Towards Temporal Information in Workflow Systems

Carlo Combi[1] and Giuseppe Pozzi[2]

[1] Università di Verona, strada le Grazie 15 I-37134 Verona - Italy
combi@sci.univr.it
[2] Politecnico di Milano, P.za L. da Vinci 32 I-20133 Milano - Italy
giuseppe.pozzi@polimi.it

Abstract. A workflow management system (WfMS) is a software system that supports the coordinated execution of different simple activities, assigning them to human or automatic executors, to achieve a common goal defined for a business process. Temporal aspects of stored information cannot be neglected and the adoption of a temporal database management system (TDBMS) could benefit.
By this paper we scratch the surface of the topic related to the use of a TDBMS in a WfMS, identifying some advantages in managing temporal aspects by a TDBMS inside some of the components of a WfMS. E.g., queries to reconstruct the schema of the business process or to assign activities to executors balancing their workload over time, or the definition of constraints among tasks can benefit from the use of a TDBMS.

1 Introduction

Workflows are activities involving the coordinated execution of multiple tasks performed by different processing entities. A worktask (or task) defines some work to be done by a person, by a software system or by both of them. Specifying a workflow involves describing those aspects of its component tasks (and the processing entities that execute them) that are relevant to control and coordinate their execution, as well as the relationships between the tasks themselves.

Information needed to run a workflow are stored by a database management system (DBMS). In most cases, the adopted DBMS is a traditional, off-the-shelf, relational DBMS which does not provide any facility to manage temporal aspects of stored information, forcing the designer to explicitly define and manage those aspects. Temporal information to be managed in a workflow involves several aspects, such as the starting and ending timestamps of a task, the deadline for completing an activity, the validity time of a data item, the time interval between the execution of two different activities, to mention few of them.

With regards to these temporal aspects, the adoption of a temporal DBMS (TDBMS), which is based on a data model where time and time dimensions are suitably represented and managed [12], could easily improve the development of a system devoted to the automatic management of workflows (workflow management system: WfMS). Indeed, the adoption of a temporal data model and of a related temporal query language, could help to:

A. Olivé et al. (Eds.): ER 2002 Ws, LNCS 2784, pp. 13–25, 2003.

- manage in a homogeneous way information related to different aspects of a workflow system, providing a uniform data model to all the software components of a workflow system;
- express in a general way those temporal aspects that are not application-dependent, thus allowing the focus on specific issues of the application when designing a workflow;
- allow a powerful specification of workflows, where application-dependent temporal aspects of real world situations can be simply expressed.

This paper provides a first analysis of specific temporal issues of workflow systems which could be suitably faced by the adoption of a TDBMS; more specifically, we first consider temporalities of information modeled in workflow systems and then examine the temporal issues related to the management of this information during the automatic execution of workflows. In this paper we shall adopt a generic temporal relational data model and the widely known temporal query language TSQL2 [13], to show how to model and manage different temporalities in WfMSs. Throughout the paper, we shall mainly consider the two basic temporal dimensions, namely valid and transaction times, which have been extensively considered in the temporal database literature [12]; more recent topics in the temporal database area will be mentioned for specific needs of WfMSs.

In the following, Section 2 provides some basic concepts on workflow systems, a motivating example, and a description of used models. Section 3 discusses the workflow models which require temporal dimensions, and describes the components of a workflow system that could greatly benefit from the adoption of a TDBMS, mentioning also relevant related work. Finally, Section 4 sketches out some conclusions and general remarks.

2 Workflow Management Systems - WfMSs

Workflow Management Systems (WfMSs) are software systems that support the specification of business processes (described by their *process model* or *schema*), the execution of process instances (*cases*) described in terms of activities (*tasks*) and of dependencies among tasks. A business process may refer to the management of a car rental company, of a travel agency, of an assurance company or even a bank loan. WfMSs control process executions by scheduling activities and by assigning them to executing agents. WfMSs are very complex systems and include many different components, such as: the interpreter of the process definition language (PDL) used to formally model the business process; the process model designer, which helps the user to suitably define a process model according to the supported PDL; the resource management unit, to assign tasks to executing agents; the database connectivity unit, to access data stored into a DBMS; the transaction manager; and the e-mail feeder.

2.1 A Motivating Example

Throughout the paper, as motivating example of a business process that can be managed by a WfMS, we shall refer to a car rental company: Figure 1 depicts

the car rental example. Although the semantics of the graphical representation is very intuitive, we refer to the conceptual model described in [5].

A new case is started as the car rental company receives a reservation request. The task GetRentalData collects customer's data and pick-up and return date and place. Next, ChooseCar specifies the type of car the customer prefers (Ferrari, Cadillac, ...). The task CheckCarAvailability queries the database and verifies whether the specified car is available by defining a value for the variable Available. According to the value of Available observed by routing task R1, the outgoing left arch may be followed, leading to the task RejectReservation which informs the customer about failure in reserving the car: otherwise, if the car is available, the task MakeReservation performs the reservation and informs the customer, while SendConfirmation sends the customer a mail with reservation details.

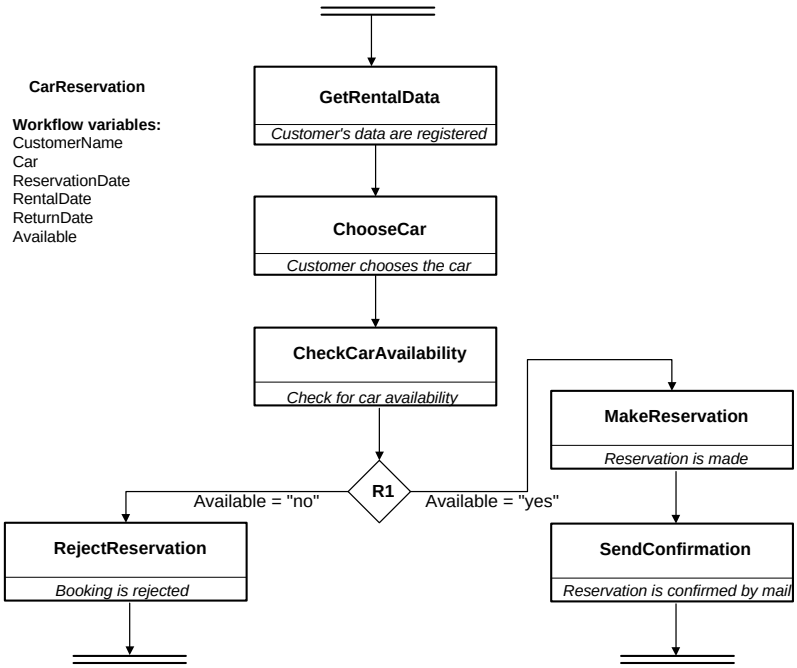

Fig. 1. Schema for the CarReservation process

2.2 Workflow Models

A WfMS stores all the information it needs inside a database managed by a DBMS. Several models can be defined for a WfMS considering the organization where the process is enacted and managed, the formal description of the process model, the data used by the process, the exception model to manage expected events which may force a deviation from the main flow [6], the transaction model

to manage aspects of concurrent access to shared data [5]. In the following, we categorize some of the models we shall refer to, as defined in [5].

The Organizational Model The organizational model formally depicts the organization of the company where the process is being enacted. Information mainly relates to agents, owned skills and hierarchy inside the company: all this information is stored by suitable database tables, whose structure is independent both from the organization and from the application domain. Every agent belongs to a group: a group collects agents either involved in the same project or working in the same geographical area. Every agent has a function (or role) specifying the skills the agents owns and the tasks he/she can be assigned, and a supervisor. A generic description of the organizational model includes several tables: however, for clarity reasons, we shall assume that all the information are stored into one unique table, namely Agent, storing the name and the Id of the agent, the e-mail address, the role owned, the group the agent belongs to, and the Id of the manager for that agent. The structure of the table, with some tuples for the CarRental process, is the following:

Agent	AgentName	AgentId	E-mail	Role	Group	ManagedBy
	Hernest	H03	hernest@mailer	PhoneOperator	Boston	B08
	Rudy	R01	rudy@mailer	PhoneOperator	Boston	B08
	Laura	L05	laura@mailer	PhoneOperator	Harvard	B08
	Ken	K13	ken@mailer	PhoneOperator	Harvard	B08
	Doug	D10	douglas@mailer	CarExpert	Syracuse	B08
	Bob	B08	mass_dealer@mailer	LocalDealer	Boston	B01
	Luke	B01	boss@mailer	CEO	Arlington	B01

The Process Model The process model describes the schema of the process; all the information is stored inside suitable database tables whose structure is statically determined and independent from the application domain. At process model design time, the process engineer specifies which is the first task to be executed, the subsequent tasks, the conditions to be evaluated to select different outgoing arcs, the criteria used to assign tasks to agents.

Namely, the tables of the process model are: Workflow, storing the names of the process models registered into the WfMS with their respective first task to be activated; Worktask, storing the names of the tasks of all the process models and the required role for the executing agent; RoutingTask, describing the type of all the routing tasks of any process model; Next, storing the successor task for any task of any process model; AfterFork, storing the criteria adopted to activate arcs outgoing from a routing task of any process model. Their structures with some tuples related to the process model CarReservation are the following:

Workflow	SchemaName	StartTask
	CarReservation	GetRentalData

WorkTask	SchemaName	TaskName	Role
	CarReservation	GetRentalData	PhoneOperator
	CarReservation	ChooseCar	CarExpert
	CarReservation	CheckCarAvailability	InformationSystem
	CarReservation	RejectReservation	PhoneOperator
	CarReservation	MakeReservation	PhoneOperator
	CarReservation	SendConfirmation	InformationSystem

RoutingTask	SchemaName	RTName	Type
	CarReservation	R1	mutualex_fork

Next	SchemaName	TaskName	NextTask
	CarReservation	GetRentalData	ChooseCar
	CarReservation	ChooseCar	CheckCarAvailability
	CarReservation	CheckCarAvailability	R1
	CarReservation	MakeReservation	SendConfirmation
	CarReservation	SendConfirmation	end_flow
	CarReservation	RejectReservation	end_flow

AfterFork	SchemaName	ForkTask	NextTask	Cond
	CarReservation	R1	RejectReservation	Available = "no"
	CarReservation	R1	MakeReservation	Available = "yes"

The Information Model. The information model describes the application domain data the process has to manage and the history of different cases.
Process Specific Data. The structure of process-specific tables is defined at schema design time and tables are automatically derived. Even though in most cases several tables are used, for clarity reasons in the following we assume that all the data about the cases of the process model CarReservation is stored within one single table: tables can eventually be reconstructed as a view. The table RentalData stores the process variables defined at process design time: case identifier, customer's name, date when the reservation was made, date of picking up the car, date for returning the car, and availability of the car as a result of the reservation. This latter variable is used by routing task R1 to activate the successor task (RejectReservation or MakeReservation). Table RentalData, with 2 runs of the CarReservation process, is depicted as:

RentalData	CaseId	CName	Car	ReservationDate	RentalDate	ReturnDate	Available
	50	Marple J.	Cadillac	01-03-09	01-12-01	02-01-06	"yes"
	51	Wallace E.	Lamborghini	01-07-11	02-02-01	02-02-02	"no"

Historical Data. Historical data contains information related to the execution of tasks and cases and is independent from the application domain. A generic description of historical data is obtained by the tables CaseHistory (storing the identifier of all the executed cases, the name of their respective process model, the responsible agent, which in most cases is the starting agent, and temporal information about the start time and the duration of the entire case) and TaskHistory (storing the name of the task, the identifier of the case the task belonged to, the name of the executing agent, the final status of the task and the time of start and end of the task itself). Their structures with some tuples from the process model CarReservation are the following:

CaseHistory	CaseId	SchemaName	Responsible	StartTime	Duration
	47	CarReservation	H03	01-03-09 10:03:10	8 min 40 sec
	48	CarReservation	L05	01-07-11 9:00:10	3 min 49 sec

TaskHistory	CaseId	TaskName	StartTime	EndTime	FinalState	AgentId
	47	GetRentalData	01-03-09 10:03:10	01-03-09 10:05:50	Completed	H03
	47	ChooseCar	01-03-09 10:06:00	01-03-09 10:08:50	Completed	D10
	47	CheckCarAvailability	01-03-09 10:08:51	01-03-09 10:08:52	Completed	auto
	47	MakeReservation	01-03-09 10:08:53	01-03-09 10:08:55	Completed	R01
	47	SendConfirmation	01-03-09 10:09:10	01-03-09 10:11:50	Completed	auto
	48	GetRentalData	01-07-11 9:00:10	01-07-11 9:01:43	Completed	L05
	48	ChooseCar	01-07-11 9:01:45	01-07-11 9:02:01	Completed	D10
	48	CheckCarAvailability	01-07-11 9:02:03	01-07-11 9:02:07	Completed	auto
	48	RejectReservation	01-07-11 9:02:10	01-07-11 9:03:59	Completed	K13

3 Temporal Aspects in WfMS

In this section, we show that several aspects managed by a WfMS have relevant temporal dimensions. We first introduce some examples of temporalities we need to consider when representing and storing information related to the different models of Section 2.2; then, we show how this information should be managed by the WfMS through a TDBMS.

3.1 Temporalities of the Models

Several temporal aspects of data could be managed by a TDBMS, providing a unified support for all the models the WfMS needs. In the following we shall focus on the application to workflow models of those temporal dimensions which are widely recognized as the fundamental temporal dimensions of any fact relevant to a database [12]: valid and transaction times. The *valid time* (VT) of a fact is the time when the fact is true in the considered domain, while the *transaction time* of a fact is the time when the fact is current in the database.

Temporality in the Organizational Model. In defining the organizational model, temporal aspects have to be considered: indeed, we need to represent when agents are available to perform any task suitable for their roles. Let us consider the case of human agents in the CarReservation process: each agent is working only during certain hours of the day and only during some days of the week; furthermore, each agent spends some holidays out of work during the year. All these aspects have to be managed in any case by the WfMS. The adoption of an organizational model which explicitly considers these temporal features can improve the usability of the system by the workflow designer, which can suitably represent the temporal constraints of the organization under consideration. As an example, in the following table we describe when a given agent is available.

AgentAvailability	AgentId	VT
	H03	working days in $[00\text{-}08\text{-}09 \div +\infty]$
	R01	mondays, tuesdays in $[00\text{-}01\text{-}05 \div +\infty]$
	L05	working mornings in $[00\text{-}05\text{-}07 \div +\infty]$
	K13	working days in $[01\text{-}01\text{-}23 \div +\infty]$
	D10	working days in $[01\text{-}01\text{-}09 \div +\infty]$
	B08	working days in $[01\text{-}11\text{-}11 \div +\infty]$
	B01	all days in $[00\text{-}01\text{-}03 \div +\infty]$

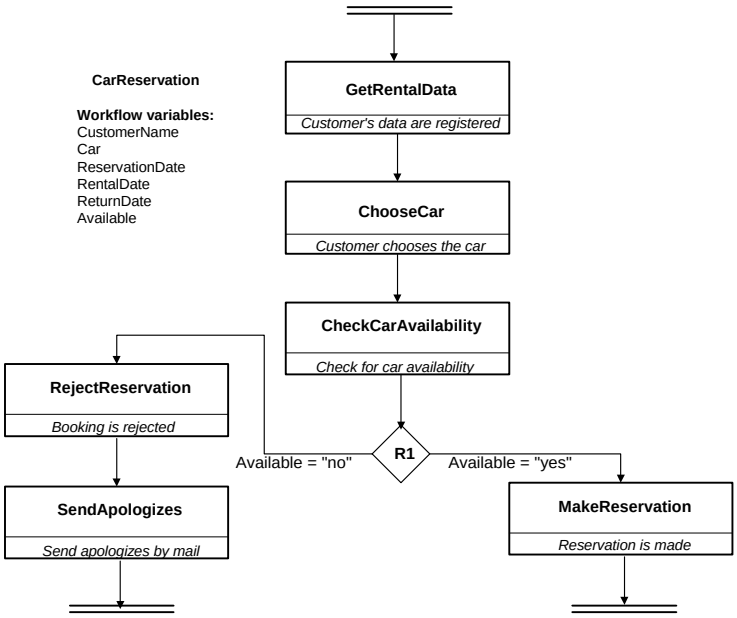

Fig. 2. Schema for the CarReservation process after a change to schema has been applied

In the previous example we assume we are able to express valid time according to different granularities, suitably defined and managed by the TDBMS. Several works on time granularity have been done within the temporal database community, which should be deeply considered in modelling this kind of temporal data for a WfMS [1,8,13,10].

Temporality in the Process Model. In the process model we mainly distinguish two different temporal aspects: versioning and temporal constraints.

Versioning. A process can have different versions, which result from the modification of the process schema due to the evolution of the application needs: for example, we can imagine that the schema of the process CarReservation, defined on January 5th, 2000, is that one depicted in Figure 1 until October 12^{th} 2001, when it is modified as depicted in Figure 2. As a difference from the previous version of the schema, if the reservation succeeds, the customer is no longer sent a confirmation: if the reservation fails, the customer is informed and an apologizing mail is sent along with some advertising flyers (task SendApologizes). Each version of the process model has, thus, a valid time, which identifies the temporal interval during which the given version was used. Tables WorkTask and Next, which are now valid time relations, are suitably updated according to the changes to the schema as follows:

WorkTask	SchemaName	TaskName	Role	VT
	CarReservation	GetRentalData	PhoneOperator	[00-01-05 ÷ +∞]
	CarReservation	ChooseCar	CarExpert	[00-01-05 ÷ +∞]
	CarReservation	CheckCarAvailability	InformationSystem	[00-01-05 ÷ +∞]
	CarReservation	RejectReservation	PhoneOperator	[00-01-05 ÷ +∞]
	CarReservation	MakeReservation	PhoneOperator	[00-01-05 ÷ +∞]
	CarReservation	ConfirmReservation	InformationSystem	[00-01-05 ÷ 01-10-12]
	CarReservation	SendApologizes	InformationSystem	[01-10-13 ÷ +∞]

Next	SchemaName	TaskName	NextTask	Tcons	VT
	CarReservation	GetRentalData	ChooseCar	[0 m ÷ 2 m]	[00-01-05 ÷ +∞]
	CarReservation	ChooseCar	CheckCarAvailability	[0 m ÷ 10 m]	[00-01-05 ÷ +∞]
	CarReservation	CheckCarAvailability	R1	[0 m ÷ 1 m]	[00-01-05 ÷ +∞]
	CarReservation	MakeReservation	ConfirmReservation	[0 m ÷ 1 m]	[00-01-05 ÷ 01-10-12]
	CarReservation	ConfirmReservation	end_flow	[0 m ÷ 0 m]	[00-01-05 ÷ 01-10-12]
	CarReservation	RejectReservation	end_flow	[0 m ÷ 0 m]	[00-01-05 ÷ 01-10-12]
	CarReservation	MakeReservation	end_flow	[0 m ÷ 0 m]	[01-10-13 ÷ +∞]
	CarReservation	RejectReservation	SendApologizes	[0 m ÷ 2 m]	[01-10-13 ÷ +∞]
	CarReservation	SendApologizes	end_flow	[0 m ÷ 0 m]	[01-10-13 ÷ +∞]

Temporal Constraints. A process model includes the definition of temporal constraints among the different tasks and their durations [1]. For instance, the column labelled Tcons in table Next defines a constraint for the scheduler of the WfMS, specifying that the successor task is to be scheduled, after the current task finished, within the minimum and maximum time stated by Tcons. Similarly, also time constraints for the maximum allowable task duration can be defined for every task, serving as temporal constraints on the execution time. TDBMS should be able to represent temporal constraints even with different temporal granularities, as discussed in [1,2].

Temporality in the Information Model. Several different temporal dimensions can be considered for data inside the information model; in this context, valid and transaction times are very important, managing both the history of workflow data and of possible changes on entered data. As an example, the table RentalData, managed by a bitemporal database system [12], allows the WfMS to discover changes both in the customer preferences and in entered data.

The table RentalData presents two situations. In the first one, customer Smith for case 47 modified his reservation. In fact, the customer named Smith on November 12*th* 2001 made the reservation for a car to be returned on January 6*th* 2002 (see the first tuple with CaseId=47). The valid time for the case started on November 12*th* 2001. On November 21*st* 2001 he changed the return date to January 7*th* 2002 - thus forcing the same case to be still opened and closing the first transaction for case 47 - and asked for a new car type FIAT.

In the second situation, data originally entered were wrong. Customer Jones did not change anything about his reservation; there has been only an error in data insertion (collection). Indeed, in this case, there is only one tuple about Jones in the current database state (i.e., the state consisting of tuples having the transaction time with $+\infty$ as upper bound): valid time of this tuple confirms that the reservation of Jones was valid from its definition up to now.

The table RentalData is the following:

CaseId	Cname	Car	RentDate	RetDate	Avlb	VT	TT
47	Smith J.	Ferrari	01-12-01	02-01-06	yes	[01-11-12 ÷ +∞]	[01-11-12 ÷ 01-11-21]
48	Jones J.	Lotus	02-02-01	02-02-02	no	[01-08-02÷ +∞]	[01-08-02 ÷ 01-08-03]
47	Smith J.	Ferrari	01-12-01	02-01-06	yes	[01-11-12 ÷ 01-11-21]	[01-11-21 ÷ +∞]
47	Smith J.	Fiat	01-12-01	02-01-07	yes	[01-11-21 ÷ +∞]	[01-11-21 ÷ +∞]
48	Jones J.	Lotus	02-02-01	02-02-03	no	[01-08-02÷ +∞]	[01-08-03÷ +∞]

As for historical data of the information model, by a TDBMS we are able to manage in a homogeneous way temporal data already present into the database and explicitly managed by the WfMS as user-defined times. Indeed, tables CaseHistory and TaskHistory depicted in Section 2.2 represent the valid time of stored information by attributes StartTime, Duration and StartTime, EndTime, respectively; being these attributes user-defined times [12], the WfMS has to deal with them explicitly. Using a TDBMS, instead, these two tables would be simply valid time relations, directly and homogeneously managed be the temporal database system. Instances of tables CaseHistory and TaskHistory, when represented in a temporal database system as valid time relations, are depicted in the following:

CaseHistory	CaseId	SchemaName	Responsible	VT
	47	CarReservation	H03	[01-11-12 10:03:10 ÷ 01-11-12 10:11:50]
	48	CarReservation	L05	[01-08-02 9:00:10 ÷ 01-08-02 9:03:59]

TaskHistory	CaseId	TaskName	FinalState	Agent	VT
	47	GetRentalData	Completed	H03	[01-11-12 10:03:10 ÷ 01-11-12 10:05:50]
	47	ChooseCar	Completed	D10	[01-11-12 10:06:00 ÷ 01-11-12 10:08:50]
	47	CheckCarAvailability	Completed	auto	[01-11-12 10:08:51 ÷ 01-11-12 10:08:52]
	47	MakeReservation	Completed	R01	[01-11-12 10:08:53 ÷ 01-11-12 10:08:55]
	48	GetRentalData	Completed	L05	[01-08-02 9:00:10 ÷ 01-08-02 9:01:43]
	48	ChooseCar	Completed	D10	[01-08-02 9:01:45 ÷ 01-08-02 9:02:01]
	48	CheckCarAvailability	Completed	auto	[01-08-02 9:02:03 ÷ 01-08-02 9:02:07]
	48	RejectReservation	Completed	K13	[01-08-02 9:02:10 ÷ 01-08-02 9:03:59]

Observation. *To reconstruct the complete history of a case, it could be useful to distinguish at the modelling level both the time during which the task has been executed, i.e., the valid time of the task, and the time at which the task has been assigned to the agent. This latter temporal dimension could be suitably modeled by the concept of event time, which is the occurrence time of a real-world event that either initiates or terminates the validity interval of the modeled fact [7].*

3.2 Managing Temporal Aspects

All the components of a WfMS heavily interact with the DBMS to store and reference all the data from the different models (organizational, process and information models). If the adopted DBMS does not feature the management of temporal information, all the components of the WfMS have to explicitly manage those temporal aspects of data: on the other hand, if a TDBMS is adopted, temporal aspects of data are directly managed by the DBMS itself. Without loss of generality, in the following we shall use the widely known temporal query

language TSQL2 [13], to show how the interaction of the WfMS with a TDBMS could be powerfully performed by a temporal query language. In the following, we briefly outline some of the components of a WfMS that could take the greatest advantages from the adoption of a TDBMS.

Process Instancer: This component defines the proper process model to be adopted for all the running cases and also for completed ones. Obviously, if no change has never been applied to the model of a given process, there is no need to manage temporal aspects related to schema migration of the process model itself: however, a very very small percentage of process models (a good reasonable estimation is less than 1%) does not need for a change during their life, not even for adaptive, perfective or corrective maintenance [4]. On the other hand, if the process model has been changed, one may want to reconstruct the exact process model defined for all the cases, no matter if completed or not. For completed cases, the table CaseHistory defines only the executed tasks, while the table does not allow one to completely reconstruct the schema: in fact, if in front of a routing operator with three or more outgoing arcs one of them only was followed, the table CaseHistory does not specify which outgoing arcs were available but only the followed outgoing arcs. The complete reconstruction of the process model is thus not feasible by the workflow history only.

Assuming that a case evolves according to the process model valid at the time of its initiation [4], the temporal query in TSQL2 [13] to reconstruct the process model for any case should sound like:

```
SELECT CaseId, TaskName, NextTask
FROM Next N, CaseHistory C
WHERE C.SchemaName = N.SchemaName AND
      VALID(N) OVERLAPS BEGIN(VALID(C))
```

On the other hand, whenever a new case is started, the process instancer has to define which is the current process model to be followed for the starting case. The TSQL2 query to reconstruct the *current* process model for CarReservation, if many different versions are defined, should sound like:

```
SELECT TaskName, NextTask
FROM Next N
WHERE N.SchemaName = "CarReservation" AND
       VALID(N) OVERLAPS CURRENT_DATE
```

In both the described situations, the reconstruction of the correct process model would be completed by considering the information about routing tasks stored in the table AfterFork, with the same approach previously described.

Scheduler of the Workflow Engine: This component assigns tasks to agents for execution based on their skills. The component has to mainly consider working time and work load of agents, which are very critical and may vary continuously.

Working time: the scheduler should not assign to an agent a task whose expected completion, computed as the maximum task duration started from the initiation of that task, will fall into or will include an holiday period for that agent. Additionally, holiday time for agents is in most situations made of a set of intervals, e.g., agent Bob will be on holiday on 02-04-04÷02-04-11 and on

02-05-01÷02-05-06. If the task is expected to start on 02-04-02 and the expected duration is 3 days, the agent will suspend the execution of the task on 02-04-03 (beginning of holiday) and will resume its execution at the end of the holiday on 02-04-11, resulting in a duration of the execution of that task of approximatively 10 days, much beyond the expectancy of 3 days. The scheduler, thus, has to compare intervals of holidays of agents and expected execution time of tasks by suitable techniques, to be able to manage comparisons between intervals at different levels of granularities and/or with indeterminacy [1,8]. In this direction, Bettini and colleagues in [1] study the problem of specifying, even with different granularities, time constraints for tasks and for delays between tasks; they also propose a general algorithm for finding *free schedules*, i.e. schedules which do not impose further constraints on the task durations, for a workflow.

Work Load: The scheduler has to assign tasks to agents balancing the load of work among them. When evaluating the history of tasks executed by agents, the scheduler could consider, for example, whether there is some agent which is not working when some other agent (with the same role) is working. Let us consider the following TSQL2 query:

```
SELECT A.AgentId, TaskName
FROM AgentAvailability A, TaskHistory T, Agent G1, Agent G2
WHERE A.AgentId <> T.Agent AND A.AgentId = G1.AgentId AND
   T.AgentId = G2.AgentId AND G1.Role = G2.Role AND
   NOT EXISTS (SELECT *
      FROM TaskHistory T1
      WHERE T1.Agent=A.AgentId AND VALID(T1) OVERLAPS VALID(T))
```

Through this query, by using the default semantics of TSQL2 in the `FROM` clause, tuples of tables `AgentAvailability`, `TaskHistory`, and `Agent` are associated only if their valid times intersect (in this case, tuples of atemporal tables are considered as always valid); the condition expressed in the `WHERE` clause is then evaluated on these tuples. Valid times of tuples in the result are the intersection of the tuples considered in the evaluation of the query.

Process Model Designer Tool: The process model engineer uses a tool, namely workflow designer, to formally define the schema inside the WfMS. The process engineer has to define for every task an expected task duration (i.e., the average duration for a task) and the maximum allowable duration (i.e., a deadline for the completion of the task): if the execution of a task exceeds the maximum duration, the agent has to be urged to complete the task and if the delay persists, a compensation action could possibly be executed. The tool also has to enable the engineer in defining temporal constraint (e.g., through suitable values of the attribute Tcons in table Next of Section 3.1) over dependencies among tasks, identifying the maximum allowable interval between the completion of a predecessor task and the activation of its successor(s): these constraints must be observed by the workflow scheduler.

Exception Designer Tool: The process model engineer, after having defined the process model, has to define *expected exceptions* [6,11], i.e., those anomalous situations that are part of the semantics of the process and that are known in advance at workflow design time. Exceptions are an important component

of workflow models and permit the representation of behaviors that, although abnormal, occur with high frequency (sometimes even more than 10% of the times). Examples of expected exceptions are the violation of either constraints over data or temporal conditions, a car accident in a car rental process, as well as a change in the RentalDate or in the ReturnDate: if the customer postponed the ReturnDate, some compensation actions must be performed, e.g., re-arranging the subsequent rentals of the same car to other customers. Exception management modules in a WfMS are periodically invoked, sometimes with a frequency ranging from tens of minutes to several hours: such a frequency considers that business processes are long-running activities [9] and in most cases there is no need for an immediate management of exceptions. These modules manage exceptional situations that may have occurred during the process enactment since the previous activation of the modules themselves [6] and consider for execution the several exceptions occurred to different cases, grouping them by exception type: e.g., all the task instances that for any reason exceeded their maximum allowable task durations are processed altogether. In order to select all the exceptions to be processed, a temporal query must be performed to select all the exceptions that actually occurred since the last execution of the exception manager: the use of a TDBMS could easily help.
Additionally, a TDBMS can help in retrieving values of process specific data at given instants. Let us assume that we need an exception detecting whether the customer switched from a car to another car. The exception mechanism has to compare the currently booked car with the previously booked one. The event of the exception is a modify of an attribute, namely Car: the condition part has to monitor the value of Car with its previous value. By a TDBMS the query to reconstruct the changes to the Car attribute sounds like the following:

```
SELECT SNAPSHOT R1.Car, R2.Car
FROM RentalData R1, RentalData R2
WHERE R1.CaseId = R2.CaseId AND R1.Car <> R2.Car AND
   VALID(R1) BEFORE VALID(R2) AND
   NOT EXISTS (SELECT *
      FROM RentalData R
      WHERE R.CaseId = R1.CaseId AND VALID(R) AFTER VALID(R1)
         AND VALID(R) BEFORE VALID(R2))
```

It can be easily observed that if the adopted DBMS is not a temporal one, the query would result in a more specific expression: indeed, the attributes representing the valid time would be treated as application-dependent ones. Furthermore, the query would be executed without any special support by the DBMS.

4 Conclusions

In this paper, we showed how temporal database systems applied to WfMS could greatly improve performances, e.g. by allowing one to define different versions of the same process model, picking the last one for new cases to be started and using the original one valid at the time the already running case was started,

and ease-of-usage of WfMSs, e.g. by storing the subsequent changes to process-specific data. We also showed how to model and query in a homogeneous way temporal information concerning different models needed by WfMSs.

The management of temporal information in WfMSs is an underestimated research topic, where research results from the temporal database area could be extensively applied and suitably extended. To this regard, we are going to consider the following possible future research directions:

Design of visual tools for process model design: the process engineer can define real world temporal constraints among tasks and on task/case durations, possibly by a powerful and easy-to-use tool;

Exception modelling: the process engineer can define suitable compensation actions if deadlines and/or temporal constraints are violated. Compensation actions, on the other hand, can be immediately executed, delayed, or temporally grouped, depending on the violated constraints and on the specific process.

References

1. Bettini C., Sean Wang X., Jajodia S. Free Schedules for Free Agents in Workflow Systems. Seventh International Workshop on Temporal Representation and Reasoning, TIME 2000, IEEE Computer Society, 31–38
2. Brusoni V., Console L., Terenziani P., Pernici B. Qualitative and Quantitative Temporal Constraints and Relational Databases: Theory, Architecture, and Applications. *IEEE TKDE* 1999, 11(6): 948–968
3. Casati F., Ceri S., Pernici B., Pozzi G. Deriving Production Rules for Workflow Enactment. In: Proceedings of the 7th Database and Expert Systems Applications International Conference, Springer-Verlag, LNCS, 1996, 94–115
4. Casati F., Ceri S., Pernici B., Pozzi G. Workflow Evolution. *Int. Journal Data and Knowledge Engineering* 1998, 24(1): 211–239
5. Casati F., Pernici B., Pozzi G., Sánchez G., Vonk J. Conceptual Workflow Model. In: *Database Support for Workflow Management*, Dordrecht, Kluwer Ac., 1999, 23–45
6. Casati F., Ceri S., Paraboschi S., Pozzi G. Specification and Implementation of Exceptions in Workflow Management Systems. *ACM TODS* 1999, 24(3): 405–451
7. Combi C., Montanari A. Data Models with Multiple Temporal Dimensions: Completing the Picture. In: Advanced Information Systems Engineering, 13th International Conference, CAiSE 2001, Berlin, Springer, LNCS, 2001, 187–202
8. Combi C., Pozzi G. HMAP - A temporal data model managing intervals with different granularities and indeterminacy from natural language sentences. *VLDB Journal* 2001, 9(4): 294–311
9. Dayal U., Hsu M., Ladin R. Organizing long-running activities with triggers and transactions. SIGMOD Record, 1990, 19(2): 204–214.
10. Dyreson C. E., Evans W. S., Lin H, Snodgrass R. T.: Efficiently Supported Temporal Granularities. *IEEE TKDE* 2000, 12(4): 568-587 (2000)
11. Eder J., Liebhart W. The Workflow Activity Model WAMO. Proceedings of the 3^{rd} International Conference on Cooperative Information Systems, 1995, 87–98.
12. Jensen C., Snodgrass R.T. Temporal Data Management. *IEEE TKDE* 1999, 11(1): 36–44
13. Snodgrass R.T. (ed.) The TSQL2 Temporal Query Language. Boston, Kluwer Ac., 1995.