

Implementation of Ontology-driven Workflow Flexibilization Mechanisms

Tatiana A. S. C. Vieira

Marco A. Casanova

Luis G. Ferrão

Department of Informatics, Pontifical Catholic University of Rio de Janeiro, Brazil
Rua Marquês de São Vicente, 225 - Rio de Janeiro, RJ - Brazil - CEP 22.453-900
{tati, casanova, ferrao}@inf.puc-rio.br

Abstract. Workflow management systems usually interpret a workflow definition rigidly. However, there are real life situations where users should be allowed to deviate from the prescribed static workflow definition for various reasons, including lack of information and unavailability of the required resources. To flexibilize workflow execution, this paper proposes mechanisms that allow execution to proceed in the presence of incomplete information, by adopting presuppositions, and in the presence of negative information, by suggesting execution alternatives. This paper also presents an architecture for the workflow system, which is driven by ontologies that capture semantic relationships between workflows, resources and users. The focus is on the matching module instance, the component responsible for finding alternatives for workflows, resources and users.

1 Introduction

Workflow management systems received considerable attention lately, motivated by their wide spectrum of applications. Standardization efforts are under way in the context of consortia, such as WfMC, OASIS and W3C. In particular, WfMC was created in 1993 by several companies with the special purpose of standardizing workflow model specification [Hollingsworth, 1995]. Among other contributions, these consortiums efforts resulted in new workflow definition languages and coordination protocols.

Requirements for workflow management systems comprise a long list. Among the requirements, we may highlight distributed execution, cooperation and coordination, and synchronization, that model the way user communities work cooperatively to perform a given task [Georgakopoulos et al., 1995] [Alonso et al., 1997].

This paper addresses what we collectively call *flexible execution*. Briefly, workflow management systems usually interpret a workflow definition rigidly. However, there are real life situations where users should be allowed to deviate from the prescribed static workflow definition for various reasons, including lack of information and unavailability of the required resources.

To achieve flexible execution, we propose in this paper: (i) a *mechanism to handle presuppositions* that allows execution to proceed in the presence of incomplete information; (ii) a *mechanism for choosing alternatives* to subworkflows, resources and users, thereby allowing workflow execution to proceed when the predefined subworkflow or resource, or the pre-assigned user is unavailable (i.e., in the presence of negative information) and when the workflow definition is abstract. These mechanisms represent

the component substitution idea applied to the case of workflow execution, where one component can be seen as a subworkflow, a resource or a user [Antonellis et al., 2003].

Such mechanisms use *workflow ontologies*, as described in Section 5 to: (i) compute presuppositions when the required information is unavailable; (ii) find alternative subworkflows, resources and users when the predefined subworkflow or resource, or the pre-assigned user is unavailable, or even when the workflow definition is abstract; (iii) assess the presuppositions made and the alternatives taken when workflow execution terminates.

It must be noted that our flexibilization mechanisms are not intended to allow users to directly interfere with workflow execution. The mechanisms rather use previous semantic information about workflows, their resources and the available users, and the current state of workflow execution, to suggest better alternatives to the user, or make educated guesses about missing data, that allow execution to proceed when otherwise it would have been stopped. In addition, the mechanisms make possible a more flexible modelling, allowing the workflow designer to reference abstract definitions (see Section 4).

The evolution of workflow systems towards less restrictive coordination approaches is discussed in [Nutt, 1996]. Most of the flexibilization mechanisms proposed in the literature suggest to dynamically change the workflow structure at runtime [Casati et al., 1996].

Rule-driven frameworks that support structural changes in a workflow, by dictating how tasks can be inserted or removed from the workflow, at runtime, are presented in [Joeris, 1999]. The flexibilization mechanisms described in [Weske, 1998] [Bider and Khomyakov, 2000] also offer the user the possibility to include, remove or even stop activities during workflow execution. The mechanisms proposed in this paper follow a different strategy by allowing execution to proceed in the presence of incomplete or negative information, with minimal user intervention. The idea is, in the case of the mechanism for choosing alternatives, to give to the user the possible components (workflows, resources and users) to substitute the “failed” one.

The OPENflow system [Halliday et al., 2001] offers two distinct flexibilization approaches: flexibility by selection and flexibility by adaptation. Flexibility by selection allows several paths to be included in the workflow description, but only one path is chosen at runtime. Flexibility by adaptation leads to workflow adaptation, at the instance level, when changes occur in the workflow structure at runtime (inclusion of one or more execution paths). Our approach does not cover flexibility by selection, but it achieves flexibility by adaptation by using semantic information about workflow description to partly guide instance execution.

The flexibilization mechanisms in [Grigori et al., 2001] accounts for the cooperation between users and the anticipation of task execution, following the COO approach. COO offers a special transaction model for cooperative systems that deviates from the conventional start-end synchronization model [Canals et al., 1998]. These mechanisms allow some activities to be anticipated according to the workflow description, but the termination order of the activities is preserved. Our approach allows task anticipation, based on additional semantic information.

The construction of workflow schemas from a standard set of modelling constructs is proposed in [Mangan and Sadiq, 2002]. This is partially done at design time, and

completed at runtime, according to selection, termination and workflow definition constraints. These constraints dictate how each fragment of work can be included in the workflow, under what conditions the workflow instance can be terminated and what conditions must hold during workflow definition. Flexibilization is achieved, in this case, by leaving workflow definition to be completed at runtime, according to the specified constraints. To some extent, our mechanism for choosing alternatives achieves a similar effect, as discussed in Section 4.

Lastly, the idea behind the flexibilization mechanisms proposed in this paper is very similar to the query relaxation strategy used in the CoBase system [Chu et al., 1994] and in the CoSent system [Chu and Mao, 2000]. CoBase is a database system that uses the idea of cooperative queries. When queries are submitted to the system, COBASE analyses an hierarchy of additional information to enhance the query with relevant information. The information added to the query is bounded by a maximum semantic distance from the information present in the original query. This query modification mechanism is similar to that we propose in this article to deal with negative or incomplete information. In our proposal, the workflow description is enhanced with additional information available in the workflow ontology, as discussed in Section 5.

This paper is organized as follows. Section 2 outlines the basic distributed workflow execution model. Section 3 presents the example which will be used in this paper. Section 4 describes the flexibilization mechanisms we propose. Section 5 presents a workflow ontology, based on the example in Section 3. Section 6 briefly outlines an implementation based on Web services technologies. Finally, Section 7 contains the conclusions of this work.

2 Basic Execution Model

In the basic workflow execution model proposed, as presented in Figure 1, when a user U starts a workflow instance W , he becomes the *coordinator* of W . User U may delegate a nested subworkflow W' of W to another available user U' , perhaps with permission to redistribute subworkflows. Then, U' becomes the coordinator of W' . We call U the *coordinator* of U' and U' a *subordinate* of U . Hence, we effectively have a *coordination hierarchy* for W , rooted at U , which is also called the *central coordinator* of the hierarchy. Each user is his own coordinator or is coordinator of the subordinated users in the hierarchy.

The system assigns an identifier to each workflow or subworkflow W'' and creates a common data space B'' for W'' , called here a *blackboard*. Among other data, B'' will keep the values of all variables that W'' uses, the identifier of W'' and the address of the coordinator U'' of W'' . The system treats W , with its nested subworkflow instances, as a *workflow transaction*, in the sense of [Worah and Sheth, 1997].

In summary, our basic distributed workflow execution model can be viewed as a hierarchy of three basic components: coordinators, workflow instances, and blackboards. This execution model dilutes the burden of controlling the distributed execution of a workflow instance among a hierarchy of coordinators and avoids the danger of transforming the central coordinator into a bottleneck.

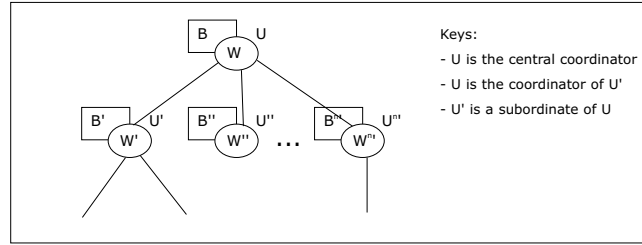


Fig. 1. Basic workflow execution model proposed.

3 A Brief Example

The example presented in this section is inspired on a real emergency plan, defined in a large (paper) document. The plan has been translated into a workflow that runs under the InfoPAE system [Casanova et al., 2002]. Albeit very simplified and schematic, the example retains the essential characteristics of the original plan.

Consider the problem of cleaning coastal areas affected by an oil spill. This problem is addressed by defining a set of cleaning procedures that take into account the oil type and the coastal area characteristics. Table 1 provides an schematic example of cleaning procedures when the type of coastal area is Sand Beach and the types of oil are simply Type I through Type V. Cells are filled with a weight indicating the environmental impact of each of the procedures: 0.00 indicates the smallest environmental impact, 0.25 some impact, 0.50 a significant impact, 0.75 the greatest impact, and 1.00 inapplicable.

Oil Type	Type I	Type II	Type III	Type IV	Type V
ND: Natural Degradation	0.00	1.00	1.00	1.00	1.00
UA: Use of Absorbents	1.00	0.25	0.00	0.00	0.00
VC: Vacuum Cleaning	1.00	0.00	0.00	0.00	0.00
CL: Cold, Low Pressure Cleaning	1.00	0.00	0.00	0.25	0.25
CH: Cold, High Pressure Cleaning	1.00	0.25	0.25	0.25	0.25
HL: Hot, Low Pressure Cleaning	1.00	1.00	0.50	0.50	0.50
HH: Hot, High Pressure Cleaning	1.00	1.00	0.50	0.50	0.50
PC: Vapor Cleaning	1.00	1.00	0.75	0.75	0.75

Table 1. Environmental impact of cleaning procedures for sand beaches.

Now, suppose that the user comes to a point in the overall emergency plan execution where he needs to select a cleaning procedure for a sand beach affected by an oil spill. The user (or the workflow management system) can then look up in Table 1 to select (or invoke) the best procedure to clean the beach. For example, if the oil is of Type II, the best procedures are “VC: Vacuum Cleaning” and “CL: Cold, Low Pressure Cleaning”.

However, if he has no information about the oil type, Table 1 becomes useless. In this case, he may take an educated guess and assume, say, that the oil is of Type II. He will then proceed with the emergency plan based on this assumption. Moreover, if he cannot execute the best procedures for oil Type II (those with weight 0.00), because a

predefined resource is unavailable, then he may resort to “UA: Use of Absorbents” and “CH: Cold, High Pressure Cleaning”, which are the second best choices (those with weight 0.25).

4 Flexibilization Execution Mechanisms

This section describes two mechanisms that allow workflow execution to proceed in the presence of incomplete information, by adopting presuppositions, and in the presence of negative information and of an abstract definition, by suggesting alternatives for workflows, resources and users.

Such mechanisms use *workflow ontologies* (see Section 5) that provide the required additional information, in the form of: (i) *presupposition rules* to compute presuppositions, when the required information is unavailable; (ii) *consistency rules* to assess the presuppositions made when workflow execution terminates; (iii) *semantic proximity properties* that help finding alternative subworkflows, resources and users, when the predefined subworkflow or resource, or the pre-assigned user is unavailable and when the workflow definition is abstract; and (iv) *semantic rules*, that dictated the way alternatives can be found, for instance, based on the last parallel workflow executed and on the execution context. These semantic rules basically define the semantic proximity properties of a pair of nodes in the workflow ontology, based on the execution context.

The mechanism to handle presuppositions is invoked whenever the required information is unavailable and essentially interprets the presupposition rules. It may be invoked in two distinct points of the execution. First, when delegating a subworkflow instance W' of a workflow instance W to another user U' , the coordinator U of W may find out that the value of a variable V , required to test the pre-condition C' of W' , is missing. In this case, U executes the presupposition mechanism to find out a value p for V , tests C' with p , and delegates W' to U' , if C' succeeds when V receives p as value. Second, when executing W , he may also miss the value of some variable T . Then, U may again invoke the presupposition mechanism to find out a value for T .

The mechanism for choosing alternatives operates in two different modes, which are essentially equivalent, but require slightly different interpretations (and modelling) of the semantic proximity properties.

In the first mode, the workflow definition contains specific (concrete) subworkflows, resources and users, but it may indicate that the assignment is flexible. In this case, the mechanism is invoked when the predefined subworkflow or resource, or the pre-assigned user is unavailable (i.e., in the presence of negative information). It uses the semantic proximity properties to: (i) find a semantically equivalent subworkflow such that the required resources and the assigned user are available; (ii) find semantically equivalent resources that are available; (iii) find a user, with equivalent capacities, that is available. It is worth noting that these three choices are mutually dependent. For instance, if an alternative subworkflow is considered, one must verify if the required resources and users are available. Likewise, if an alternative user is considered, one must check if he can really execute that subworkflow and if the resources are available.

In the second operation mode, the workflow definition specifies an abstract subworkflow or task, leaving it to the mechanism, or in some extent to the user itself, to decide,

at run time, which is the best alternative that can be executed, that is, whose resources and users are all available. This operation mode causes the workflow modelling to be made in 2 different ways: (1) a concrete modelling, where the workflow definition is totally concrete; and (2) an abstract modelling, that implies a more flexible approach, causing the mechanism for choosing alternatives to be invoked at runtime.

Moreover, in both operation modes, typically more than one alternative is possible. Hence, the semantic proximity properties must provide some form of cost estimate, and the mechanism for choosing alternatives must be equipped with a cost model that is invoked to select the best alternative, or at least to guide some heuristics that will select a reasonable alternative.

The mechanism for choosing alternatives can be made more sophisticated by taking into account the execution context. Briefly, alternative activities should be chosen in a way that favors parallelism and optimizes the use of resources. For instance, the mechanism should avoid replacing an activity A by an alternative activity B, if B will block another activity C being executed or that was scheduled to run in parallel with A. As another example, if the alternative B chosen will be followed by an activity D then, if possible, the resources B uses should have a non-trivial intersection with the resources D requires (intuitively, B will be able to handle to D a number of resources, thereby reducing the setup time of D).

It is important to note that semantic rules must be followed during execution to allow the right alternatives to be found. These rules can indicate, for example, that a specified alternative can just be taken according to the value of a variable managed by another fragment execution. If the value is not as expected, this alternative must be discarded.

Finally, when a subworkflow instance terminates, the workflow management system must invoke the consistence rules to analyze the results produced, if presuppositions were made.

5 Workflow Ontology

Using the basic OWL terminology [W3C, 2004], the idea behind a simple decision table, such as that presented in Section 3, can be extended and generalized by considering a *workflow ontology* (and other additional information), with the following major classes and properties:

- subclasses *Workflow*, *Resource* and *User* of the class *owl:Thing*, with the obvious meaning;
- subclasses *Abstract* and *Concrete* of the classes *Workflow*, *Resource* and *User*, that represent abstract and concrete objects, respectively (see the example in Figure 2);
- property *is-implementation-of* that maps concrete to abstract objects, and the inverse property *is-implemented-by*;

- class *Variable*, that represents variables concerning each workflow, with the properties *has-value* and *has-default-value*, the last to be used by the mechanism to handle presuppositions;
- property *has-name*, applicable to the classes *Workflow*, *Resource*, *User* and *Variable*, with the obvious meaning;
- property *is-related-to*, that maps *Variable* to *AbstractWorkflow* and its inverse property *relates*;
- property *is-responsible-for* that maps *User* objects to *Workflow* objects, and the inverse property *is-responsibility-of*;
- property *requires* that maps workflows to resources, and the inverse property *is-required-by*;
- property *is-available* that maps *ConcreteUser* objects and *ConcreteResource* objects to a Boolean value and that intuitively indicates whether the user or resource is available or not.

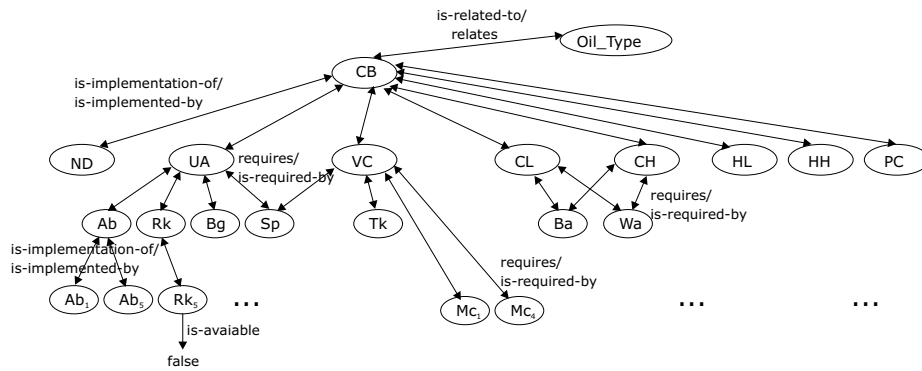


Fig. 2. Fragment of a workflow ontology that shows the beach cleaning procedures for oil Type II.

As an example, consider the fragment of a specific workflow ontology that captures the oil cleaning procedures when the coastal area is Sand Beach. Rather than formalizing the fragment in RDF/XML or OWL, for example, we represent it as a simplified RDF graph (see Figure 2), where:

- *CB* is an abstract workflow that represents all cleaning procedures for Sand Beaches;
- *ND*, *UA*, *VC*, *CL*, *CH*, *HL*, *HH*, and *PC* (see Table 1) are concrete workflows that model specific cleaning procedures;

- *CB* and *ND*, *UA*, *VC*, *CL*, *CH*, *HL*, *HH*, and *PC* are related by the *is-implementation-of* property, with the appropriate weights (semantic proximity properties) presented in Table 1 defined through the semantic rules;
- *Ab*, *Rk*, *Bg*, *Sp*, *Tk*, *Ba*, and *Wa* are abstract resources;
- *Ab*₁, *Ab*₅, *Rk*₅, *Mc*₁, and *Mc*₄ are concrete resources of *Ab*, *Rk*, and *Mc*, respectively. The weight of the relationship is defined through the semantic proximity properties;
- *Oil_Type* has as values *Type I*, *Type II*, *Type III*, *Type IV* and *Type V*.

The weights presented in Table 1 are specified through semantic rules that show the dependency between the semantic proximity properties and the execution context.

At design time, this workflow ontology fragment would be used as follows. Rather than using a concrete workflow in the workflow definition, the designer may use an abstract workflow, such as *CB*, and leave it to the mechanism for choosing alternatives the decision of what specific cleaning procedure to use. We can call this modelling the abstract modelling, in opposition to the concrete modelling. Hence, this example corresponds to the second mode of operation of the mechanism for choosing alternatives, alluded to in Section 4.

Furthermore, the designer may indicate when the mechanism for handling presuppositions can be invoked. For instance, for workflows that can not be compensated it is not recommended to use flexibilization mechanisms, because if anything goes wrong, it will not have a way to make it ends execution correctly.

At run time, the workflow management system will indicate which resources and user (teams) are available at a given point in time. This is done through the *is-available* property, which value can be true or false.

When workflow execution reaches a point where the oil type is required, but unknown, the presupposition mechanism may be invoked to determine what oil type must be assumed. Therefore, workflow execution may conditionally proceed based on the presupposed oil type.

Likewise, when workflow execution reaches an abstract subworkflow *A*, the mechanism for choosing alternatives will invoke a search procedure that traverse the ontology to select the concrete workflow that is an implementation of *A*, whose required resources and users are available and that has the smallest *overall weight*. The mechanism computes the overall weight for a given concrete workflow from its own weight and from the weights of the resources it requires (including user teams), according to the semantic proximity properties and to a given cost model, whose range must also be the interval [0,1]. Besides, the semantic rules indicate which alternatives can be chosen. The cost model can be weighted, based on the importance of each object in the workflow.

Finally, note that the workflow definition should indicate which workflow ontology the flexibilization mechanisms will use, but it should not contain the ontology itself. In other words, the workflow ontology is an independent object that can be referenced (and re-used) by different workflow definitions, as also can the semantic rules. Indeed, this

very intuitive re-use of ontologies considerably simplified the definition of emergency plans alluded to in Section 3.

6 Implementation

This section outlines the architecture of the system, shown in Figure 3. Briefly, the system consists of a collection of *execution nodes*, responsible for workflow execution, and of one (or more) *ontology management node*, that offers services to store and manipulate the workflow ontology.

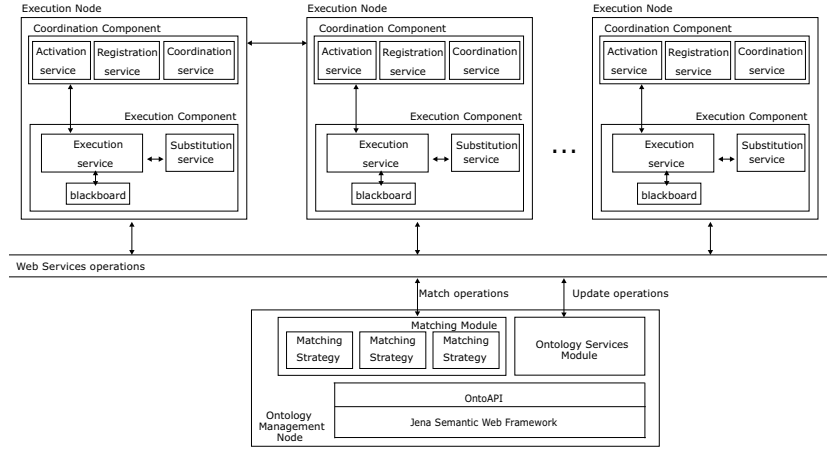


Fig. 3. The ontology-driven architecture proposed for flexible workflow execution.

6.1 Ontology Management Node

The *ontology management node* is decomposed into the matching module and the ontology services module.

The *matching module* is an instance of a framework designed to solve matching problems among individuals of a given domain, such as “find the resource whose semantic distance from resource B is the smallest possible”.

This framework has the following major characteristics:

1. independence from the application domain knowledge: achieved by the *Domain Knowledge* component, which encapsulates the matching functions required to solve the matching problem. Such functions define *similarity* values (the semantic proximity property values) between individuals in the domain, covering 3 cases: one individual matching one individual (1-1); one individual matching many individuals (1-N); and many individuals matching many individuals (N-N);
2. independence from the application data model: achieved by the *Model* component, which offers an interface that can be extended to essentially translate the application data model to the data model the framework implements. The interface can

be extended to cover the most popular data models used to design ontologies and databases;

3. independence from the matching strategies: achieved by the *Matching Strategy* component, which offers an interface that can be extended to cover new matching algorithms. Such algorithms use the information contained in the Domain Knowledge component to match individuals of the application domain.

Recall that, in the context of the workflow flexibilization mechanisms, the workflow ontology plays the role of the application data model. We therefore have the following major adaptations of the framework.

First, the matching functions define similarity values between pairs of workflows, pairs of resources and pairs of users. The matching module uses the matching functions to create an ordered list of alternative workflows, resources or users.

Second, the extension of the Model component must support a variety of ontology data models, as the one written in RDF, RDF-S and OWL. It uses the OntoAPI, based on Jena Framework [HP, 2004], that offers this support and also a mechanism to cache individuals that speeds up the computation of alternative workflows, resources and users.

Briefly, the main goal of the OntoAPI is to provide a series of services for the access to the information described in the ontologies it manipulates, about both the schema and the instances.

One can list three advantages of using the OntoAPI. First, it simplifies accessing ontologies stored in a Jena repository. Second, it simplifies the development of object-oriented applications that manipulate ontologies by mapping ontology information into sets of objects and classes. Lastly, the OntoAPI provides efficient access to ontology information since it implements an optimized object cache. However, the OntoAPI does not define a complete set of operations, as does the Jena Framework. Instead, the OntoAPI provides an interface that is simple to use and that covers most of the methods commonly used to access ontology information.

6.2 Execution Nodes

The execution nodes are decomposed into an execution component and a coordination component. The *execution component* is responsible for the execution of workflow instances. It may execute some or all workflow tasks itself, or it may distribute some or all of the tasks to other users, generating a coordination hierarchy.

The execution component has an *execution service*, a *substitution service* and a *blackboard*. The *execution service* manages the blackboard, which keeps information about the workflow instances being executed. The *substitution service* invokes the flexibilization service to compute a list of possible alternatives. Then, the substitution service receives and analyzes the list, according to the current execution context (kept in the blackboard), and selects the best alternative, as discussed at the end of Section 4.

The *coordination component* is responsible for the coordinating task. This component is based on the WS-Coordination [Cabrera et al., 2003] and the WS-Transaction [Cabrera et al., 2002] frameworks, which facilitate coordinating the execution of the

registered Web Services involved in the workflow execution. Very briefly, when a Web service will participate in a workflow execution, it must register accordingly and it must choose, among the available protocols, a protocol that will control its behavior in the system. To coordinate workflow execution, the coordination component then exchanges information about the execution with the execution service.

7 Conclusion

We proposed in this paper two mechanisms to flexibilize the execution of workflow instances: a mechanism to handle presuppositions that allows workflow execution to proceed in the presence of incomplete information, and a mechanism for choosing alternative subworkflows, resources and users, in the presence of negative information or of one abstract definition. These two mechanisms use additional semantic information about the workflow definitions, resources and users involved.

We also briefly outlined an implementation for the workflow system, based on Web Services technology and pointed out how ontologies will be handled. The focus was on the matching module instance, which is the component responsible for finding alternatives for workflows, resources and users.

Finally, to validate our work, we will run a significant set of workflow definitions, with and without using the flexibilization mechanisms, and compare the distance of the results thus obtained.

Acknowledgment

This work was partially supported by CNPq, under grants 140600/01-9 and 55.2040/02-9, and also by CAPES/PROSUP. We gratefully acknowledge the fruitful discussions with the InfoPAE development team at TeCGraf/PUC-Rio.

References

- Alonso et al., 1997. Alonso, G., Agrawal, D., Abbadi, A. E., and Mohan, C. (1997). Functionality and Limitations of Current Workflow Management Systems. *IEEE Expert*, 2(5).
- Antonellis et al., 2003. Antonellis, V. D., Melchiori, M., and Plebani, P. (2003). An Approach to Web Service Compatibility in Cooperative Processes. In *Proceedings of the Symposium on Applications and the Internet Workshops*, pages 95-100. IEEE.
- Bider and Khomyakov, 2000. Bider, I. and Khomyakov, M. (2000). Is it Possible to Make Workflow Management Systems Flexible? Dynamical Systems Approach to Business Processes. In *Proceedings of the 6th International Workshop on Groupware (CRIWG 2000)*, pages 138-141.
- Cabrera et al., 2002. Cabrera, F., Copeland, G., Cox, B., Freund, T., Klein, J., Storey, T., and Thatte, S. (2002). Web Services Transaction (WS-Transaction). <http://www.ibm.com/developerworks/library/ws-transpec/>. IBM, DeveloperWorks.
- Cabrera et al., 2003. Cabrera, L. F., Copeland, G., Cox, W., Feingold, M., Freund, T., Johnson, J., Kaler, C., Klein, J., Langworthy, D., Nadalin, A., Orchard, D., Robinson, I., Shewchuk, J., and Storey, T. (2003). Web Services Coordination (WS-Coordination). <http://www.ibm.com/developerworks/library/ws-coor/>. IBM, DeveloperWorks.

- Canals et al., 1998. Canals, G., Godart, C., Charoy, F., Molli, P., and Skaf, H. (1998). COO Approach to Support Cooperation in Software Developments. In *IEEE Proceedings in Software Engineering*, volume 145, pages 79-84.
- Casanova et al., 2002. Casanova, M. A., Coelho, T. A. S., de Carvalho, M. T. M., Corseuil, E. T. L., Nobrega, H., Dias, F. M., and Levy, C. H. (2002). The Design of XPAAE - An Emergency Plan Definition Language. In *IV Simpósio Brasileiro de Geoinformática (GeoInfo' 2002)*, Ca-xambu, Minas Gerais.
- Casati et al., 1996. Casati, F., Ceri, S., Pernici, B., and Pozzi, G. (1996). Workflow Evolution. In *International Conference on Conceptual Modeling / the Entity Relationship Approach (15th ER' 96)*, pages 438-455, Cottbus, Germany. Lecture Notes in Computer Science.
- Chu et al., 1994. Chu, W. W., Chen, Q., and Merzbacher, M. (1994). *Studies in Logic and Computation: Nonstandard Queries and Nonstandard Answers*, volume 3, chapter CoBase: a Cooperative Database System, pages 41-72. Oxford University Press, New York. Edited by R. Demolombe and T. Imielinski.
- Chu and Mao, 2000. Chu, W. W. and Mao, W. (2000). CoSent: a Cooperative Sentinel for Intelligent Information Systems. Computer Science Department - University of California, LA.
- Georgakopoulos et al., 1995. Georgakopoulos, D., Hornick, M. F., and Sheth, A. P. (1995). An Overview of Workflow Management: from Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2):119-153.
- Grigori et al., 2001. Grigori, D., Charoy, F., and Gobart, C. (2001). Flexible Data Management and Execution to Support Cooperative Workflow: the COO Approach. In *Proceedings of the Third International Symposium on Cooperative Database Systems for Advanced Applications (CODAS 2001)*, pages 124-131.
- Halliday et al., 2001. Halliday, J. J., Shrivastava, S. K., and Wheeler, S. M. (2001). Flexible Workflow Management in the OPENflow System. In *Proceedings of the Fifth IEEE International Enterprise Distributed Object Computing Conference (EDOC '01)*, pages 82-92. IEEE.
- Hollingsworth, 1995. Hollingsworth, D. (1995). The Workflow Reference Model. The Workflow Management Coalition Specification TC00-1003, Workflow Management Coalition, Hampshire, UK.
- HP, 2004. HP (2004). Jena 2 - A Semantic Web Framework . <http://www.hpl.hp.com/semweb/jena.htm>.
- Joeris, 1999. Joeris, G. (1999). Defining Flexible Workflow Execution Behaviors. In *Enterprise-wide and Cross-enterprise Workflow Management - Concepts, Systems, Applications, GI Workshop Proceedings - Informatik '99*, pages 49-55. Ulmer Informatik Berichte Nr. 99-07.
- Mangan and Sadiq, 2002. Mangan, P. and Sadiq, S. (2002). On Building Workflow Models for Flexible Processes. In *ACM International Conference Proceeding Series - Proceedings of the Thirteenth Australasian Conference on Database Technologies (ADC'2002)*, volume 5, pages 103-109, Melbourne, Victoria, Australia. Australian Computer Society, Inc. Darlinghurst.
- Nutt, 1996. Nutt, G. J. (1996). The Evolution Toward Flexible Workflow Systems. In *Distributed Systems Engineering*, volume 3, pages 276-294.
- W3C, 2004. W3C (2004). OWL Web Ontology Language - Overview. W3C Recommendation. <http://www.w3.org/TR/owl-features/>.
- Weske, 1998. Weske, M. (1998). Flexible Modeling and Execution of Workflow Activities. In *Proceedings of the Thirty-First Hawaii International Conference on System Sciences*, volume 7, pages 713-722.
- Worah and Sheth, 1997. Worah, D. and Sheth, A. (1997). Transactions in Transactional Workflows. In *Advanced Transaction Models and Architectures*, pages 3-41. Kluwer Academic Publisher.