Workflow Modelling using a Temporal Object-Oriented Model with Roles

Nina Edelweiss[1]
Mariano Nicolao[1]

Abstract: The representation of all the processes that compose a workflow, including all the constituent activities, their execution sequence and relationships, the agents responsible for their execution, and the resources that are used during execution, is known as Workflow Modelling. Several techniques are being proposed to model workflow. In this paper a workflow modelling technique is proposed, using a temporal object-oriented data model, the TF-ORM model. The TF-ORM model is presented, as are the extensions made to adequate the model to workflow modelling. All the activities that shall be executed during the workflow are represented, including the mechanisms for their activation and termination, both in normal and exception situations. The model allows the representation of structured and unstructured work, the interactions that will occur among executing activities, and their synchronization. The workflow model represented in TF-ORM includes a set of rules to be used during the workflow management. The occurrence of failures and exceptions during workflow execution may cause serious problems, especially in mission-critical applications. The TF-ORM formalism, when used to model workflow, allows the representation of several exceptions and the definition of information to be used for the systems recovery.

1 Introduction

Administrative work accomplished in enterprises is usually composed by different activities, executed by several persons in a defined sequence. As the applications become more complex, the coordination of their execution becomes an important feature to be considered while planning activities. The representation of the planned application activities prior to their implementation is of fundamental importance, especially in complex applications. The definition of the whole process, with the clear identification of all the activities to be executed, of their relationships, and including the agents responsible of their execution, is known as *workflow* [WFMC 99].

Each one of the processes that compose a workflow can be composed of several activities, also with a specific execution order, executed probably by different agents, in different locations. The representation of all the processes and activities is known as *Workflow Mod-*

[1]Instituto de Informática, Programa de Pós-graduação em Computação, Universidade Federal do Rio Grande do Sul, Brasil
[nina,nicolao]@inf.ufrgs.br

elling. This model helps to understand the complete process, and may be useful to identify possible problems that may occur during execution [Tang 98]. Workflow Modelling concepts are being used not only in commercial enterprises – as an example, workflow-modelling techniques can also be used in the educational area, to implement a course on the Web [Oliveira 98].

One of the main goals of workflow modelling is to decrease the number of problems due to activities' coordination. In traditional administrative processes, it is not usually possible to detain absolute control of all the activities that shall be executed. In addition, it is not always possible to identify which data is manipulated by each activity at each moment, and who are the persons manipulating these data. These important aspects shall be clarified by the workflow modelling process. A workflow model shall define not only the sequence of activities to be executed, but also the temporal restrictions to their execution, the dynamic data, and the persons (agents) responsible for each activity.

The understanding and validation of a workflow is fundamental in critical application, as in areas related to health, insurance companies banking, and electronic commerce. They present critical properties that shall not fail: properties related to security or that may lead to life risks. Special attention shall be devoted to these properties, in order to insure (or, at least, increase) their security. A sorrow analysis of the workflow can identify possible exceptions and failures, and the modelling of their treatment will increase the security of the whole system.

A Workflow Management System – also called *Workflow Engine* – controls the execution of all the activities. The importance of managing workflow can be identified by recent researches in this area, leading to a set of academic experiences of Workflow Engines [Alonso 97, Georgakopoulos 95, Jablonski 96, Vossen 96, Weissenfels 98], and a number of commercial tools (Process Buider from Action Technologies, LotusNotes).

As any other computing system, a Workflow Engine may present problems during executions, due to the occurrence of exceptions or to system failures. Experience shows that "Exceptions are not exceptions – they occur all the time". Recent researches on this subject show the importance of this issue when workflow is considered [Eder 96, 98, Ellis 95, Heinl 98, Reichert 97, Saastamoinen 95, Van Stiphout 98, Tang 98]. If some predictable exceptions are represented in the workflow model, it is possible to program the Workflow Engine to solve them. Further on, when the solution for an identified problem is not trivial, the Workflow Engine needs special information to provide a failure recovery. The possibility of representing recovery information in the workflow model is also a desirable feature when selecting a modelling tool.

Several workflow-modelling techniques are proposed in recent researches, using different modelling paradigms. Some examples are the model proposed by the *Workflow Man-*

agement Coalition [WFMC 94], the model of Casati, Ceri, Pernici & Pozzi [Casati 95], the trigger model of Joosten [Joosten 94, 94a], the WAMO activity model [Eder 95], object-oriented models [Missikoff 98] and Petri Nets [Aalst 95, Eddis 93].

A methodology is suggested in [Baresi 99], the WIDE methodology, aimed to support a workflow project from the initial analysis up to the implementation in specific workflow systems.

An alternative technique to model workflow is proposed in this paper. The technique is based on the use of the temporal object-oriented model TF-ORM (*Temporal Functionality in Objects with Roles*) [Edelweiss 93, 93a, 94]. This model was extended with some new syntactic constructions, with the aim of better representing the workflow characteristics [Nicolao 98]. Besides being temporal and using the object-orientation paradigm, this model uses the concept of *roles* to represent different behaviors of objects, allowing a richer representation of the workflow [Edelweiss 97].

Using TF-ORM, the behavior of the workflow processes is represented by classes, encapsulating the representation of the activities of each class. The classes are instantiated for each process execution. The definition of all the classes' interfaces with other tasks, and with agents and resources involved in the workflow completes the model. A sort of finite state machine, presenting temporal logic expressions to constrain the state transitions, represents the evolution of a process. The main difference of this method comparing to traditional object-oriented models is the use of *roles* to represent activities composing a process, and extending the conditional state transitions also to these roles.

The use of the TF-ORM formalism allows also the representation of possible exceptions identified in the modelling level, and of the actions the Workflow Engine shall perform when these exceptions occur. And it also provides ways of representing information that can be used in failure recovery mechanisms [Edelweiss 98].

The paper is organized as follows. In Section 2 some concepts concerning workflow modelling are presented, including exception and failure procedures. The TF-ORM data model used in this proposal is presented in Section 3.. To better represent workflow models, the TF-ORM model was extended, ad this is presented in Section 4. In Section 5 the constructing of the workflow modelling technique, using TF-ORM is detailed. Section 6 complements the explanation of the modelling technique, explaining how the different synchronisms between workflow activities are represented in TF-ORM. And the exceptions representations and the failure handling are detailed in section 7.

2 Workflow Modelling

A workflow, also called a business process, is usually concerned with the coordination of activities in a business environment. Workflow Modelling is a very important subject in enterprise analysis processes. The expressive number of researches published recently in this area attest this fact. Three different dimensions are identified in a workflow [Leymann 98]:

1. *What* shall be represented – the set of activities that compose the workflow, and the sequence of their execution;
2. *Who* shall execute each one of the activities – the agents of the workflow; and
3. *With* what the activities shall be executed – the resources involved in the execution, including automatic tools and support.

Information related to these three dimensions shall be defined in the workflow model. However, most of the available modelling tools allow only the representation of information related to the first one. Here lies the main difference between *Enterprise Modelling* and *Workflow Modelling*.

The construction of a workflow model has two important uses. First, during the construction of the model the whole process is analyzed, and the activities to be executed and sequence of their execution is identified. This is done in an abstract way, with the purpose of analyzing the workflow. The constructed workflow model may lead to a process re-engineering, with the aim of solving detected errors and to achieve a better work distribution [Georgakoupoulos 95]. And after the workflow model is complete, the resulting model can be used to manage the workflow during the execution.

A workflow model shall include:

a. The description of all the processes that compose the workflow, and the activities performed in each one of these processes;
b. The definition of agents responsible for the execution of each one of the identified activities;
c. The definition of temporal restrictions to the execution of activities; and
d. The identification of communication between activities and processes (for information change and control).

In order to reach this goal, the following strategy can be used:

- Construct a mathematical model of the workflow;
- Formalise the processes and their relationships, using a temporal logic, in order to generate all the possible logical computing trees of the workflow (in such a tree, the successors of a node are the possible states that can be assumed starting from the state represented by the corresponding root); and
- Use some method to verify this model, usually done by a simulation of the workflow execution.

Workflow Modelling can be done using different formalisms. Several modelling techniques were proposed recently. And some commercial workflow tools are available. But all of them present some limitations – it is important to make a sorrowful analysis of the technique to be used in an application, to have all the necessary functionality needed to the modelling. The use of a temporal model is advised in order to allow the representation of execution synchronism among the activities. This paper proposes the use of the temporal object-oriented data model TF-ORM to used as a workflow-modelling tool. This model combines the advantage of being a formal method with the possibility of representing human decisions in the formal framework, and all the possible synchronism needed in workflow applications.

2.1 Temporal modelling

Temporal aspects are fundamental in workflow. Workflow models need to support expressions related to processes, temporal restrictions, dynamic changes and treatment of exceptions [Ellis 95]. The synchronism of the different tasks and activities involved in the workflow is usually controlled by temporal logic rules. The choice of a modelling method to represent workflow shall support the representation of temporal rules to allow a complete representation.

However, this not always happens in the existent modelling techniques. Temporal modelling allows the representation of many of these aspects. Using temporal modelling the dynamics characteristics of the applications and the temporal interaction among different processes can be represented. The possibility of storing, manipulating and recovering temporal data should also be considered when choosing a workflow modelling method.

A workflow also needs time restriction controls. These controls produce warnings (e.g.: First Reminder After $\underline{x}$ days, Repeat Reminders Every $\underline{y}$ days, Deadline Occur After $\underline{z}$ days) to automatically alert the participants of pending activities, as well as to fire other activities related with time. In TF-ORM these controls may be easily represented through state transition rules, more specifically in the transition condition, explained in section 3.2.

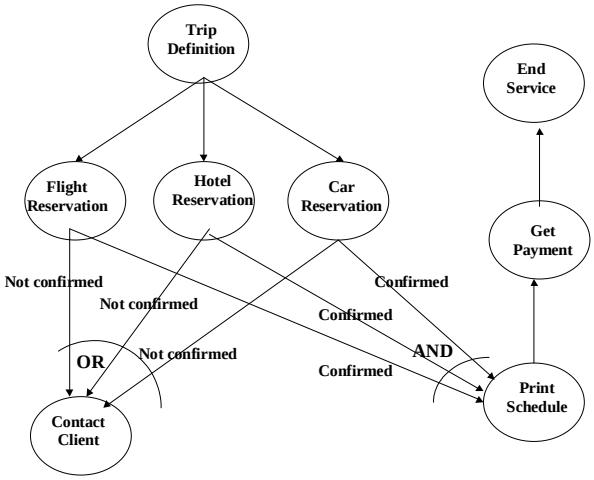

Figure 1. The Traveluck Example

2.2 Exceptions and Modelling Errors

The modelling of a workflow can be considered as the first phase of a re-engineering process of the enterprise. The analysis of possible exceptions that can occur during execution, and of eventual system failure consequences is important for the complete understanding of the application. *Exceptions* (also called semantic failures) occur when activities cannot by executed according to the workflow model, or do not present expected results. *Failures* consist of problems due to equipment (errors in programs, failure of some equipment, failure of communication).

An important distinction shall be made between exceptions occurring during the workflow execution and errors due to deficiencies of the workflow representation. To exemplify this difference, consider the following example, based on the "Traveluck Example" presented in [Leymann 98]. It concerns a travel agency, and the activities to be executed when a client of this agency hires a trip, composed of the airplane ticket, hotel reservations and car rental. The main activities correspondent to this application are represented in Figure 1, using an activity diagram notation often used to represent workflow graphically - circles represent activities, and the arrows define the sequence of their execution. The workflow begins with the trip definition activity, when the client, together with the agency employee, will choose the itinerary. When the itinerary is fixed three different and parallel activities will be fired: flight, hotel and car reservation.

Two alternative and independent results can output these three activities: the reserva-

tion can be confirmed or not. If one of the three reservations is not confirmed, the client is contacted (the OR operator represents a conditional join). And only if all the three reservations are confirmed (AND operator) the next activity is fired, and the trip schedule is printed, followed by the payment and ending the workflow with the ticket and vouchers delivery.

The workflow of this example presents a *modelling error*: when one of the reservation activities is not confirmed, another may already be confirmed, and this last reservation is not undone. Once the client decides what shall be done to overcome the detected reservation problem, the workflow would begin once again, and the reservation that was already confirmed would be done again. The workflow should verify if any other reservation was made in such a case.

One possible *exception* identified in this example would occur in case the client does not pay the trip – the workflow would wait the payment indefinitely. The workflow model can provide some treatment to avoid this exception.

3 TF-ORM

TF-ORM (*Temporal Functionality in Objects with Roles Model*) [Edelweiss 93, 93a, 94] is a temporal object-oriented data model. It differs from other temporal object-oriented data models by the use of the role concept to represent the different behaviors an object can present.

3.1 The Role Concept

Object-oriented data models are often used to model real applications. However, it is difficult to represent the temporal evolution of the application when traditional object-oriented data models are used. Some mechanism should be provided to support the real world temporal evolution. According to the object-oriented paradigm, an object is an instance of a class, and presents the properties and methods of this class during all its existence. Even if, due to temporal evolution, some of the characteristics of this object change, the class definition keeps unchanged. This evolution may lead to situations in which the behavioral characteristics become similar or identical to those of another class, and should produce the migration of this object to the other class. Some models allow some kind of limited migration, but these are usually restricted between a class and its subclasses.

This problem is overcome when using roles associated to the object-oriented paradigm. Roles, encapsulated in classes, represent different behaviours an object of a class can play, simultaneously and independently. Roles can be instantiated dynamically, allowing this way the representation of the dynamic evolution of an object.

The role concept associated to the object-oriented paradigm was introduced with the ORM model (*Objects with Role Model*) [Pernici 90] and has been adopted by some modelling techniques since then [Belkhatir 94, DeAntonellis 91].

The ORM model introduces the role concept in an object-oriented model to represent the dynamic behavioural evolution of an object. A class presents different roles, each one representing a specific behaviour of an object of this class. An object is still an instance of only one class, but it can play different roles during its lifetime. As an example, consider the class *person*, in an academic environment. Three different roles are identified for this person (Figure 2): *professor*, *student* and *administrative_employee*. For each one of these roles a person presents specific properties and methods – the lectures he is used to present as a professor, the courses he already has completed as a student, or the salary as an employee.

The role concept provides the separation of the dynamic aspects of an object from the static ones. The roles can be dynamically instantiated - considering always the same class instance (the same object), instances of roles can be created, destroyed, or even temporarily suspended and resumed. This allows the representation of the temporal evolution of an object's behavior. In the above example, a person (an object) can be accepted at the university in, for instance, March 1990. This is represented with the creation of an instance of the role *student* for this person. Supposing he graduates in December 1995, this instance is destroyed. But in March 1991 he is accepted as a teacher at the same university, represented by the creation of an instance of the role *teacher*. The evolution of the behavior of this person is dynamically represented by the roles he is playing at different times. Note that the object - the person - is always the same, even if his behavior is now absolutely different as at the beginning.

An object may present, at the same time, two or more instances of roles. In the previous example, the same person can be, at the same time, a student and a professor: consider that, in March 1996 this person, that is already a professor, is accepted as a PhD student at another university. He keeps on playing the role of professor, but an additional instance of the role *student* is created. Both instances of roles (*professor* and *student*) are played at the same time from that moment on, each one independent of the other one.

And, finally, an object can have more than one instance of the same role at the same time, all absolutely independent. This could be represented in the previous example if the person becomes teacher of another university, keeping also the previous post - an additional instance of *teacher* would be created, and from that moment on each one of the two instances of this role would evolve dynamically on their own way.

In traditional object-oriented models an object is represented as a unique instance of a class. It is not possible to represent that a person has two or more different jobs, each one having different characteristics. In this case, the object should be an instance of the employee

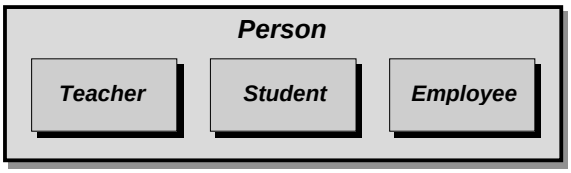

Figure 2. Graphical representation of a class with three roles

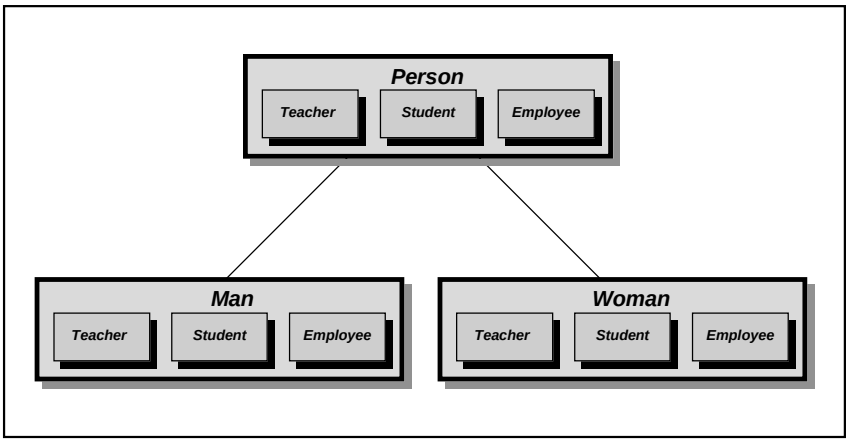

Figure 3. Subclasses with roles

subclass and two instances of this subclass would represent two objects. The use of the role concept associated to object-oriented model aims at overcoming also this limitation, allowing that an object presents multiple instances of the same role.

3.2 Roles and Subclasses

The use of roles is not equivalent to the subclass concept - subclasses can also be defined when roles are used. A subclass inherits the roles of the superclass, can eventually redefined a role definition, or even present new roles. Subclasses specialise properties and/or methods of the superclass, not only in the class level, but also in the role level. Figure 3 presents an example of two subclasses defined for the class of Figure 2, each one with specific properties and methods: one represents a *Man*, and the other a *Woman*. In each one of the subclasses the same roles of the superclass are defined.

An example that shows clearly the distinction between roles and subclasses is pre-

sented in [Wieringa 91]:

"Assume that *passenger* is a subclass of *person*, and consider a person who migrates to the *passenger* subclass of person, say by entering a bus. This bus can carry 4000 passengers in one week, but counted differently, it may carry 1000 persons in the same week. So counting persons differs from counting passengers".

If *passenger* is represented as a subclass of the class *person*, then counting passengers would be the same as counting persons. The OId (object identifier) of the object in the subclass is the same as the one of the superclass. Each time this person enters a bus, he becomes a passenger, but each time a different passenger (it can be another bus, or even the same bus but at another time, he can stop at a different point, etc.). To allow this representation, each time this person enters a bus a new object should be created, as persons would be identical to passengers. The same person would thus be represented as different objects.

To keep the identity of a person as only one object, the same OId should always be used each time this person becomes a passenger. To be a passenger represents in reality a *state* of this person. This state can be represented as a *role* played by this person - the *passenger* role. Now we have only one object representing a person (with a corresponding OId). Different instances of the role *passenger* can be created and destroyed during the lifetime of this object. To count passengers is now the task of counting how many persons are involved, independently of how many instances of the role passenger are created.

3.3 The TF-ORM Model

TF-ORM [Edelweiss 93, 93a] is an extension of the ORM model [Pernici 90], introducing the representation of temporal features. Time is modelled as varying in a discrete form. A unique name and a set of roles define a ***class***:

$class_i = (cn_i, R_0, R_1, ..., R_n)$

Each ***role*** is defined by a name (rn_i), a set of properties (P_i), a set of abstract states the role can assume while playing this role (S_i), a set of messages the role can receive or send (M_i), and a set of rules - state transition rules and integrity rules (Ru_i):

$R_i = (rn_i, P_i, S_i, M_i, Ru_i)$

3.3.1 Properties, states, decisions, and messages ***Properties*** may be static (having the same value all over the instances lifetime) or dynamic (when they may assume different values with time). Dynamic properties have two different time points associated with each value: the *transaction time*, corresponding to the moment when the information is introduced in the database, and the *valid time*, the time when that information starts to be valid in the real world. Thus, this data model implements a bitemporal Database [Jensen 98]. Domains are

assigned to the property values. TF-ORM presents a set of pre-defined classes, called *data types*, which can be used as domains of properties. In addition to the usual numeric data types (*real*, *integer*), several temporal data types are supported (like *date*, *hour*, *instant*, *semester*, *interval*). Complex domains can also be defined, as an object, a set of objects, or a list of objects.

The ***states*** an object can play are simply identified by a name. This name must be unique within a role definition. These state names are referenced in the state transition rules.

The objects' methods are represented as ***messages***. Incoming and outgoing messages are defined, together with the information of which class is sending or receiving the message. Values are passed by the way of message parameters, to be used in the methods. Human decisions are represented in agent classes as incoming messages called ***decisions***. These can also present parameters.

3.3.2 State transition and constraint rules ***State transition rules*** define the dynamic evolution of an object. The arrival of a message sent by another class (or by another role of the same class) does not mean that the corresponding method will always be executed - state transition rules control these messages. Such a rule defines a combination of an object state s_1 and incoming message(s) mi_1 to change to state s_2. One or more messages can be send when a transition rule is executed (mo_1 through mo_n). A transition condition, represented by a first order temporal logic formula, can be associated to a rule, acting like a restriction to the state transition execution - the transition will only occur if this condition is true. The general form of a state transition rule r_i is the following:

$$r_i : state(s_1), msg(mi_1) \Rightarrow msg(mo_1), msg(mo_2), ..., msg(mo_n), state(s_2); [<transitioncondition>]$$

A rule may also be defined based on the arrival of a set of messages - the rule is only executed when all the messages have arrived, in any order or arriving time. This is represented as follows:

$$r_i : state(s_1), msg(mi_1), msg(mi_2), ..., msg(mi_n) \Rightarrow msg(mo_1), state(s_2); [<transitioncondition>]$$

All the components of a state transition rule are optional. When the initial state is not defined, the transition takes place every time the incoming message(s) arrive, independently of this state. When the incoming message is not defined, the transition is executed every time the object comes to the initial state. If the outgoing message is not defined, the transition takes place without sending anything. And when the final state is not defined, only the outgoing messages are sent, without changing the objects state. In any one of these situations, the transition condition shall always be obeyed.

Constraint rules can also be defined, and are represented by two conditions: if the first condition holds, the second shall also hold. Temporal conditions can be used in both

forms of rules. The set of rules completes the object's behavior definition.

3.3.3 Process, agent and resource classes Three different kinds of ***classes*** can be defined: (i) *resource classes*, modelling information and resources; (ii) *process classes*, representing the processes to be executed with this information and the resources; and (iii) *agent classes*, representing the agents that carry out the processes. The three class types are modelled in a similar way. The only difference concerns agent classes: in addition to the above-mentioned messages, agent classes also include human decisions, representing non-structured work in a formal definition environment.

The role concept is different depending on the class type. In agent classes, the roles represent the different behaviors of an agent may present (e.g.: class *person* – roles *teacher*, *employee*, and *student*). Resource classes roles represent different ways of visualizing these resources, depending on the processes that act on them (e.g.: class *book* – roles *confection*, *distribution*, *sell*). And in process classes, the roles represent the activities that compose the process (for ex: class *accountancy* – role *salary control*, *budget*).

Temporal information are associated to all the instances (class and role instances) - the instance's creation time and destruction time, and the time instants in which the instance's activity was suspended and resumed. These temporal information are stored in special pre-defined properties and can be used by the query language to retrieve information. Pre-defined properties are inherited from a superclass *Object*, from which all the TF-ORM defined classes are sub-classes.

Each class presents a special role, the *base role*, where the global properties inherited by all other roles and the initial characteristics of the other roles are described. The TF-ORM model supports specialization and aggregation mechanisms, with the possibility of inheriting roles, or redefining them.

3.3.4 Pre-defined messages TF-ORM presents a set of pre-defined messages, used to manipulate instances of classes and of roles. For instance, the creation of an instance of a class is made by the following pre-defined message:

$$add\_object(OId, < Class\_Name >)$$

$$add\_role(OId, RId, < Role\_Name >)$$

The *add_object* message creates an instance of the specified class, and the object identifier (*OId*) of this object is returned in the parameter *OId*. Similarly, the message *add_role* creates an instance of the specified role of the object identified by *OId*, and returnes the identifier of this role instance in the parameter *RId*.

Other pre-defined messages allow suspending an instance's activity, to resume his

execution, and to end the instance's life.

3.3.5 TF-ORM examples An example of the TF-ORM definition of the *Person* agent class with the three mentioned roles is partially bellow, using the TF-ORM definition language:

```
agent class (
  PERSON,
  < base_role,
static properties = {(person_id, integer)},
  dynamic properties = { (name, string), (address, string)},
rules = { ... } >,
  < employee,
     dynamic properties = {(department, string), (salary, real),
          (hired, date), (holidays, interval(closed, date), ... },
   states = {hired, in_holidays, fired},
   messages = {
              new_salary(Oid, Value) from Control.Salaries,
              ask_vacations(oid, Period) to Control.Holidays,  ... },
   decisions = { get_vacations( Period), ...  },
   rules = {
init: add_role ⇒ state( hired ),
holidays: state(hired), decision(get_vacancies( Period) ⇒
msg(ask_vacancies(oid, Period), state (hired),
salary: state(hired), msg(new_salary(oid, Value) ⇒ sate(hired);
                          (Value > salary),
... } >
< teacher,
   dynamic properties = { (gratification, real), (start, date), ... },
   ... >
< student,
        static properties = { (student_number, integer) },
   dynamic properties = { (courses, string), (start, date), ... },
   ... > )
```

To better explain the state transition condition, another example is presented here. The control of a deadline date for the arrival of a paper to be submitted to a conference may be represented through the following rule:

$$deadline : state(active) \Rightarrow msg(deadline(paper)), state(active);$$
$$\exists Rid(has\_role\_instance(Oid, paper, Rid))$$
$$and\ (value(Rid, upper\_bound(deadline)) \geq now)$$

The rule *deadline* describes the following situation: if the object is in the state *active* and if the transition condition is satisfied, then this activity sends the message *msg*(*deadline*(*paper*)) staying in the state active. The message *msg*(*deadline*(*paper*)) fires an activity that will treat

the end of the receipt period. The transition condition verifies if the deadline for this has been reached: $\exists$ *Rid* (*has_role_instance*(*Oid, paper, Rid*) verifies if there exists an instance of the object *paper*, and (*value*(*Rid, upper_bound*(*deadline*)) $\geq$ *now*) if for this paper (*RId* identifies the paper), the deadline is reached.

4 Extensions of TF-ORM for workflow modelling

To support the formalism and the necessary flexibility of a workflow model, some extensions were accomplished in TF-ORM, commented below. In this section some of these extensions are presented – extensions that shall be used during the modelling phase. An example of these extensions, using the application presented in Section 2.2, is presented in the Appendix of this paper. Further extensions, done with the aim of solving possible exceptions and failures, are presented in Section 7, together with the explanation of these problems.

4.1 Definition of agents for processes and activities

In workflow modelling, an important feature is the definition of agents for processes and activities. To achieve this the TF-ORM model was extended with the following clauses:

$$responsibleagent =< agent\_class > . < role >$$

$$executingagent =< agent\_class > . < role >$$

$$delegationagents =< agent\_class > . < role >$$

These clauses may be used only in a process or in a role definition of a process class. They allow the definition of three different types of agents, linked to the corresponding process or role.

The *responsible agent* definition identifies the role that that will be played by the agent that will be responsible for this process/role. Note that, during the workflow modelling, only a role is defined for agents. Just the skills that this agent shall present are defined. When an instant of a process/role is created, an instance of that agent role is selected and associated to it. If any problem occurs during the execution of the process/role, this specific agent shall solve it. This clause is obligatory in process classes definition – each process shall present a responsible agent. Even when the process to be executed is automatic, a responsible agent shall always be designated to it, to solve eventual failures. When the responsible agent is not defined in a role definition, the responsible for the class that contains this role will also be responsible for this role.

The *executing agent* achieves the execution of the process/role. When the executing agent is not defined the responsible agent for this process/class will also be the executing one.

In order to solve possible problems due to a sudden impossibility of an agent responsible for a process/role, a set of other agents may be defined, to whom the responsibility can be delegate. This is done by the *delegation agents* clause.

The choice of agents to be responsible/executing/delegated is done when the instance of the process/role is created. To cope with these agents definition, the pre-defined messages that create instances of processes/roles were also modified, with the addition of new parameters:

$$create\_object(OId, Class, [Resp], [Exec], [Deleg], [Auto])$$

$$add\_role(Oid, Role, RId, [Resp], [Exec], [Deleg], [Auto])$$

where:

- *Resp* - identifies the responsible person for the activity;
- *Exec* - identifies the executor of the activity;
- *Deleg* - determines the list of delegated agents; and
- *Auto* - determines that the activity is performed automatically.

The argument *Resp* is obligatory. If *Exec* isn't represented, it is understood that the responsible agent is also the executor. If *Deleg* is not represented, then it is understood that this process cannot be delegated to another person.

4.2 Decisions in process classes

Another extension of TF-ORM was done to allow the representation of the decisions not only in agent classes, but also in the process classes. This allows identifying and monitoring the agents' interactions during the processes execution, improving that way the modelling and maintaining the necessary formalism to the representation of workflow associated with the processes. The decisions are registered in the process class through the *decision* clause:

$$decision(< decisionname > (< messageparameters >)$$

Only the agent classes can take decision – they represent the unstructured work of these environments. But representing them directly in the process classes, they can be used in the state transition rules, as incoming messages sent by agents. As an example, the following decision and state transition rule may be defined in a process class (complete example in the Appendix):

$$decisions = \{add\_city(City : string) from PERSON.Client, \ldots\},$$

$$rules = \{cities : state(defining), decision(add\_city(City)) \Rightarrow state(defining), \ldots$$

Table 1. TF-ORM classes and roles for Workflow Modelling

TF-ORM class	Corresponding meaning in the workflow	Meaning of the roles in the workflow
Process	Process to be executed	Activities that compose the process
Agent	Agents of the workflow	Roles an agent can play
Resource	Resources handled by the workflow	Different ways of visualising the resource

5 Workflow Modelling using TF-ORM

The TF-ORM model can be used to model workflow in a quiet natural way. The workflow concept is strictly associated to the representation of processes. In TF-ORM these processes are represented as process classes. Each activity of a process is represented as a role of the corresponding process class. The agents involved in the processes (as responsible or executors) are represented by agent classes, and the roles represent the different roles each agent can play during the process and activities execution. And the resources involved in the execution – data and documents – are represented as resource classes. Resource roles represent different ways of visualizing the resource. Table 1 summarises the correspondence of TF-ORM classes and roles, and workflow concepts.

The integration between agents and resources is done by the activities' representation. So, a workflow is actually represented by the process classes, including the actions executed on resources, and the responsibility and cooperation of agents.

5.1 The meaning of 'roles'

According to the workflow concept, the term "role" means a group of participants exhibiting a specific set of attributes, qualifications and/or skills [WFMC 96]. The workflow role concept is similar to the role concept used in the TF-ORM model, concerning agent classes. TF-ORM agent class roles represents the different behaviours an agent (or a group of agents) may present. For each role a group of properties (static or dynamic) is defined, representing the participant's characteristics in the business process.

Some models show limitations in the agent's formal representation. The model described in [Joosten 94, 94a] characterises the agent role by a label. The models described in [Casati 95] and [WFMC 96a] present a certain degree of formalism in the definition of the agents' roles but the properties related to the roles are pre-defined, restricting the flexibility

in the representation. Using TF-ORM it is possible to define formally any group of static or dynamic properties needed specifically for roles' representation.

The activities of a workflow are executed according the organisational structure of the enterprise, where the agents' functional roles and possible relationships are defined [WFMC 96]. The activities composing a process are usually scheduled to specific agents, who are responsible for their execution. An agent may play different roles, simultaneously or not.

5.2 How to construct the Workflow Model

The modelling of a workflow using TF-ORM begins with the identification of the processes that compose this workflow, representing each one as a process class. Each process is then analysed separately.

For each process, the following steps are the performed:

1. Identify the component activities, each one represented by a role in the corresponding process class;
2. Define the executing / responsible / delegate agents for the process, defined through predefined properties in the base role;
3. Identify the properties that are common to all the activities, and define them as static or dynamic properties of the base role;
4. Define the rules that control the activities instantiation (base role state transition rules);
5. Complete the definition of each activity (role) defining first the executing / responsible / delegate agents for the activity, then the static and dynamic properties of the activity, the messages to be sent and received, the states the activity can present, and finally, the state transition rules among these states.

Table 2 summarises which workflow concepts are represented in the TF-ORM process classes.

Once the process classes are defined, the modelling is completed identifying all the agents involved in the processes (represented as agent classes) and all the manipulated resources (represented as resource classes). The equivalence between workflow concepts and the correspondent TF-ORM representation are showed in Table 3 (for the agent classes) and Table 4 (for resource classes).

The processes send messages to agents and to resources, with the aim of recording new property values or to change their states. As the main focus in workflow is on processes,

Table 2. Workflow process class

Workflow	TF-ORM
Process	Process class
Responsible, executing and delegate agents for the process	Base role pre-defined properties
Properties common to all the activities of a process	Base role properties
Activity control	Base role rules
Activity	Role
Responsible, executing and delegate agents for the activity	Role pre-defined Properties
Specific properties of an activity	Role properties
Operations executed by an activity	Messages and decisions in the role
Initial and final states of an operation	States of the role
Synchronism among activities	State transition rules
General integrity constraints	Integrity rules

the representation of the process classes evolution is controlled only through the income of messages sent by other processes. Therefore, the message flow between the three TF-ORM class types is supposed to be the one represented in Figure 4.

6 Synchronism between activities

The activities that compose the processes of a workflow can present several different synchronism features, represented in the TF-ORM model through the state transition rules. The messages describe the interactions among the activities, determining the flow control and allowing the synchronisation of the activities that compose a process. The possibility of formal representation of these interactions is a fundamental issue in a workflow model. The activity execution order (synchronism) determines the evolution of the work in a workflow.

Activities are fired by messages sent by other activities. By “fired” we mean not only the beginning of an activity’ execution, but also the continuation of a paused activity, waiting for an event to continue her execution.

Activities can execute in sequence or in parallel, independently or in a synchronised way. The following synchronism conditions can occur: *sequential*, *convergent* (Join), *divergent* (Fork) and *conditional*. The analysis of several forms of synchronism and the corresponding representation in TF-ORM is presented here.

Table 3. Workflow Agent Class

Workflow	TF-ORM
Person	Agent class
Person executing an activity	Role of the agent class
Properties common to all the agent behaviours	Base role properties
Specific properties of a behaviour	Property of the correspondent role
Decisions that this agent can take in any behaviour	Base role decisions
Decisions that this agent can take in a specific behaviour	Decision of the correspondent role
Actions this agent can perform	Messages of the correspondent role
Initial and final state of an operation	States of a role
Agent evolution	State transition rules

Table 4. Workflow Resource Class

Workflow	TF-ORM
Resource manipulated by a process	Resource class
Different aspects of the resource	Role of the resource class
Properties common to all the resource behaviours	Base role properties
Properties of a specific behaviour	Properties of the correspondent role
Actions that are executed on this resource	Messages of a role
Initial and final states of an operation	States of a role
Resource evolution	State transition rules

6.1 Sequential

Activities can be scheduled in sequential form, obeying a fixed execution order. Two activities are *sequential* when the execution of the second activity only begins when the previous one has finished her own. Figure 5 shows a graphical representation of the transition of activity a_1 from state s_1 to a final state s_f in consequence of the receiving of message m_1. This transition causes the start of another activity a_2, sending a message that creates a new instance of that activity (represented by the message *add_role*):

$$st(s_1, msg(m_1) \Rightarrow msg(add\_role(< pr_i >, < a_2 >)), st(s_f);$$

To simplify the examples presented here, no transition condition is represented.

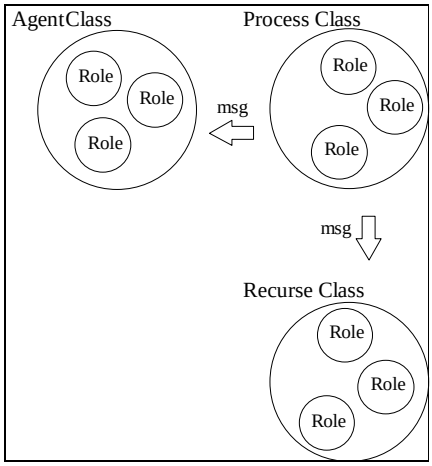

Figure 4. Messages for Workflow Modelling using TF-ORM

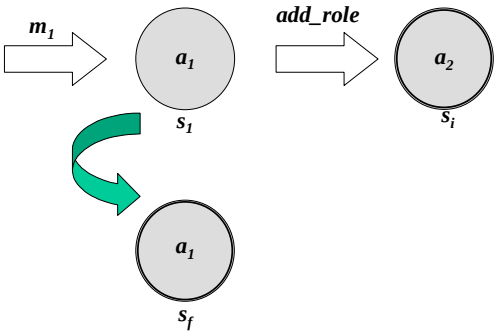

Figure 5. Sequential activities

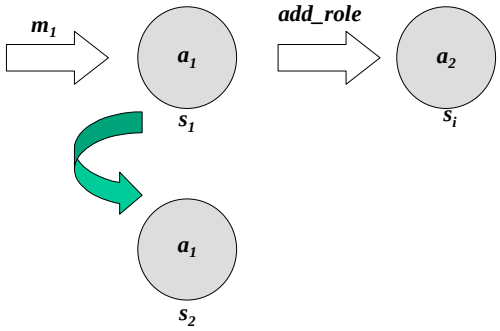

Figure 6. Parallel activities

6.2 Parallel activities

In the same example presented before, if s_f is not the final state of a_1 and this activity continues executing, the first activity only fires the second one, and from there on both evolve *in parallel* (Figure 6):

$$st(s_1), msg(m_1) \Rightarrow msg(add\_role(< pr_i >, < a_2 >)), st(s_2);$$

6.3 Synchronised, parallel activities

A similar situation is when an activity is temporarily suspended, waiting a specific message to resume execution, characterising a *synchronised execution*. In the example of Figure 7 the second activity would be in a waiting state, and the output message of a_1 would not be *add_role* but the message that represents the event for which the second activity is waiting. The first activity may continue her execution, or evolve to another waiting state, and stay so until another message activate it again. The rule representing this situation has the following form:

$$st(s_1), msg(m_1) \Rightarrow msg(m_2), st(s_2);$$

The following rule shall be defined in the set of transition rules of the second activity:

$$st(s_x), msg(m_2) \Rightarrow st(s_y);$$

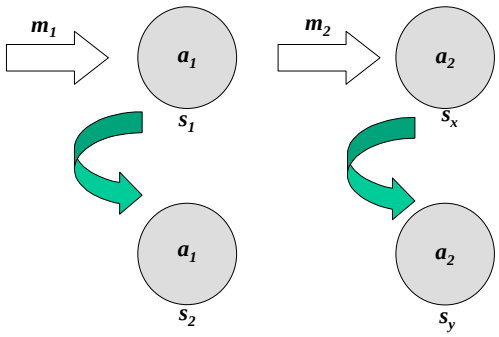

Figure 7. Synchronised, parallel activities

6.4 Totally convergent synchronism (total join)

The execution of an activity can be conditioned to the receiving of a set of incoming messages, sent by different activities. This situation is usually called *convergence*, represented by a *join* of incoming messages. All the messages must arrive to execute the transition (*total convergence*) (Figure 8). The order in which the messages arrive is not relevant. In TF-ORM this is represented by a unique state transition rule with a set of incoming messages (example in the Appendix):

$$st(s_1), \{msg(m_1), msg(m_2), ..., msg(m_n)\} \Rightarrow msg(m_o), st(s_2);$$

6.5 Partially convergent activities (partial join)

A situation similar to the previous one may occur when only a subset of the incoming messages is required (*partial convergence*). If the set contains *n* messages, and only *k* messages are required, the relation ($1 \leq k < n$) must hold. This is represented in TF-ORM defining the number of messages that shall arrive in front of the set of incoming messages:

$$st(s_1), k\{msg(m_1), msg(m_2), ..., msg(m_n)\} \Rightarrow msg(m_o), st(s_2);$$

Also in this case the messages may arrive in any order. Figure 9 represents this kind of synchronism supposing that only two messages are required.

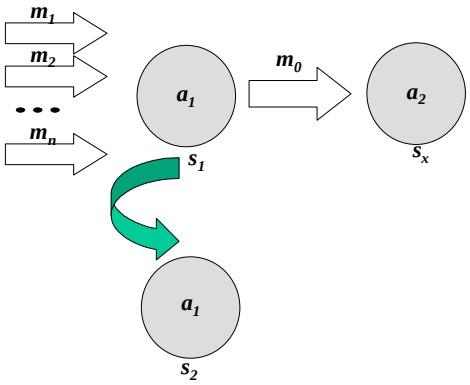

Figure 8. Totally convergent synchronism

6.6 Divergent activities (fork)

The opposite situation occurs when an activity fires the execution of several other activities, which is known as a *fork*. This is represented by a state transition rule with several outgoing messages (example in the Appendix) (Figure 10), represented by the following rule:

$$st(s_1) \Rightarrow msg(m_1), msg(m_2), \ldots, msg(m_n), st(s_2);$$

6.7 Conditional activities

All the previous situations can be conditioned by the transition conditions, in which present and past values of properties and states can be verified to define if a transition shall be executed or not.

$$st(s_1), msg(m_1) \Rightarrow msg(m_2), st(s_2); (< transitioncondition >)$$

The possibility of representing this temporal condition is one of the most important features of the proposed modelling technique.

7 Representing exceptions and failure recovery information in TF-ORM

The use of TF-ORM enables the representation, in the workflow model, of significant information that can be used in case exceptions occur during the execution, including information for failure recovery.

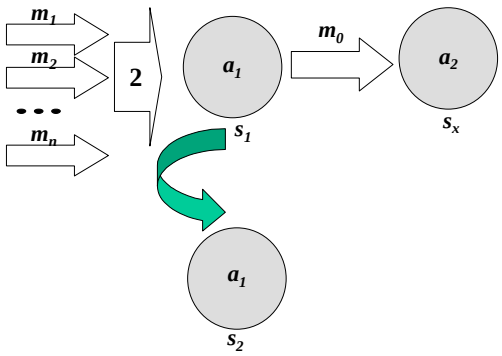

Figure 9. Partially convergent synchronism

Exceptions can be represented in the workflow model when using the TF-ORM formalism, with the corresponding actions to be executed. In the following sections some possible exceptions are identified and their representation in TF-ORM is explained. At the end, some explanation is given about failure recovery.

7.1 Impeachment of the agent in charge of an activity

Each workflow activity shall have a responsible agent, to control the execution and solve possible problems. When the TF-ORM model is used in workflow modelling, the definition of a responsible agent for each activity is required. In the model, only the role that the agent shall play is defined. The specific person is defined the moment an instance of that activity is created.

An exception will occur when the agent presents an impeachment, and the responsibility of the corresponding activity shall be transferred to another one. To prevent this exception, the TF-ORM model allows the representation of a set of agents to whom the activity can be delegated. These agents shall also be defined the moment the activity is instantiated.

Even if there are some agents to whom the responsibility for the activity can be delegated, the exception can still occur if the whole list is impeached. To prevent this last case, the TF-ORM model requires the definition of an agent responsible for the whole process, and this one shall define who will be the new agent responsible for that activity.

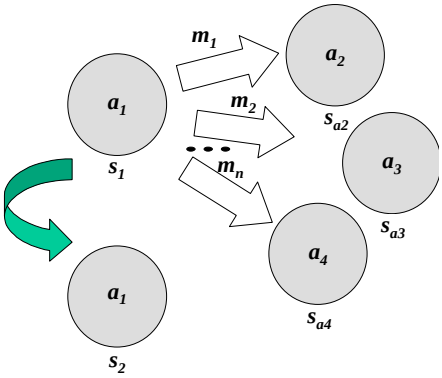

Figure 10. Divergent activities

7.2 Changing the activity flow during execution

When the order of the activities is changed during the workflow execution, serious problems can occur. To avoid these problems, the TF-ORM requires the definition of an agent responsible for each process. Only this agent shall have the power of changing the order of executing activities.

To implement this interference, each activity presents two pre-defined incoming messages, to be sent by the agent responsible for the process in which the activity is defined. These messages shall be received any time, independent of the actual state of the activity. The process' responsible agent shall send these messages to all the activities that will be involved in the changing he is planning. The first message will change the actual state of the activity to the state defined by the agent, with passed as a parameter. The second message concerns property values - if there is the need to adapt some properties values, specific messages shall also be sent, with the name and the new value of these properties.

As an example, consider the same example presented in Section 2.2, concerning a travel agency. Suppose the responsible agent wants to suspend a running hotel reservation activity and, undoing reservations that were eventually already made. This would be done sending to this activity the following messages:

$msg(agent\_interfer(hotel\_reserve, suspended))$

$msg(values\_interfer(hotel\_name, null)),$

$msg(values\_interfer(hotel\_reservation, nok))$

The effect of these messages is the same as if a rule were defined for each one of the

states of the hotel reservation role, as the following one (for the "reserving" state):

$$ri : state(reserving),$$
$$msg(agent\_interfer(hotel\_reserve, suspended)),$$
$$msg(values\_interfer(hotel\_name, null)),$$
$$msg(values\_interfer(hotel\_reservation, nok)) \Rightarrow state(suspended)$$

It is important to remember that the executing agent is responsible for the change in the activities' execution order, and it is his task to adapt all the involved processes and activities to the new executing order. The modification may reflect on property values that were changed in former executing activities.

7.3 Activity waiting for not available resources

A possible exception may be caused when an activity is suspended waiting for a resource, and this resource is not available. The availability of a resource is represented in TF-ORM by a message sent by the corresponding resource class to the activity.

A way of avoiding this problem is to model the interaction between the process and resource class requiring an answer of the required resource – positive or negative. The transition rules of the activity shall consider both the answers, and provide an alternative solution in case the resource is not available. If there is no alternative solution, the activity shall send a pre-defined message to the agent responsible for the activity (represented by a message sent to the same role), asking for an intervention. This message has the following form:

$$msg(resource\_interfer)\ to\ itself$$

7.4 Cycles

The occurrence of cycles during a workflow executing is one of the most important problems that can happen. A cycle is characterized when an activity a_1 fires another activity a_2, and the activity a_2, directly or not, activates again the activity a_1. This situation can only be avoided by a serious analysis of the workflow, considering all the possible evolutions.

However, in behave of diminishing the possibility of cycle occurrence, the TF-ORM model presents a pre-defined property, defined in the base-role of all classes (a role that shall be defined in all classes and where the global characteristics of all objects of that class are defined). This property, called *cycle_alert*, has the role of controlling cycles, and of alerting the responsible agent in case of detecting a possible occurrence. The domain of this property is a set of pairs – the name of a role and the number of active instances of that role. The

responsible agent of the process shall control this property, and identify possible cycles.

7.5 Deadlocks

Rules to prevent deadlocks are more difficult to represent especially because of the possibility of constructing temporal logic conditions to constraint the transition rules. What can be done is to associate a time period to a state that may present deadlocks. After this time period elapsed, another rule would be executed, to undo the deadlock.

To avoid deadlock exceptions, the TF-ORM model was extended with the definition of a timing class. This class, pre-defined in all the constructed models, can be used to count the elapsed time. This is done creating an instance of this class with the maximum waiting time passed as parameter. The behaviour of the timing class is the following: once an instance is created, this instance counts the elapsed time and when the time parameter is reached, sends a message *interrupt* to the prior class.

The complete treatment of deadlock is the following. When the possibility of a deadlock occurrence in the state *state_1* of *activity_1* is identified, each time this state is reached a message is sent to the timing class, with the maximum time this activity shall wait for another event (represented by a new state transition). This message has the following form:

msg(timing(<class.role>, <state>, <waiting time>, <temporal granularity>))

The timing class begins to count the elapsed time. When the waiting time is reached, an interrupt message is sent to the sending class/role (representing the activity). If this role is still in the same state, a deadlock really happened, and a state transition to a recovery state is provided. On the other hand, of the role evolved and is in another state, the interrupt message will not be receive, and will not affect the activity. An example of a possible deadlock control in state *dl_state* is the set of transition rules:

state(x), msg(m_in) $\Rightarrow$ *msg(m_out), msg(timing(C.R, dl_state, 5, min)), state(dl_state)*;

state(dl_state), msg(waited_msg) $\Rightarrow$ *state(y);*

state(dl_state), msg(interrompt) $\Rightarrow$ *state(recovery_state);*

7.6 Waiting message receipt acknowledgement

The sending of a message to another activity does not guarantee that the message is received. This can cause a problem in the whole workflow, when the receipt of the message is fundamental. When using TF-ORM to represent a workflow, messages are sent and there is really no certainty of the reception, once a transition rule is only executed if the message is received in a specific state, and additionally if the transition condition is satisfied. If one of these (state or transition conditions) is not satisfied, the message is not received.

To guarantee that a message is received, the TF-ORM model shall represent the following:

- The role that sends the message shall evolve to a waiting state, and stay in this state for a defined time (using the timing class) until a confirmation of the message reception is received;
- If the message is correctly received, the receiving role sends an acknowledge message to the sending role and continues his own evolution;
- If the message is not received, after the maximum waiting time is elapsed the role evolves from the waiting state to a state that will solve this problem – probably sending another message in place of the first one.

7.7 Information to be used for failure recovery

Two aspects shall be clearly identified in failure recovery processes:

- Who (what *agent*) is in charge of the recovery, taking the necessary decisions to overcome the failure; and
- What kind of *information* is needed in this process?

An agent must do the recovery of a system failure – these failures can not be predicted, in order to mechanise the recovery. To guarantee that an agent is available in case a failure occurs, the TF-ORM model requires the definition of responsible agents for the whole workflow. Each identified process must have a responsible agent, so as each activity. In case of a failure occurrence, first the activity responsible tries to recover the execution and, if this is not possible within the activity domain, the process responsible takes charge.

The recovery of a failure will restart the executing activities from a previous execution point, after solving the problem that caused the failure. This process is known as *rollback*. To enable this, additional information is needed. A form of doing rollback that has proved to be efficient is to store the past states assumed by the activities and processes, with temporal information associated (time when each one of these states was assumed). This is already a feature of the TF-ORM model – being a temporal model, temporal information (transaction and valid time) is associated to every information (states and property values), and all the past information is supposed to be kept in the temporal database modelling the workflow. This enables the agent responsible for the recovery to, analysing the stored information, choose a past temporal instant, delete information defined after that instant (considering the transaction time of each stored value), this way restoring a past state of the considered process or activity.

An alternative way of implementing system recovery would be defining in each class a property that would store the activation history of each instance of that class, also associated to the correspondent transaction time. Analysing the evolution of these instances, the responsible agent would choose how the recovery should be made.

8 Conclusion

To make the modelling of workflow systems more effective it is necessary to improve the conceptual level specification with an unified model able to represent its internal behaviour (cooperation and interaction among tasks) and the relationship with the environment (designation of tasks to the executors).

In this paper, a technique of conceptual modelling of workflow using the model TF-ORM is presented. This technique is intended to specify the workflow, and to support the workflow implementation. The TF-ORM model was extended, with the definition of new syntactic constructions, to allow the representation of workflow characteristics in an efficient way.

When using TF-ORM in workflow modelling all the processes involved in a workflow are represented, together with their relationships and the coordination of their execution. In addition, the data flow between these processes is also defined, and agents are identified as responsible for each process. A set of temporal logic rules incorporates a solid formalism to express reactive computations, usually influenced by events external to the workflow, like exceptions and pre and post-conditions associated to process execution. The final model is a formal model, and can be used to analyse the workflow, identifying possible definition problems that can be solved prior to the implementation of the workflow.

Most of the existent workflow modelling techniques do not represent formally the work portion that needs human intervention (unstructured work). TF-ORM allows the representation of the processes' unstructured work portion through agent classes. Agent classes represent people acting in the system. Agents have an own functionality that is the human decision. A decision represents the result of a formally undefined process executed by an agent. The structures of the resources (data, documents) are represented by the resource classes. The process classes integrate these agents and resources, describing the organisation of the work executed in the application and the co-operation among agents.

The TF-ORM model proved to be a powerful tool for workflow modelling, due to the following aspects:

- It is a formal model, allowing the complete modelling of the workflow, including the possibility of validation of the whole process;

- not only the structured part of the workflow can be represented, but also the decisions involved in the processes;
- All the possible synchronism among activities execution may be represented, using rules expressed in temporal logic;
- Allows the representation of the communication among processes and activities;
- The relationship between activities belonging to different processes may also be represented, fact that is not always possible in other workflow modelling methods;
- based on the role concept, allows the representation of the different roles an agent can perform, in a natural way;
- Exceptions handling and information to be used in failure recovery may be represented using this model, in a convenient way.

The use of a new object-oriented model to specify an application does not imply the need of implementing a special database management system for this data model. Existing commercial DBMSs can be used, if a mapping from this data model to the data model of the adopted DBMS is provided. In [Oliveira 95] the implementation of TF-ORM is discussed using three different DBMSs: the commercial object-oriented O_2 DBMS [O2T 91], the research system Postgres [Postgres 94] and the commercial relational DBMS Ingres [Ingres 91].

A complete environment based on TF-ORM is under development [Edelweiss 00], including a modelling tool, the mapping of the TF-ORM model to a commercial database, and a query interface using the TF-ORM query language [Edelweiss 94].

9 References

[Aalst 95] Aalst, W.M.P. *Petri-net-based Workflow Management Software.* Eindhoven University of Technology, 1995.

[Alonso 97] Alonso, G.; Agrawal, D.; El Abbadi, A.; Mohan, C. Functionality and Limitations of Current Workflow Management Systems. *IEEE Expert*, v.12, n.5, 1997.

[Baresi 99] Baresi, L; Casati, F.; Castano, S.; Ceri, S.; Fugini, M. G.; Mirbel, I.; Pernici, B.; Pozzi, G. *WIDE Workflow Development Methodology***.** University of Twente, Netherlands, 1999.

[Belkhatir 94] N. Belkhatir, W.L. Melo. "Tempo: defining software processes in an approach based on objects with roles", Proceedings of the ORM-1 Conference, July 4-7, 1994, Magnetic Island, Australia. Brisbane: University of Queensland, 1994. p.157-166.

[Edelweiss 00] Edelweiss, N.; Hübler, P.N.; Moro, M.M.; Demartini, G. A Temporal Database Management System Implemented on Top of a Conventional Database. To be published in the *Proceedings* of the XX International Conference of the Chilean Computer Science Society - SCCC 2000, to be held 16-18 November 2000 in Santiago, Chile.

[Casati 95] Casati, F.; Ceri S.; Pernici, B.; Pozzi, G. Conceptual Modeling of Workflows. *Proceedings* of OO-ER Conference. Gold Coast, Australia, 1995.

[DeAntonellis 91] Antonellis, V.; Pernici, B.; Samarati, P. F-ORM Method: a F-ORM Methodology for reusing specifications. In: Assche, F.V.; MOoulin, B.; Rolland, C., *Object Oriented Approach in Information Systems*, Amsterdam, North-Holland, 1991. p.117-35.

[Eddis 93] Eddis, C.; nutt, g.j. Modelling and enactment of Workflow Systems. *Applications and Theory of Petri Nets*. Berlin: Springer-Verlag, 1993. p.1-16.

[Edelweiss 93] Edelweiss, N., Oliveira, J.P.M. and Pernici, B., *An Temporal Object-Oriented Model*, Proceedings, 5th International Conference on Advanced Information Systems Engineering, Paris, France, June 8-11, Lecture Notices in Computer Science n. 685, pp.397-415.

[Edelweiss 93a] Edelweiss, N., Oliveira, J.P.M. and Castilho, J.M.V, *Temporal Logic Language for Temporal Conditions Definition*, Proceedings, 13th International Conference of the Chilean Computer Science Society, La Serena, Chile, Oct. 14-16, pp.163-178.

[Edelweiss 94] Edelweiss, N., Oliveira, J.P.M. and Pernici, B., *An Object-oriented approach to temporal query language*, Proceedings, 5^{th} Database and Expert Systems Applications Conference, Athens, Greece, Sept. 7-9, Lecture Notes in Computer Science n. 856, pp.225-235.

[Edelweiss 97] Edelweiss, N.; Oliveira, J. P.M. Roles Representing the Evolution of Objects. *Proceedings* of the Argentine Symposium on Object Orientation of the 26^{th} Jornadas Argentinas de Informática e Investigación Operativa - JAIIO, Aug. 11-12, 1997, Buenos Aires, Argentina. p.57-65.

[Edelweiss 98] Edelweiss, N.; Nicolao, M. Workflow Modeling: Exception and Failure Handling Representation. *Proceedings* of the XVIII International Conference of the Chilean Computer Science Society - SCCC'98, Antofagasta, Chile, November 09-14, 1998. Los Alamitos: IEEE Computer Society, 1998. p.58-67.

[Eder 95] Eder, J.; Liebhert, W. The Workflow Activity Model WAMO. *Proceedings* of the 3^{rd} Int. Conference on Cooperative Information Systems, Vienna, Austria, My 1995. p.87-98.

[Eder 96] Eder, J.; Liebhert, W. Workflow recovery. *Proceedings* of the 1^{st} IFCIS Int. Conference on Cooperative Information Systems, Brussels, Belgium, June 1996. IEEE Computer Society Press. p.124-134.

[Eder 98] Eder, J.; Liebhert, W. Contributions to exception handling in Workflow Management. *Proceedings* of the EDBT Workshop on Workflow Management Systems, Valencia,

Spain, March 27-28, 1998. p.3-10.

[Ellis 95] Ellis, C.; Keddara, K.; Rozenberg, G. Dynamic change within Workflow Systems. *Proceedings* of the Conference on Organization Computing Systems – COOCS, Milpitas, CA, Aug. 1995. Comstock, N.; Ellis, C. (eds.), ACM Press, 1995. p.10-21.

[Georgakoupoulos 95] Georgakoupoulos, D.; Hornick, M.; Sheth, A. An Overview of Workflow Management: from process modeling to workflow automation infrastructure. *ACM Distributed and Parallel Databases*, n.3, p.119-153, Sep. 1995.

[Heinl 98] Heinl, P. Exceptions during workflow execution. *Proceedings* of the EDBT Workshop on Workflow Management Systems, Valencia, Spain, March 27-28, 1998. p.11-20.

[Ingres 91] Ingres Corporation. *Ingres SQL Reference Manual.* Release 6.4, 1991

[Jablonski 96] Jablonski, S.; Bussler, C. *Workflow Management, Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, 1996.

[Jensen 98] Jense, C.S. et al. The Consensus Glossary of Temporal Database Concepts - February 1998 Version. In: *Temporal Databases Research and Practice*. O. Etzion, S. Jajodia and S. Sripada (eds.) Springer-Verlag. Berlin Heidelberg 1998. pp. 367-405.

[Joosten 94] Joosten, M. M. Stef - *Trigger Modelling for Workflow Analysis* - Design Methodology Group - Center for Telematics and Information Technology, University or Twente, P.O. Box 217, 7500 AE Enschede, the Netherlands.

[Joosten 94a] Joosten, S. Trigger modelling for Workflow Analysis. *Proceedings* Workflow Management, Challenges, Paradigms, and Products – COM'94, Vienna, Austria, Oct. 1994.

[Leymann 98] Leymann, F. *Production Workflow Systems.* Tutorial at the 6^{th} Intl. Conf. On Extending Database Technology, Valencia, Spain, Mar. 24, 1998. 90p.

[Missikoff 98] Missikoff, M.; Pizzicannella, R. An Object-oriented approach to WorkFlow Modeling. *Proceedings* of the EDBT Workshop on Workflow Management Systems, March 27-28, 1998, Valencia, Spain. O. Bukhres, J. Eder, S. Salza (Eds.) p. 57-65.

[Nicolao 98] Nicolao, M.; Edelweiss, N. Workflow Modelling using a Temporal Object-Oriented Model. *Proceedings* of the EDBT Workshop on Workflow Management Systems, March 27-28, 1998, Valencia, Spain. O. Bukhres, J. Eder, S. Salza (Eds.) p. 71-79.

[Oliveira 95] Oliveira, J.P.M. et al., On the Implementation of an Object-Oriented Temporal Model using Object-Oriented and Relational DBMSs. *Workshop Proceedings* of the 6^{th} Database and Expert Systems Applications - DEXA'95, London, United Kingdom, Sept. 1995. Norman Revell, A. Min Tjoa (Eds.). p.35-44.

[Oliveira 98] Oliveira, J.P.M.; Nicolao, M.; Edelweiss, N. Conceptual Workflow Modelling for Remote Courses. *Proceedings* of the International IFIP Conference on Distance Learning, Training and Education – TELETEACHING'98, Viena, Austria, 1998.

[O2T 91] O_2 TECHNOLOGY. *The O_2 Programmer's Manual*, 1ed. Versailles: O_2 Technology, 1991.

[Pernici 90] Pernici, B., 1990, *Objects with Roles*, Proceedings, ACM/IEEE Conference on Office Information Systems, Cambridge, MA, April 25-27, SIGOIS Bulletin, v.11, n.2-3, pp.205-15.

[Postgres 94] The Postgres Group. *The Postgres User Manual*, Version 4.2, The Postgres Group, Computer Science Div., Dept. of EECS, University of California at Berkeley, 1994.

[Reichert 97] Reichert, M.; Dadam, P. A Framework for dynamic changes inWorkflow Management Systems. *Proceedings* of the 8^{th} Int. Workshop on Database and Expert Systems Applications – DEXA'97, Toulouse, France, Sept. 1997. Wagner, R.R. (ed.), IEEE Computer Society Press, 1997. p.42-48.

[Saastamoinen 95] Saastamoinen, H.; White, G.M. On Handling exceptions. *Proceedings* of the Conference on Organizational Computing Systems – COOCS, Milpitas, CA, Aug. 1995. Comstock, N.; Ellis, C. (eds.), ACM Press, 1995. p.302-310.

[Tang 98] Tang J.; Hwang, S.-Y. A Scheme to specify and implement ad-hoc recovery in workflow systems. *Proceedings* of the 6^{th} Int. Conf. On Extending Database Technology – EDBT'98, Valencia, Spain, March 23-27, 1998. Springer, 1998. p.484-498. (Lecture Notes in Computer Science no.1377).

[Van Stiphout 98] Van Stiphout, R. et al. TREX: Workflow Transations by means of Exceptions. *Proceedings* of the EDBT Workshop on Workflow Management Systems, Valencia, Spain, March 27-28, 1998. p.21-26.

[Vossen 96] Vossen, G.; Weske, M. (Eds.) *Business Process Modelling and Workflow Management, Models, Methods, and Tools.* International Thomson Publishing, Bonn, 1996.

[Weissenfels 98] Weissenfels, J.; Muth, P.; Weikum, G. Flexible Worklist Management in a light-weight Workflow Management System. *Proceedings* of the EDBT Workshop on Workflow Management Systems, Valencia, Spain, March 27-28, 1998. p.29-38.

[WFMC 94] *Workflow Management Coalition - The Workflow Reference Model.* Document Number: WFMC-TC00-1003, Issue 1.1, Nov.1994.

[WFMC 99] *Workflow Management Coalition - Terminology & Glossary*, Document Number: WFMC-TC-1011, Issue 2.0, Jun. 99 Available at http://www.wfmc.org.

[WFMC 96a] *Workflow Management Coalition - The Workflow Reference Model*, Document Number: WFMC-TC00-1003, Issue 1.1, Nov.1994.

[Wieringa 91] Wieringa, R.; De Jonge, W. *The identification of objects and roles - object identifiers revisited*. Technical Report IR-267, Vrije University, Amsterdam, Holland, Dec. 1991.

Appendix

```
Process class (
```

```
TRIP,
< base_role,
   /*  properties and rules that apply to all other roles  */
   responsible agent = PERSON.Manager,
   executing agent = PERSON.Employee,
   static properties = { reg_nr: integer },
   dynamic properties = { contact_date: date, client: PERSON.Client },
   rules = {
      init: msg(add_object) ⇒ state(active),
      start: state(active) ⇒  msg(add_role(define_route)),
        msg(allow_role(fligth_reserve)), msg(allow_role(hotel_reserve)),
        msg(allow_role(car_reserve)), ... }
>,
< define_route,
   responsible agent = PERSON. Employee,
   executing agent = PERSON.Employee,
   static properties = { ... },
   dynamic properties = { cities: list of string, ... },
   states = {active, defining, defined},
   messages = {
      res_flight(Cit: list of strings) to reserve_flight,
      res_hotel(Cit: list of strings) to reserve_hotel,
      res_car(Cit: list of strings) to reserve_car },
    decisions = {
      ok from PERSON.Client,
      add_city(City: string) from PERSON.Client, ... },
   rules = {
      init: msg(add_role) ⇒ state(defining),
      cities: state(defining), decision(add_city(City)) ⇒ state(defining),
      /*  rule representing a fork:  */
      done: state(defining), decision(ok) ⇒ msg(res_flight(cities)),
            msg(res_hotel(cities)), msg(res_car(cities)), state(defined)}
>,
< reserve_flight ... >,
< reserve_hotel ... >,
< reserve_car ... >,
< program_print,
   messages = {
      flight_ok (Price: real) from reserve_flight,
      hotel_ok (Name: string, Price: real) from reserve_hotel,
      car_ok (Price: real) from reserve_car  },
   states = { waiting, printing, done },
   rules = {
      init: msg(add_role) ⇒ state(waiting),
       /* rule representing a join: */
      start: state(waiting), {msg(flight_ok(P)),
        msg(hotel_ok(N,P)), msg(car_ok(P)) } ⇒ state(printing),
      ...  }
>,
... )
```