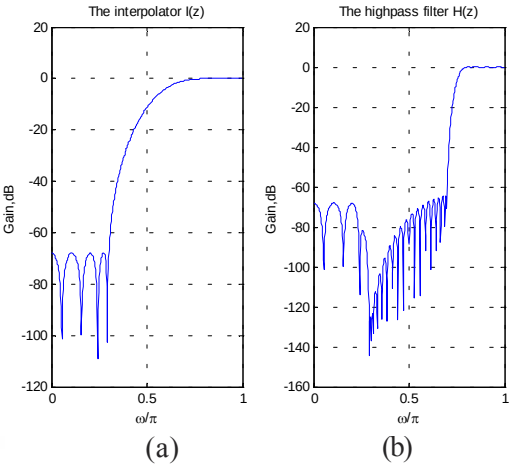
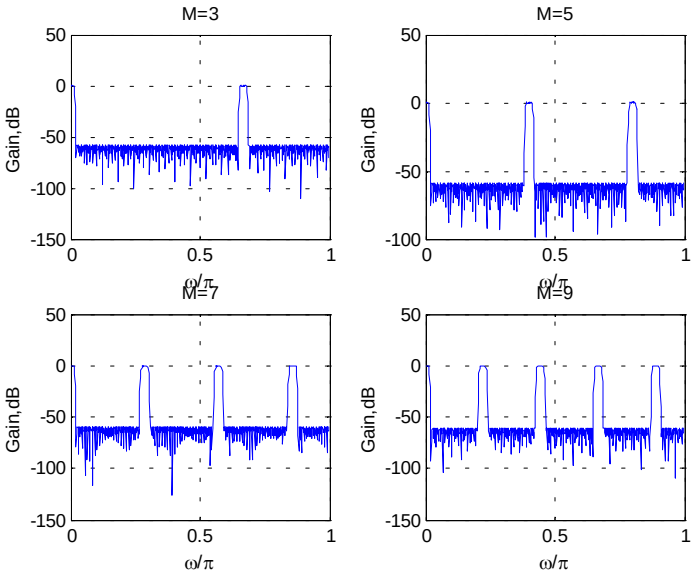


Figure 8. Expanded Filters for the Different Values M



The steps of the design are:

1. Design an LP filter $G(z)$ using Equation 1.
2. Expand the filter $G(z)$ M times.
3. Design the LP interpolator $I(z)$ with the specifications:

$$\begin{aligned} \omega_p^I &= \pi - \omega_p' \\ \omega_s^I &= \frac{2\pi}{M} - \omega_s' \\ R_p^I &= R_p / 2 \\ A_s^I &= A_s \end{aligned} \tag{8}$$

4. Transform the filters $G(z)$ and $I(z)$ into HP filters $G'(z)$ and $I'(z)$ using Equation (2).
5. Cascade the filters $G'(z^M)$ and $I'(z)$.

We illustrate the application of the above design in the following example.

Example 3: Consider the HP filter of Example 2 with an interpolation factor $M = 3$. To this end we follow the steps described so far:

1. Design an LP filter $G(z)$ with the specifications:

$$\begin{aligned} \omega_p^G &= 3 \cdot 0.2\pi = 0.6\pi \\ \omega_s^G &= 3 \cdot 0.3\pi = 0.9\pi \\ R_p &= 0.1/2\text{dB} = 0.05\text{dB} \\ A_s &= -60\text{dB} \end{aligned} \tag{9}$$

The filter $G(z)$ has an order of $N_G = 21$, and it is shown in Figure 9(a).

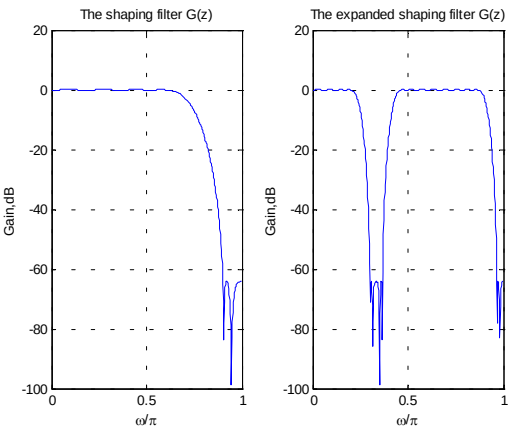
2. Expand the filter $G(z)$, $M = 3$ times. The expanded filter is shown in Figure 9(b).
3. Design a LP interpolator $I(z)$ with the specifications:

$$\begin{aligned} \omega_p^I &= \omega_p' = 0.2\pi \\ \omega_s^I &= \frac{2\pi}{3} - \omega_s' = 0.33\pi \end{aligned} \tag{10}$$

The filter has an order of $N_I = 34$, and it is shown in Figure 10(a).

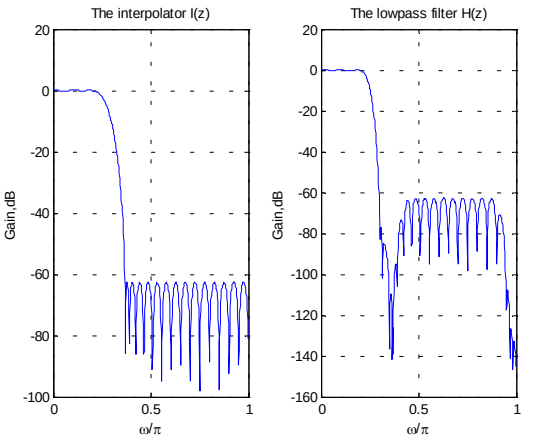
4. Cascade the expanded LP filter and the interpolator. The resulting LP filter is shown in Figure 10(b).
5. Transform the resulting LP filter into its counterpart HP filter. The result is shown in Figure 11.

Figure 9. The Shaping (a) and the Expanded Shaping Filter, M *= 3, (b)*



(a) (b)

Figure 10. The Interpolator (a) and the Resulting LP Filter (b)



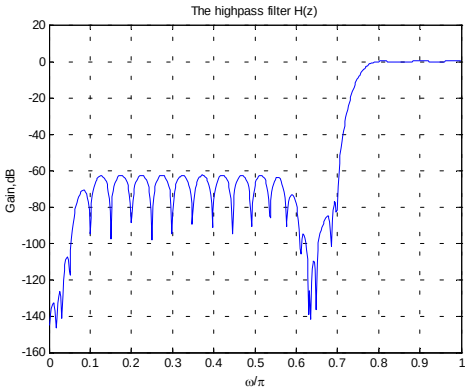
Note that the order of the filter $G(z)$ is decreased, and the order of the filter $I(z)$ is increased as the result of the increased interpolation factor M.

RECURSIVE RUNNING SUM FILTER

The simplest LP-FIR filter is the moving-average (MA) filter. Its impulse response g(n) is given in Equation 11.

$$g(n) = \frac{1}{M} \sum_{k=0}^{M-1} g(n-k) \tag{11}$$

Figure 11. The Resulting HP Filter



All impulse response coefficients are equal to 1, which require no multiplication. The corresponding transfer function is given by:

$$G(z) = \frac{1}{M}[1 + z^{-1} + \ldots + z^{-(M-1)}] = \frac{1}{M}\sum_{k=0}^{M-1} z^{-k} \tag{12}$$

A more convenient form of the above transfer function for the realization purpose is given by:

$$H_{RRS}(z) = [G(z)]^K = \left[\frac{1}{M}\frac{1-z^{-M}}{1-z^{-1}}\right]^K \tag{13}$$

which is also known as a recursive RRS (Mitra, 2001), or a boxcar filter. The scaling factor $1/M$ is needed to provide a dc gain of 0 dB, and K is the number of the cascaded sections of the filter. The magnitude response of the filter can be expressed as

$$\left|H_c(e^{j\omega})\right| = \left|\left[\frac{\sin(\omega M/2)}{M\sin(\omega/2)}\right]^K\right| \tag{14}$$

Figure 12. RRS Filters for Different Values M and K

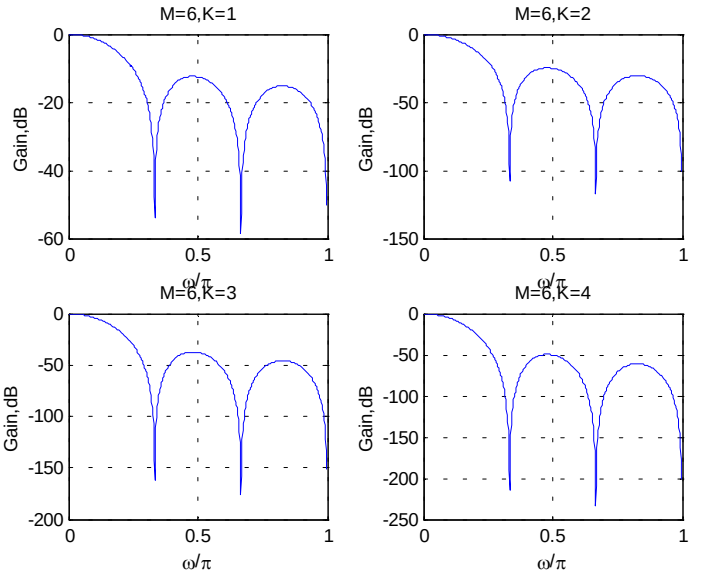
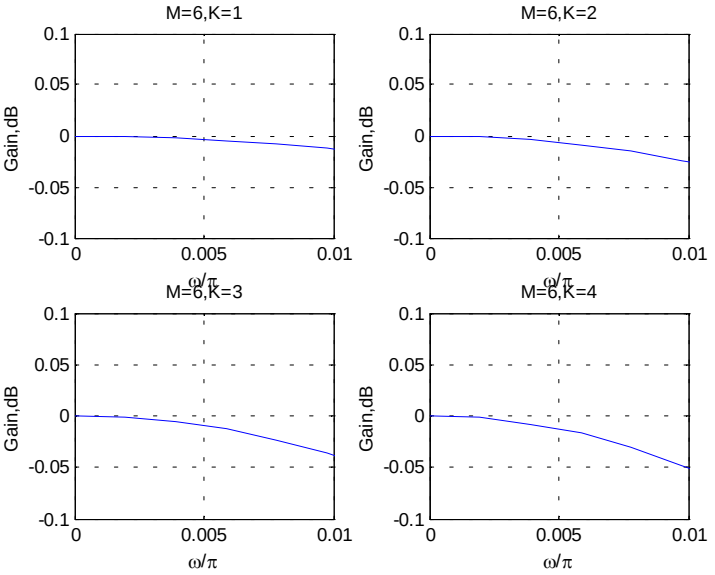


Figure 13. Passband Details of RRS Filters



The magnitude response has nulls at integer multiples of $(1/M)F_S$, where F_S is the sampling frequency. This provides natural alias-rejections introduced by expanding the shaping filter $G(z)$ by the factor M.

The magnitude response of a RRS filter of the order $M = 6$ and varying number of cascaded sections K is shown in Figure 12.

Note that the stopband attenuation is improved as a result of the increased number of stages. However, this also downgrades the passband, as shown in Figure 13.

Therefore, the use of the RRS filter as an interpolator in the IFIR structure has a limited application. The improvement of the RRS characteristics is considered in the next section.

SHARPENING THE RRS FILTER

The improved RRS characteristics are based on a technique for sharpening the response of a filter proposed by Kaiser and Hamming (1977) and Kwentus, Jiang and Willson (1997). This technique improves both the passband and stopband of a RRS filter, using multiple copies of the original filter. The simplest sharpened RRS filter $H_{shRRS}(z)$ can be related to the original RRS filter as shown in Equation (15).

$$H_{shRRS}(z) = 3H_{RRS}^2(z) - 2H_{RRS}^3(z) \tag{15}$$

where the RRS filter $H_{RRS}(z)$ is given in Equation 13.

Figure 14. Sharpening RRS Filters for Different M and K

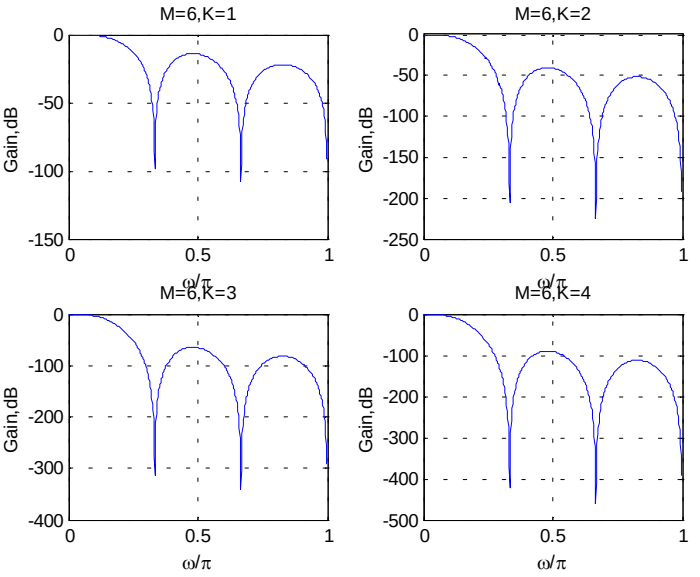


Figure 14 illustrates how the sharpening RRS filter characteristics are improved by increasing the number of stages K, using $M = 6$ and $K = 1, 2, 3$, and 4.

In Figure 14, the considerable improvements in the stopband can be observed. In order to see the details of the passband, we plot the same diagrams for the range of 0 to 0.04 of the relative frequency and the range of –0.01 to 0.01dB of the gain, as shown in Figure 15.

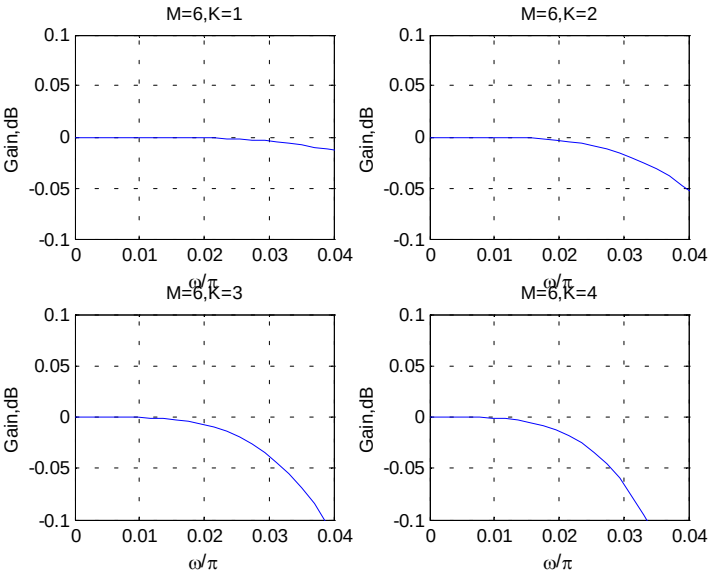
Unlike the plots in Figure 13, where RRS filters have a passband droop, we note that the sharpening RRS filter has flat response in the range of 0 to 0.01 of the relative frequency. This result indicates that the sharpening RRS filter has improved passband and stopband characteristics compared with to the ones of the RRS filter, making it a good candidate for an interpolator in an IFIR structure. The corresponding block diagram is shown in Figure 16. As previously noted, the sharpening RRS-IFIR structure can be used for the design of an LP filter. How to design an HP filter depends on whether the factor M is even or odd and is described in the next section.

Design of Sharpening HP-IFIR Filter for an Even *M*

As demonstrated so far, for an even interpolation factor M, the interpolator has to eliminate the original spectrum and also all images except the one at high frequency. As a result, the interpolator has to be an HP sharpening filter. The following seven-step procedure may be used to design a sharpening HP-IFIR filter, when M is an even number.

1. Transform the HP specifications into the LP ones.
2. Choose the value M.

Figure 15. Passband Details for Different Values of M and K



3. Design the LP filter $G(z)$.
4. Expand the filter $G(z)$ M times.
5. Choose the value K and design the LP-RRS filter. Transform the LP-RRS filter into an HP filter and design the corresponding sharpening HP-RRS filter. As an example, Figure 17 shows the HP sharpening RRS filters for $M = 10$ and 14 and for $K = 3$ and 4.
6. Cascade the expanded filter $G(z^M)$ and the sharpening HP-RRS filter.
7. Check if the specification of the resulting filter is satisfied. If not, choose different values for M and K and repeat the design.

We illustrate the above design in the following example.

Example 4: Consider an HP filter having passband edges at $\omega_p = 0.991\pi$, and $\omega_S = 0.98\pi$, passband ripple $R_p = 0.25$ dB, and the minimum stopband attenuation A_S = -60 dB.

Figure 16. Sharpening IFIR Structure

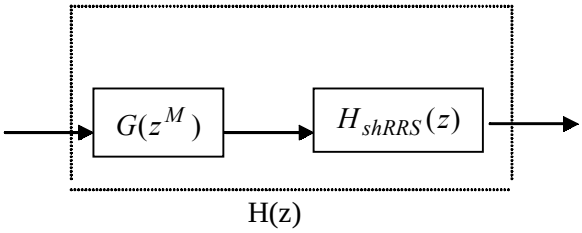
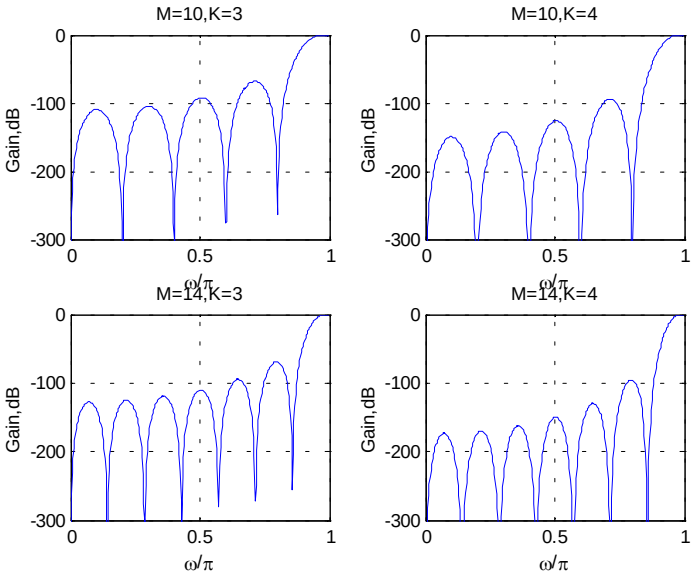


Figure 17. Sharpening RRS Filter for M = 10 and 14 and for K = 3 and 4

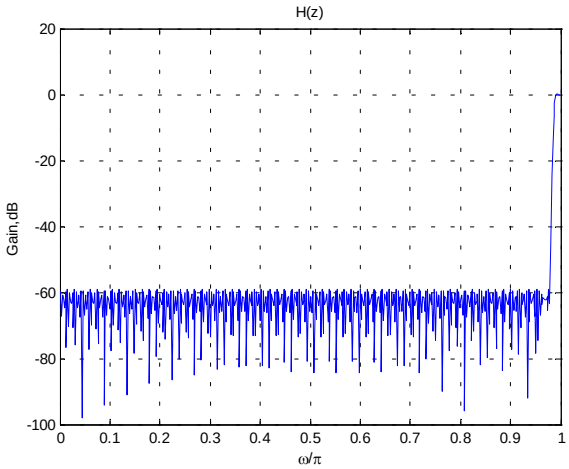


The HP filter, which satisfies the specification, designed by using Parks-McClellan algorithm, has an order of $N_H = 446$. Its magnitude response is shown in Figure 18.

The next steps illustrate the design in more detail.

1. Transform the given HP specifications into the LP specifications:

Figure 18. The Prototype Filter H(z)



$$\omega_p' = 0.009\pi$$
$$\omega_s' = 0.02\pi \qquad (16)$$

2. Choose $M = 16$.
3. Design the LP filter $G(z)$. The order of the filter N_G is 29.
4. Expand the filter $G(z)$, $M = 16$ times. The corresponding magnitude response is shown in Figure 19.
5. Choose $K = 2$ and design the LP-RRS filter. Transform it into the HP-RRS filter. Design the corresponding HP sharpening RRS filter. The magnitude response of the filter is shown in Figure 20.
6. Cascade the LP-expanded shaping filter and the HP sharpening RRS filter. The resulting magnitude response is shown in Figure 21.
7. Check if the passband and the stopband specifications are satisfied. As Figures 22 and 23 show, both specifications are satisfied. In other case choose other value for M and K.

It should be noted that instead of designing a filter with the order of 446, the lower order filter $G(z)$ that also satisfies the required specifications, but whose order is only 16, could be designed. The filter $I(z)$ is a sharpening filter, which is also a simple filter and needs only three integer multiplications.

Design of HP-IFIR Sharpening RRS Filter for an Odd *M*

Unlike the previously described design, if the interpolation factor M is an odd number, then both filters in the IFIR structure, the shaping and the interpolator filter need to be transformed into HP filters. A detailed explanation of the design is presented in the following steps.

Figure 19. Expanded Shaping Filter, M = 16

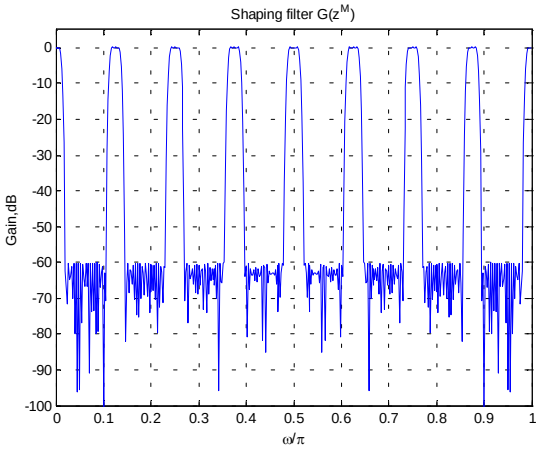
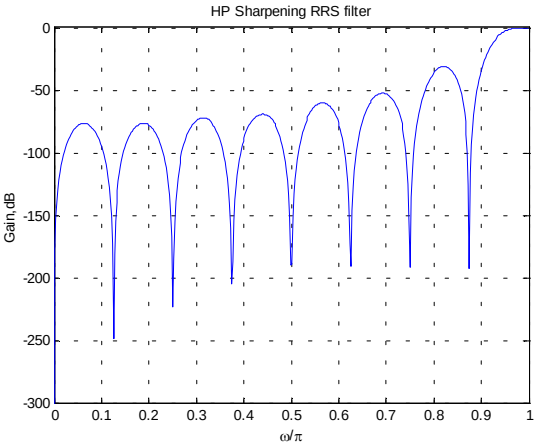


Figure 20. Sharpening RRS Filter, M = 16, K = 2



1. Transform the HP specifications into the LP ones.
2. Choose the value *M*.
3. Design the LP filter *G* (*z*).
4. Expand the filter *G* (*z*) *M* times.
5. Transform the expanded filter into the HP one.
6. Choose the value *K* and design the LP-RRS filter. Transform it into an HP-RRS filter and design HP sharpening RRS filter.
7. Cascade the expanded HP shaping filter and the HP sharpening RRS filter.
8. Check if the specification is satisfied. If it is not, try with other values for *M* and *K* and repeat the design.

We illustrate the procedure in the next example.

Figure 21. HP Sharpening RRS IFIR Filter

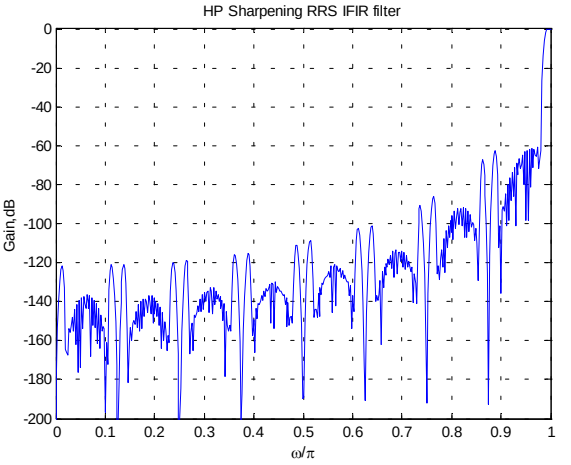
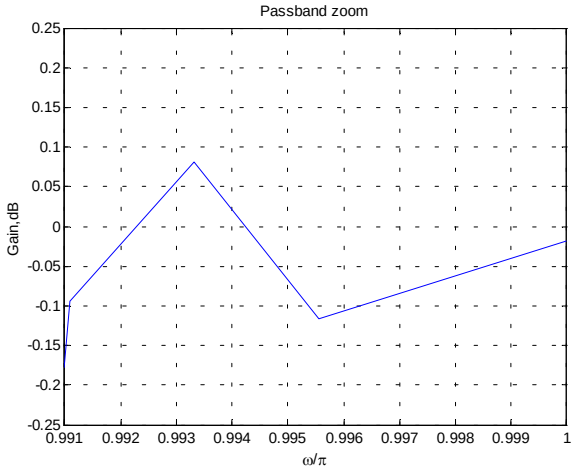


Figure 22. Passband Details



Example 5: Consider the design of an HP filter with the normalized passband edges at $\omega_p = 0.99\pi$ and $\omega_S = 0.97\pi$, passband ripple $R_p = 0.25$ dB, and the minimum stopband attenuation A_S = -60 dB.

Direct application of the Parks-McClellan algorithm results in an HP filter of an order $N_H = 248$. Figure 24 shows its magnitude response. The proposed design steps are illustrated in the following.

1. Transform the HP specification into the LP ones.

$$\omega_p' = 0.01\pi$$
$$\omega_s' = 0.03\pi \qquad (17)$$

Figure 23. Stopband Details

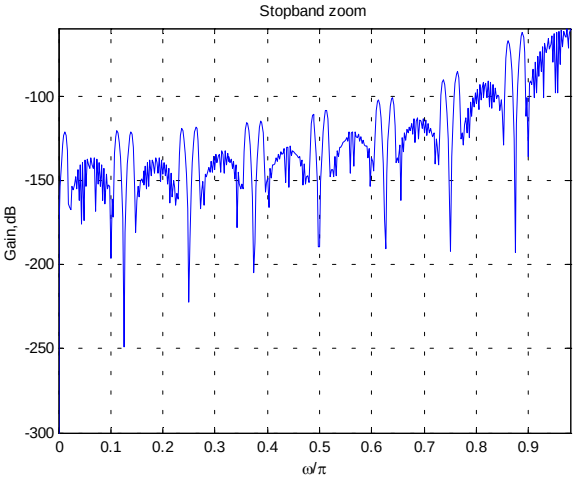
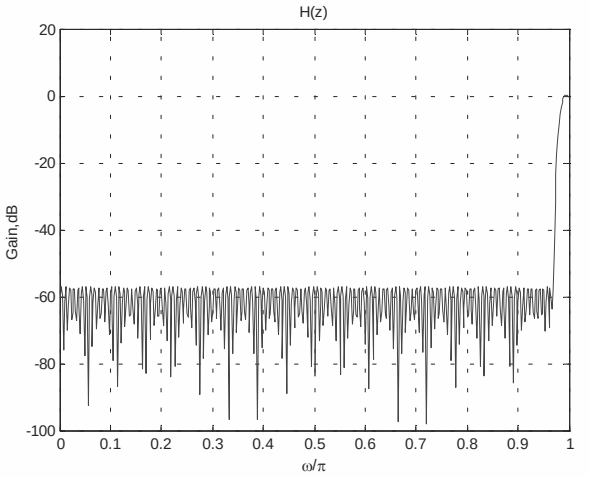


Figure 24. The Prototype Filter H(z)



2. Choose $M = 15$.
3. Design the LP filter $G(z)$. The order of the filter N_G is 19.
4. Expand the filter $G(z)$, $M = 16$ times. The magnitude response is shown in Figure 25.
5. Transform the expanded LP filter into the HP filter. This is illustrated in Figure 26.
6. Design the HP sharpening RRS filter, using $K = 3$. Figure 27 shows the corresponding magnitude response.
7. Cascade the HP expanded shaping filter and the HP sharpening RRS filter. The resulting magnitude response is shown in Figure 28.

Figure 25. Expanded LP Shaping Filter, M = 16

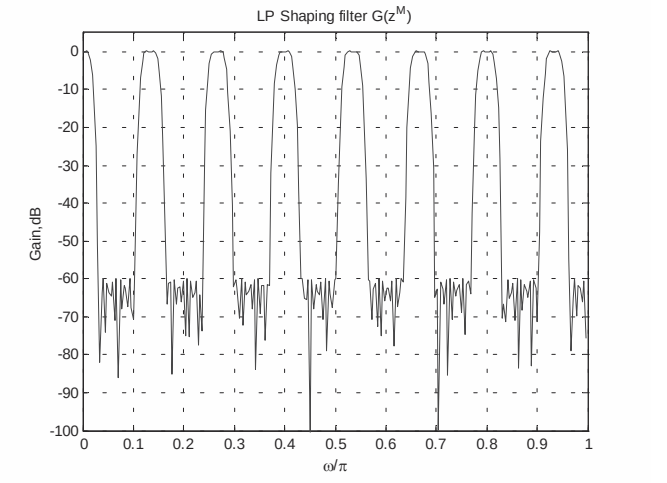
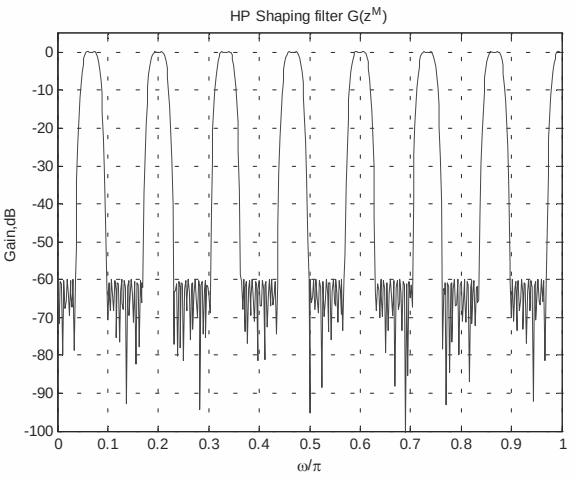


Figure 26. Expanded HP Shaping Filter, M = 16



8. Check if the specifications are satisfied. Indeed, Figures 29 and 30 show that this is the case.

This example also illustrates the efficiency of the proposed procedure. Instead of having to design a filter of an order 248, the shaping filter, whose order is only 19, has been designed. In addition, the interpolator is a sharpening RRS filter, which needs only three integer multiplications.

Figure 27. HP Sharpening RRS Filter, M = 15, K = 3

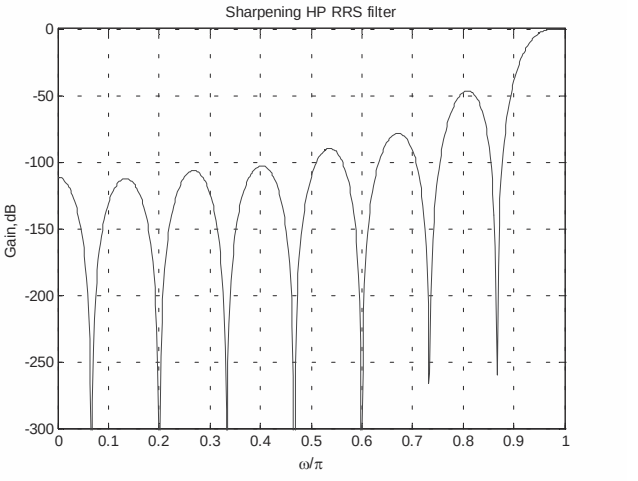
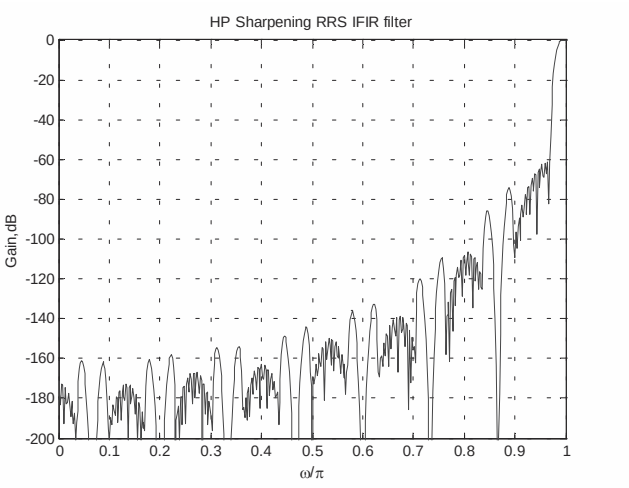


Figure 28. HP Sharpening RRS-IFIR Filter



CONCLUSION

The design of HP narrowband FIR filters has been presented. The method is based on the use of the IFIR structure, where the interpolator is the sharpening RRS filter. The RRS filter has an advantage because it requires no multipliers and no storage for the interpolation coefficients. The sharpening technique improves the passband and the stopband characteristics of an RRS filter. The overall result is the lower computational complexity of the resulting IFIR structure. The method is useful for narrowband HP filter design.

Figure 29. Passband Details

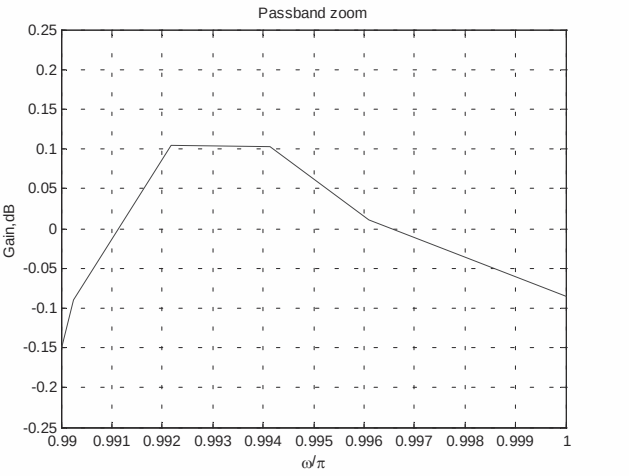
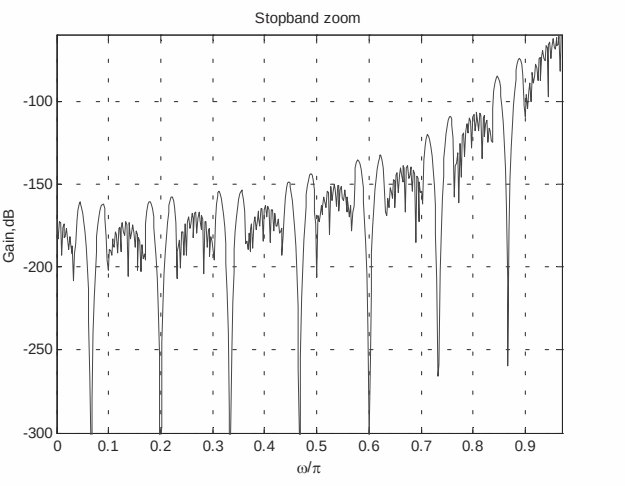


Figure 30. Stopband Details



REFERENCES

Bartolo, A., & Clymer, B. D. (1996, August 8). An efficient method of FIR filtering based on impulse response rounding. *IEEE Transaction on Signal Processing*, *46,* 2,243-2,248.

Jou, Y. D., Hsieh, C. H., & Kuo, C. M. (1997, August 4). Efficient Weighted Least-Square Algorithm for the Design of FIR Filters. *IEE Proc. Vis. Image Processing*, *144,* 244-248.

Kaiser, J., & Hamming, R. (1977, October). Sharpening the response of a symmetric nonrecursive filter by multiple use of the same filter. *IEEE Transaction on Acoustics, Speech, and Signal Processing*, *ASSP-25,* 415-422.

Kumar, B., & Kumar, A. (1999, February 2). Design of efficient FIR filters for amplitude response: $|1/\omega|$ by using universal weights. *IEEE Transaction on Signal Processing, 47,* 559-563.

Kwentus, A. Y., Jiang, Z., & Willson A. N. (1997, February 2). Application of filter sharpening to cascaded integrator-comb decimation filters. *IEEE Transaction on Signal Processing, 45,* 457-467.

Lian, Y., & Lim, Y. C. (1998, January 8). Structure for narrow and moderate transition band FIR filter design. *Electronic Letters*, *34*(1), 49-51.

Mitra, S. K. (2001). *Digital signal processing: a computer-based approach.* New York: McGraw-Hill, Inc.

Neuevo, Y., Dong, C-Y., & Mitra, S. K. (1984, June). Interpolated impulse response filters. *IEEE Transaction on Acoustics, Speech, and Signal Processing*, ASSP-23, 301-309.

Webb, J. L., & Munson, D.C. (1996, August 8). A new approach to designing computationally efficient interpolated FIR filters. *IEEE Transaction on Signal Processing,* 44, 1923-1931.

About the Authors

Joan Peckham is a professor of Computer Science at the University of Rhode Island, USA. In the 1990s she pursued semantic data modeling research in the context of active databases. She is currently interested in the application of these techniques to software engineering tools. She is concurrently working on a project to develop learning communities for freshman, with the goal of encouraging women and other underrepresented groups into the technical disciplines. She is also working with colleagues in engineering and the biological sciences to apply computing techniques to the fields of transportation and bioinformatics, respectively.

Scott J. Lloyd received his B.S. and M.S. in Management Information Systems from Virginia Commonwealth University. After that he worked for the Department of Defense for two years as a database analyst and designer. He then worked for Cap Gemini consulting and as a private consultant for another two years. Scott then received his Ph.D. in Information Systems from Kent State University. He then worked for Roadway technologies as a senior technology consultant. He accepted an assistant professor position from Eastern Illinois University and after three years, Scott moved to the University of Rhode Island, USA, as an assistant professor in Management Information Systems.

* * * *

Somjit Arch-int received an M.Sc. degree in Computer Science from the National Institute of Development Administration, Thailand, in 1990, and a Ph.D. in Computer Science from the Asian Institute of Technology, Thailand in 2002. He is currently an assistant professor in the Department of Computer Science, Khon Kaen University, Thailand. His previous experience includes the development of several industry systems and consulting activities. His research interests are business component-based software development, object-oriented metrics, ontology-based e-business modeling, knowledge-based representation, and semantic Web. He is a member of IEEE Computer Society.

Dentcho N. Batanov received the M.Sc. and Ph.D. degrees from the Technical University, Sofia, Bulgaria, in 1970, and 1975, respectively. He is currently an associate professor in the Program of Computer Science and Information Management at the Asian Institute of Technology, Thailand, where he was a coordinator of the program from 1996 to 1999. His research interests are application of information technologies in education, management, industrial engineering and manufacturing, knowledge-based and expert systems, object-oriented software engineering, distributed systems and technologies, component and framework-based software development, and e-business. He has published more than 100 papers in journals and conference proceedings, as well as numerous textbooks and manuals. He is currently a member of international board of several international journals and conferences. He is a member of the IEEE, and the ACM.

Robert Mark Bram was born in Melbourne, Australia, in 1974. He has completed a Network Computing Honors degree at Monash University, Melbourne, Australia, and is planning on pursuing a Ph.D., further developing the ideas in this work. Areas of particular interest are developing open source technologies with Java and Jini.

Jeff Brewer is an assistant professor in the School of Technology, Purdue University, USA. His areas of interest include systems analysis and design, computer-aided software engineering, rapid application development (RAD) and IT project management. Prior to joining the faculty, he worked for 19 years as a systems developer, manager and consultant in a variety of business environments.

T. Y. Chen is a professor of Software Engineering in the School of Information Technology, Swinburne University of Technology, Melbourne, Australia, where he is the director of Centre for Software Engineering (CSE). He received a Ph.D. in Computer Science from the University of Melbourne. His current research interests include software testing, maintenance and design.

Ajantha Dahanayake is an associate professor in the Department of Information and Communication Technology at the Faculty of Technology, Policy and Manage-

ment, Delft University of Technology, The Netherlands. She previously served as an associate professor in the Department of Information Systems and Algorithms at the Faculty of Information Technology and Systems. She received her B.Sc. and M.Sc. in Computer Science from the University of Leiden and Ph.D. in Information Systems from Delft University of Technology. She had served in a number of Dutch research and academic institutions. Her research interests are distributed Web-enabled systems, CASE, methodology engineering, component-based development and m-business. She is the research director of the research program Building Blocks for Telematic Applications Development and Evaluation (BETADE).

Javier Diaz-Carmona was born in México, on June 3, 1967. He received his B.S. in Electronics Engineering in 1990 from the Instituto Tecnológico of Celaya (ITC), Guanajuato, México; a Master of Science degree in Electronics in 1997 from the National Institute of Astrophysics, Optics and Electronics (INAOE), Puebla, México, where he is currently pursuing his doctoral studies. From 1990 to 1995 and from 1998 to 1999, he worked as full time professor in the Department of Electronics in the Instituto Tecnológico of Celaya. His main research interests are in digital signal processing, especially in digital filter design, multirate processing systems and DSP embedded applications.

Jana Dospisil is a lecturer at School of Network Computing, Monash University, Melbourne, Australia where she is lecturing in the field of object oriented design and programming in undergraduate and postgraduate courses. She holds a Ph.D. in Engineering from the Royal Melbourne Institute of Technology, Australia. Her research interests include object-oriented analysis and design, networked multimedia and mobile agent technology. She has published several papers in the area of separation of concerns concentrating on design and implementation of intelligent mobile agents.

Laura Felice is an assistant professor and researcher at the Department of Computer Science and Systems of the Universidad Nacional del Centro of Tandil, Argentina. She is a member of the Software Technology project of INTIA (Instituto de Investigación y Transferencia en Tecnología Informática Avanzada) of the same university. Her research interest is focused on formal software development. In particular, her work is oriented towards the definition of an integrated reuse process defined for all stages of software development.

Jesús Garcia-Consuegra graduated as an Ingeniero de Telecomunicación (Telecommunications Engineer) in 1992, and as a Doctor Ingeniero de Telecomunicación (Ph.D. in Telecommunications) in 1999, both from the E.T.S. Ingenieros de Telecomunicación of the University of Politécnica de Madrid, Spain. His research activity focuses on software engineering and databases. Since 1994, he has been a

member of the Computer Science Department (Departamento de Informática) at the Universidad de Castilla-La Mancha, Spain, and was made associate professor in 1999. He has also been head of the Research Group of Distributed Information System since 1999. He is actively involved as member and technical director in European and national projects, as well as contracts and agreements with private companies and public administrations. He has been an organizer of different national and international computer science events (congresses and workshops).

Rick Gibson has more than 20 years of software engineering experience using structured, object oriented and cleanroom software development methodologies. He is an authorized Software Engineering Institute (SEI) lead appraiser and an ASQ certified software quality engineer. Dr. Gibson he has extensive domestic and international experience in the conduct of SEI Capability Maturity Model evaluations and development of process maturity improvement and corrective action plans for evaluated organizations. He is currently an associate professor at American University, Washington, D.C., and serving as chair of the Department of Computer Science and Information Systems. His responsibilities, as a faculty member, include teaching graduate courses in software engineering, database systems, and data communication. He has published numerous books, book chapters and journal articles on software development and quality assurance.

Gordana Jovanovic-Dolecek received a B.S. from the Department of Electrical Engineering, University of Sarajevo, an M.Sc. from University of Belgrade, and a Ph.D. degree from the Faculty of Electrical Engineering, University of Sarajevo. She was with the Faculty of Electrical Engineering, University of Sarajevo, until 1993. She served in various positions including as a research assistant, assistant professor, associate professor full professor and chairman of the Department of Telecommunication (1986-1991). From 1993-95, she was with the Institute Mihailo Pupin, Belgrade, and then from 2001-2002 she was at the Department of Electrical & Computer Engineering, University of California, Santa Barbara, USA. In 1995 she joined the Institute INAOE, Department for Electronics, Puebla, México, where she works as a professor and researcher. She is the author of three books, editor of one book and author of more than 100 papers. Her research interests include digital signal processing and digital communications. She is a member of IEEE and The National Researcher System (SNI) Mexico.

Judith Kabeli is a doctoral student at the Department of Information Systems Engineering at Ben-Gurion University, Israel, where she earned her B.Sc. and M.Sc. degrees in Mathematics and Computer Science (1994). Prior to her Ph.D. studies, Kabeli held professional and managerial position at the Israel Aircraft Industries. Kabeli's research interests include data modeling, information systems analysis and design, and intelligent Web-based user interfaces. Her doctoral dissertation is

concerned with the development and testing of FOOM methodology that is described in Chapter VII, which she authored under the supervision of Professor Peretz Shoval.

D. C. McDermid has more than 10 years practical experience in the IT industry as well as many more years as an academic. His main research interests are in the area of systems modeling and systems thinking, particularly in how they can be applied to capturing information systems specifications. He is also interested in how consultants and other senior IT professionals can assist the discipline in terms of adding value to IT business processes. He holds a bachelor's degree with honors in Computing Science and an M.B.A. from the University of Glasgow, UK, and a Ph.D. in Information Systems from Curtin University of Technology in Australia and has links and memberships with several professional bodies.

John Mendonca is an assistant professor in the School of Technology, Purdue University, USA. He serves as the Computer Technology Department's chair of the graduate program and teaches graduate courses in information technology (IT) leadership and management. Prior to his teaching career, he worked as a systems developer, manager and consultant in the banking and insurance industries. His interests include strategic IT, management of IT, organizational impact of IT and software engineering.

Germán Montejano is a professor and researcher at the Universidad Nacional de San Luis, Argentina. He holds a degree in Computer Science from the Universidad Nacional de San Luis and is also a post-graduate student attending the university's M.Sc. Program in Software Engineering. He has experience in the information technology area.

Valentina Plekhanova is a senior lecturer in Computing at the School of Computing, Engineering and Technology at the University of Sunderland, UK. Plekhanova holds an M.Sc./M.Phil. in Applied Mathematics and Theoretical Mechanics from the Novosibirsk State University, Academgorodok, Russia. Her Ph.D. is in Application of Computer Technology, Mathematical Modelling and Mathematical Methods in Scientific Research, from the Institute of Information Technologies and Applied Mathematics, Russian Academy of Sciences. She held a number of research and lecturer positions in Russia and Australia. Plekhanova has international experience in lecturing on such subjects as software engineering, knowledge engineering, artificial intelligence, the theory of probability, computational science, and optimization. She was a consultant with P-Quant, Sydney, Australia. She was a project investigator in several international research and industrial projects in Russia and Australia. Her research results were published in international journals and conference proceedings. Her research interests include engineering the cognitive pro-

cesses, learning processes, machine learning, knowledge engineering, modeling intelligence in software systems, quantitative software project management, software process analyses and process improvement.

Vijay V. Raghavan is currently an associate professor of Information Systems at Northern Kentucky University, USA. His main research interests are in the areas of software development process and personnel issues relating to software developers. He has published in the *Journal of Information Technology Management* and *Journal of Systems Management* in addition to numerous presentations and proceedings' publications at national and international conferences. He has also consulted with Fortune 100 companies in the greater Cincinnati area.

Iyad Rahwan is a Ph.D. student at the Intelligent Agent Laboratory, Department of Information Systems, University of Melbourne, Australia. He received a Master of Information Technology degree from Swinburne University of Technology, Australia, and a B.Sc. (1st Class Honors) in Computer Science from the UAE University. His current research interests include multi-agent systems, electronic marketplaces and automated negotiation.

Daniel Riesco is a professor and researcher at Universidad Nacional de San Luis and Universidad Nacional de Río Cuarto, Argentina. Riesco holds a degree in Computer Science from Universidad Nacional de San Luis, Argentina, and an M.Sc. in Knowledge Engineering from Universidad Politécnica de Madrid, Spain. He is also a professor of the M.Sc. Program in Software Engineering at Universidad Nacional de San Luis, Universidad Nacional de Jujuy and Universidad Nacional de Catamarca, Argentina. Riesco is the director of a project at the National University of San Luis. He has made numerous research presentations in various national and international conferences. He served as the program coordinator of the SCITeA'02 for the International Association of Computer and Information Science (ACIS) and Central Michigan University, USA, and as program committee member of different conferences. Riesco is author or co-author of more than 60 publications in numerous refereed journals and conference proceedings. He has been working on research problems involving formal methods, UML, development process, workflow and software quality assurance.

Dan Shoemaker is a professor and the academic coordinator in the Computer and Information Systems program in the College of Business Administration at the University of Detroit Mercy, USA. He has been in software work since 1968 and he has had positions ranging from managing large IT operations to teaching. He founded the software management master's degree program (1991) at the University of Detroit Mercy. He is interested in all aspects of strategic process infrastructure and strategic management for IT.

Peretz Shoval is a professor of Information Systems and head of the Department of Information Systems Engineering at Ben-Gurion University, Israel. He earned his B.Sc. in Economics (1970) and M.Sc. in Information Systems (1975) from Tel-Aviv University, and Ph.D. in Information Systems (1981) from the University of Pittsburgh, USA, where he specialized in expert systems for information retrieval. In 1984 he joined Ben-Gurion University, where he started the Information Systems Program at the Department of Industrial Engineering and Management. Prior to moving to academia, he held professional and managerial positions in computer companies and in the IDF. Shoval's research interests include information systems analysis and design methods, data modeling and database design, and information retrieval and filtering. He has published numerous papers in journals and presented his research in various conferences. Shoval has developed various methodologies and tools for systems analysis and design, and for conceptual and logical database design.

Zoran Stojanovic is currently a Ph.D. researcher at the Faculty of Technology, Policy and Management, Delft University of Technology, The Netherlands. He received his Dipl. Eng. and M.Sc. in Computer Science from the Faculty of Electronic Engineering, University of Nis (Yugoslavia) in 1993 and 1998, respectively. His research interests are in the areas of component-based development, Web services and geographic information systems. He has been working since 1993 as a researcher and a teaching assistant in the fields of computer science and software engineering, with the University of Nis and then with the Delft University of Technology. He has been an author of a number of publications.

Roberto Uzal is a part-time professor both of Software Engineering at Universidad Nacional de San Luis (State University of San Luis) and of Information Technology, at Universidad de Buenos Aires (University of Buenos Aires), both in Argentina. He has been working on research problems involving software systems architecture, real-time software and software projects management. Uzal is also the local coordinator of the Master of Science Program on Software Engineering at Universidad National de San Luis (this Master of Science Program is directed by Professor Dines Bjorner of Denmark.). Uzal has made numerous teaching and research presentations in various national and international conferences, industries and teaching and research institutions in Argentina, México and the United States. He also served as session chair and tutorial chair at various international conferences. Uzal has experience as general manager of very important software projects. Some of his projects are mentioned as "success cases" in the Web pages of international software firms. Uzal is a member of the Argentine Society of Information Technology and Operational Research (SADIO) and the Software Engineering and Information Technology Institute (SEITI).

Gabriel Vilallonga is an assistant professor and researcher at the Universidad Nacional de San Luis, Argentina. Vilallonga holds a degree in Computer Science from the Universidad Nacional de San Luis and is also a post-graduate student attending the university's M.Sc. Program in Software Engineering.

Yun Yang is an associate professor in the School of Information Technology, Swinburne University of Technology, Melbourne, Australia, where he is the director of Centre for Internet Computing and E-Commerce (CICEC). He received a Ph.D. in computer science from the University of Queensland, Australia, and a master's degree in Computer Science from the University of Science and Technology of China. His current research interests include Internet- and Web-based computing technologies and applications, workflow environments for software development and e-commerce, and real-time CSCW and groupware systems.

Index

D

E

F

G

H

I

J

K

L

M

O

P

Q

R

S

T

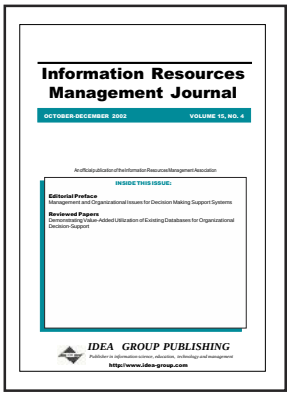
U

W

Information Resources Management Journal (IRMJ)

An Official Publication of the Information Resources Management Association since 1988

Editor:
Mehdi Khosrow-Pour, D.B.A.
Information Resources Management Association, USA



ISSN: 1040-1628; eISSN: 1533-7979
Subscription: Annual fee per volume (four issues): Individual US $85; Institutional US $265

Mission

The *Information Resources Management Journal* (IRMJ) is a refereed, international publication featuring the latest research findings dealing with all aspects of information resources management, managerial and organizational applications, as well as implications of information technology organizations. It aims to be instrumental in the improvement and development of the theory and practice of information resources management, appealing to both practicing managers and academics. In addition, it educates organizations on how they can benefit from their information resources and all the tools needed to gather, process, disseminate and manage this valuable resource.

Coverage

IRMJ covers topics with a major emphasis on the managerial and organizational aspects of information resource and technology management. Some of the topics covered include: Executive information systems; Information technology security and ethics; Global information technology Management; Electronic commerce technologies and issues; Emerging technologies management; IT management in public organizations; Strategic IT management; Telecommunications and networking technologies; Database management technologies and issues; End user computing issues; Decision support & group decision support; Systems development and CASE; IT management research and practice; Multimedia computing technologies and issues; Object-oriented technologies and issues; Human and societal issues in IT management; IT education and training issues; Distance learning technologies and issues; Artificial intelligence & expert technologies; Information technology innovation & diffusion; and other issues relevant to IT management.