On the Design of Ontology-driven Workflow Flexibilization Mechanisms

Tatiana A. S. C. Vieira, Marco A. Casanova and Luis G. Ferrão
Department of Informatics
Pontifical Catholic University of Rio de Janeiro
Rua Marquês de São Vicente, 225
Rio de Janeiro, RJ - Brazil - CEP 22.453-900
Phone: +55-21-3114-1500 ext. 4347 / FAX +55-21-3114-1530

{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 first 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. Then, the paper presents an architecture for a workflow system, driven by ontologies that capture semantic relationships between workflows and resources. The architecture includes a component which uses matching techniques to find alternatives for workflows and resources.

Keywords: workflow, flexibilization, ontology, matching

1 Introduction

Workflow management systems have 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 purpose of standardizing workflow concepts and technologies [15]. Among other contributions, these 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 [1] [2] [12].

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 uses default values which allow execution to proceed in the presence of incomplete information; (ii) a *mechanism for choosing alternatives* to subworkflows and resources, thereby allowing workflow execution to proceed when the predefined resource is unavailable. In some sense, these mechanisms implement a component substitution strategy, applied to workflow execution, where subworkflows and resources play the role of components [3].

The proposed mechanisms use *workflow ontologies* to: (i) choose appropriate default values, when the required information is unavailable; (ii) find an alternative subworkflow or resource, when the predefined resource is unavailable, or even when the workflow definition is abstract; (iii) assess the choice of defaults and alternatives, when workflow execution terminates.

It must be noted that our flexibilization mechanisms are not intended to allow users to directly interfere with workflow execution. They rather use previous semantic information about workflows and their resources, and the current state of workflow execution, to suggest better alternatives to the user, or to make educated guesses about missing data, that allow execution to proceed when otherwise it would have been stopped. In addition, the mechanisms also support *abstract definitions*, a construct that introduces flexibility at the workflow design level.

The paper is organized as follows. Section 2 summarizes related work. Section 3 introduces a motivating example. Section 4 presents a workflow ontology, based on the example in Section 3. Section 5 describes the flexibilization mechanisms proposed in the paper. Section 6 outlines an architecture for a workflow management system that includes the flexibilization mechanisms. Section 7 briefly simulates how the flexibilization mechanisms behave. Finally, Section 8 contains the conclusions.

2 Related Work

The evolution of workflow systems towards less restrictive coordination approaches is discussed in [21]. Most of the earlier proposals for flexibilization mechanisms suggest to dynamically change the workflow structure at runtime [8].

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 [18]. The flexibilization mechanisms described in [24] [4] also offer the user the possibility to include, remove or even stop activities during workflow execution. The mechanisms proposed in our paper follow a different strategy by allowing execution to proceed in the presence of incomplete or negative information, with minimal user intervention.

The OPENflow system [14] 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 [13] account 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 [5]. The model allows activities to be anticipated, 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 [19]. This is partially done at design time, and completed at runtime, according to selection, termination and workflow definition constraints, which 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 5.

Lastly, the flexibilization mechanisms we propose bear some similarity to the query relaxation strategy adopted in the CoBase system [9] and in the CoSent system [10]. CoBase is a database system that uses the idea of cooperative queries. When queries are submitted to the system, CoBase analyzes a 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 the strategy we propose 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 4.

3 A Motivating Example

This sections introduces a motivating example, based on a real-life emergency plan, defined in a lengthy (paper) document. Albeit schematic, the example retains the essential characteristics of the original emergency plan. The original emergency plan has been translated into a workflow that runs under the InfoPAE system [7] [6]. We note, however, that the ideas presented here were not yet implemented as part of the InfoPAE project.

Consider the problem of cleaning coastal areas affected by an oil spill. To address this type of accident, the emergency plan defines a set of cleaning procedures that take into account the oil type and the characteristics of the coastal area. Table 1 provides an schematic example of cleaning procedures, when the type of coastal area is Sand Beach and the oil types are schematically named Type I through Type V. Table 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. The original emergency plan identifies a number of other types of coastal areas and describes the best cleaning procedures for each of them.

Now, suppose that an emergency team is assigned to the accident. The team will be referred to as the *user* of the emergency plan in what follows.

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 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 there is no information about the oil type, Table 1 becomes useless. In this case, the user 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).

This very simple example illustrates two general flexibilization mechanisms. First, the user implicitly resorted to presuppositions about the accidental scenario when he selected Type II as the default for oil type. Second, he used a form of component substitution when he decided to chose an alternative cleaning procedure. We generalize this behavior as explained in the next sections.

4 Workflow and Application Ontologies

4.1 Workflow Ontology

This sections informally describes a small fragment of the workflow ontology, *WkOnt*, behind the flexibilization mechanisms. The full ontology extends the OWL-S process ontology [20], which contains constructs to specify workflow composition and which includes many concepts we need to flexibly execute workflows. We refer the reader to [22] for the full details.

Basically, OWL-S include the notion of workflows and resources, and also the notion of abstract and concrete workflows, through the concepts of Simple, Atomic, and Composite Process.

Using OWL terminology [23], the basic classes and properties of *WkOnt* are defined as showed in Table 2.

Instances of the class *Abstract* are called *abstract objects*, and likewise for the other classes.

A concrete object O is an *implementation* of an abstract object A iff O and A are related by an instance of *is-implementation-of*. A concrete object O is an *alternative* for another concrete object O' iff O and O' are implementations of the same abstract object.

Abstract and concrete objects reflect a workflow design strategy that leaves part of the specification open. At design time, the user may create a workflow definition with the help of an abstract workflow A (or resource), rather than specifying a concrete workflow (or resource). The semantics of abstract objects, as well as of *can-be-replaced* and *has-default-value*, depend on the flexibilization mechanisms and will be discussed in Section 5.

The class *Weight* permits associating weights to instances of the object property *is-implemented-by*. Its role will be further discussed in the next section.

4.2 Application Ontology

Intuitively, when modelling an application area, the user should start by defining an *application ontology*, *AppOnt*, that imports *WkOnt*, our workflow ontology, and that contains a set of workflows, resources and parameters, defined as instances of *WkOnt* classes. He may then specify additional workflow definitions that reuse the initial set of objects defined in *AppOnt*. Indeed, this strategy considerably simplifies the specification of the emergency plans alluded to in Section 3. More importantly, when executing one such workflow, the flexibilization mechanisms must have access to *AppOnt*.

Figure 1 shows a simplified RDF graph that corresponds to the example in Section 3. Briefly, we have:

- *CB* is an abstract workflow that represents all cleaning procedures for Sand Beaches;

- *ND*, *UA*, *VC*, *CL*, *CH*, *HL*, *HH* and *PC* are concrete workflows that model specific cleaning procedures (see Table 1);

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

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 2: Classes and properties of $WkOnt$ ontology.

Class/Property name	**Class/property description**
Object	The root class
has-name	A functional datatype property with domain *Object* and range *String* that assigns a name to each object
Abstract and *Concrete*	Subclasses of *Object* that represent abstract and concrete objects
is-implemented-by	An object property with domain *Abstract* and range *Concrete*
is-implementation-of	The inverse of *is-implemented-by*
Cost	A datatype property with domain *Concrete* and range *Float* that indicates a cost value for the concrete objects
Workflow and *Resource*	Subclasses of *Object*, with the obvious meaning
AbstractWorkflow and *ConcreteWorkflow*	Classes defined as the intersection of *Abstract* and *Concrete* with *Workflow*, respectively
AbstractResource and *ConcreteResource*	Classes defined as the intersection of *Abstract* and *Concrete* with *Resource*, respectively
requires	An object property with domain *Workflow* and range *Resource* that indicates which resources a workflow requires
is-required-by	The inverse of *requires*
can-be-replaced	A datatype property with domain *Concrete-Workflow* and range *Boolean* that indicates whether a workflow can be replaced by an equivalent one or not
is-available	A datatype property with domain *Concrete-Resource* and range *Boolean* that indicates whether a resource is available or not
Parameter	A subclass of *Object* that represents parameters
parameter-of	An object property with domain *Parameter* and range *Workflow* that indicates which parameters a workflow depends on
has-parameter	The inverse of *parameter-of*
has-domain-value	A datatype property with domain *Parameter* and range *Literal* that enumerates the allowed values of a parameter
has-default-value	A datatype property with domain *Parameter* and range *Literal* that defines a default value for a parameter
has-current-value	A functional datatype property with domain *Parameter* and range *Literal* that indicates the current value of a parameter
Weight	A subclass of *Object* that represents weight values for the object type property *is-implemented-by*
has-source	An object property with domain *Weight* and range *Abstract* that indicates which abstract object the weight applies
has-target	An object property with domain *Weight* and range *Concrete* that indicates which concrete object the weight applies
has-weight-value	A datatype property with domain *Weight* and range *Float* that indicates weight values

- *CB* is related to *ND*, *UA*, *VC*, *CL*, *CH*, *HL*, *HH* and *PC* by instances of the *is-implementation-of* property;
- *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;
- *Ab* is related to Ab_1 and Ab_5, *Rk* is related to Rk_5 and Mc_1, and *Mc* is related to Mc_4 by instances of the *is-implementation-of* property;
- *Oil_Type* is a parameter;
- *Oil_Type* is related to *Type I*, *Type II*, *Type III*, *Type IV* and *Type V* by instances of the *has-domain-value* property;
- *has-oil-type*, a new datatype property with domain *Weight* and range *Literal*, that indicates to which oil types the weight applies.

The application ontology may also contain an additional set of rules (see [22] for the details), classified as:

presupposition rules : select the appropriate default values;

consistency rules : assess the flexibilization decisions made, when workflow execution terminates;

semantic proximity rules : define instances of the datatype property *has-weight-value*, perhaps depending on parameter values.

For example, the semantic proximity rules may capture the information about weights shown in Table 1. The first entry of the table would correspond to the following rule, written in SWRL human readable syntax [16]:

has-oil-type(?x,'Type I') & has-source(?x,'CB') & has-target(?x,'ND') ⇒ has-weight-value(?x,0.0)

Note that, in this specific application ontology, the type of coastal area is implicitly given when the user defines the abstract workflows. For example, *CB* applies only to sand beaches. However, the dependency of the weights on the oil type had to be modelled by an additional datatype property, *has-oil-type*, of the class *Weight*. Then, the semantic proximity rules may be used to implicitly generate instances of *has-weight-value*.

This circumvented way of capturing the information in Table 1 just reflects the limitations of the ontology language chosen, viz. OWL, which supports only binary relationships. On the other hand, with the help of rules, we may define weight values that depend on complex configurations.

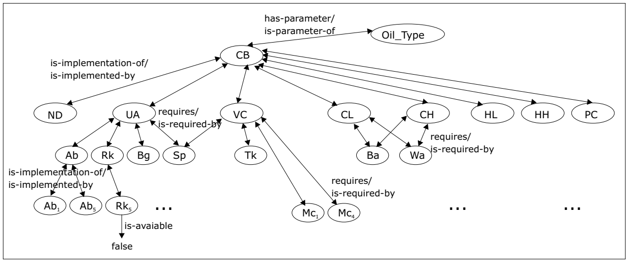

Figure 1: Fragment of a workflow ontology that shows the beach cleaning procedures.

5 Flexibilization 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 or when abstract definitions are used, by suggesting alternatives for workflows or resources.

In what follows, assume that *AppOnt* is the underlying application ontology.

Let W be a workflow definition and E be an execution engine that runs a workflow instance I_W of W. Let V be a subworkflow of W and assume that V has a pre-condition $C[p]$, where p is a parameter, and that V requires a resource r. Therefore, V cannot be executed when $C[p]$ is false or the value of p is undefined, and when resource r is unavailable.

The engine E may invoke the mechanism to handle presuppositions in two distinct points of the execution. First, when delegating a subworkflow V to another execution engine F, the engine E may find out that the value of parameter p, required to test $C[p]$, is undefined. In this case, E takes the following actions: (1) E executes the presupposition mechanism to select a default value d_p for p; (2) if $C[p/d_p]$ is true (the precondition C when p has value d_p), then E delegates V to F.

Second, when the engine E executes I_W, it may also miss the value of some parameter p. Then, E may again invoke the presupposition mechanism to select a default value for p.

The mechanism to handle presuppositions has a reasoner component that uses *presupposition rules* and instances of *has-default-value*, both defined in *AppOnto*, to select a default value for p. The mechanism will fail, if no such instance is found. Intuitively, this indicates that the user, when he defined *AppOnt*, decided that p should not be flexibilized in the current context. Note that the use of rules is necessary to select default values based on context information. Therefore, the process of selecting a default value is a deductive process, and not merely a query over instances of *has-default-value*.

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 rules.

To analise the first mode, suppose that E is about to delegate a subworkflow V to another execution engine F. Assume that $C[p]$ is true for the current value of p, but resource r is unavailable. The engine E may invoke the mechanism for choosing alternatives, which in turn uses the semantic proximity rules to try to find: (i) a resource r' such that r' is semantically equivalent to r and r' is available; or (ii) a subworkflow V' such that V' is semantically equivalent to V and V' can be executed, i.e., all pre-conditions of V' are true and all resources that V' requires are available. It is worth noting that these two choices are not independent, in the sense that (ii) may trigger (i) when V' requires resources that are unavailable.

To analise the second mode, assume that W has an abstract subworkflow A. Suppose that E is about to create a subworkflow instance of A. Then, E has to invoke the mechanism for choosing alternatives to select a concrete workflow B that is related to A by an instance of the *is-implementation-of* property, contained in *AppOnt*.

In both operation modes, typically more than one alternative is possible. Hence, the mechanism for choosing alternatives has a matching component that uses weight information, included in *AppOnto* (see Table 2), to select the best alternative, or at least an heuristically reasonable alternative. Therefore, the process of choosing alternatives is also a deductive process, and not merely a query over instances of classes of *AppOnto*.

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

When a subworkflow instance terminates, the execution engine must invoke the consistency rules to analyze possible conflicts caused by the use of a default value or by the choice of an alternative workflow or resource. If conflicts indeed occurred, the execution engine may invoke compensating actions, among those registered in the ontology, to try to undo the effects of the faulty workflow execution. Finally, we note that this is just one of the exceptions that the execution engine is prepared to handle, as discussed in the next section.

6 Implementation

This section outlines the architecture of the workflow management system. Briefly, the system consists of (see Figure 2): (i) an ontology manager, which offers services to store, manipulate and query the workflow ontology; (ii) a collection of execution engines, responsible for running workflow instances; and (iii) an instance manager, which keeps track of all workflow instances in the system. The discussion that follows is independent of the protocol used to invoke the services these components provide; in particular, they could be accessed as Web services.

A complete description of the architecture in ACME [11] can be found in [22].

6.1 Ontology Manager

The *ontology manager* 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, covering

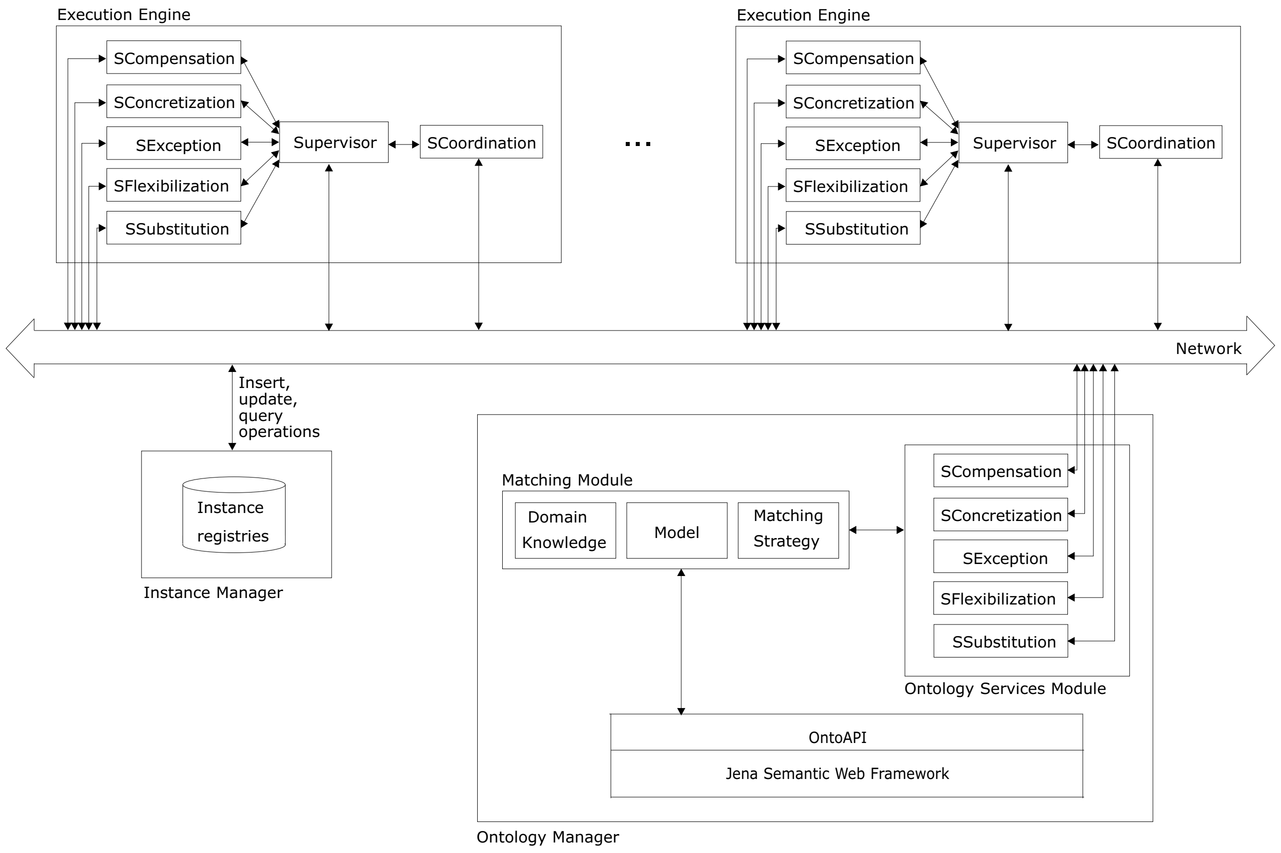

Figure 2: The ontology-driven architecture proposed for flexible workflow execution.

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 and pairs of resources. The matching module uses the matching functions to create an ordered list of alternative workflows or resources.

Second, the extension of the Model component must support a variety of ontology data models, including RDF, RDF-S and OWL. It uses the OntoAPI, based on Jena Framework [17], that offers such support and a mechanism to cache individuals, which speeds up the computation of alternative workflows and resources.

Briefly, the OntoAPI provides a series of services that facilitate access to ontology elements (classes, properties and instances). It simplifies accessing ontologies stored in a Jena repository, facilitates the development of object-oriented applications that manipulate ontologies by mapping ontology information into sets of objects and classes, and 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 covers most of the methods commonly used to access ontology information.

The *ontology services module* uses the matching module to find alternative workflows and resources, and to compute default values, using the workflow and the application ontologies, as previously discussed.

The ontology services module offers five classes of services: *SCompensation*, *SConcretization*, *SException*, *SFlexibilization*, and *SSubstitution*. These services represent the implementation core of the flexibilization mechanisms proposed in this paper. They are accessible only through the corresponding services in the execution engines, which are in turn invoked by their supervisor. The execution engines assume the role of client and the ontology manager assumes the role of server in this interaction.

6.2 Execution Engine

The *execution engine* is responsible for running workflow instances. The workflow management system may have any number of execution engines involved with the execution of a single workflow, organized as follows.

Let W be a workflow definition and V be a subworkflow of W. Let E be an execution engine controlling an instance I_W of W. If W allows, E may distribute V to another execution engine F, which then creates an instance I_V of V. In this case, we say that E *delegated* V to F, that E *coordinates* F and that F *is directly subordinated to* E. We also say that I_V *is directly subordinated to* I_W. Furthermore, we define the *is subordinated to* relationship between execution engines by taking the transitive closure of the *is directly subordinated to* relationship, and likewise for workflow instances.

The *Supervisor* is the core component of the execution engine. It controls the execution of one or more workflow instances, and may delegate a subworkflow to another execution machine. When the subordinated instance terminates, it is also responsible for sending the result to the coordinator machine.

The *Supervisor* may invoke six different services:

***SCoordination*,** invoked when the supervisor requires communication with other execution engines. The service is based on message passing.

***SConcretization*,** invoked when the supervisor reaches an abstract subworkflow definition A. The service then tries to find an appropriate concretization for A, among those registered in the ontology.

***SFlexibilization*,** invoked when the supervisor detects a timeout caused by an undefined parameter. The service then tries to find an appropriate default value, using the ontology.

***SSubstitution*,** invoked when the supervisor detects a timeout caused by an unavailable resource. The service first tries to find an alternative resource, among those registered in the ontology; if it fails, it then tries to find an alternative workflow, again among those registered in the ontology.

***SCompensation*,** invoked when the supervisor detects an execution conflict, possibly caused by the use of a default value or the choice of an alternative workflow or resource. The supervisor passes the faulty workflow and its execution engine to the service, which then tries to find an appropriate compensating action, among those registered in the ontology.

***SException*,** invoked when the supervisor detects an exception raised during workflow execution. The service then tries to find an appropriate treatment for the exception raised, among those registered in the ontology.

Note that, except for the first, all these services require access to the ontology. Therefore, they interact with the corresponding services implemented by the ontology manager. The supervisor directly communicates only with the instance manager.

6.3 Instance Manager

The *instance manager* keeps track of the workflow instances running in the system. Each time the supervisor of an execution engine creates a workflow instance I, it sends a message to the instance manager containing the identification of the related process, the identification and the start time of I, and the identification of its execution engine. The end time and the final state of I are marked "unknown", indicating that I has not yet finished. When I finishes and the supervisor detected no conflict, it send a message to its coordinator signalling that I terminated successfully. The coordinator then updates I's entry at the instance manager with the end time and final state.

When an execution engine is running a workflow instance and needs a parameter value that comes from another subworkflow, the supervisor sends a request for the instance manager asking for the last instance of the required process whose execution ended. Using the identification of the execution engine and of the instance, the supervisor obtains the necessary parameter value. If the value is not yet available and a timeout occurs, the supervisor must invoke the *SFlexibilization* service to obtain a default value for the parameter, as explained before.

Note that the instance manager module communicates only with execution engines to maintain information about workflow instances.

7 Simulating the Execution of a Workflow Instance

This section simulates a partial execution of a sample workflow, indicating how the flexibilization mechanisms use the workflow ontology.

Consider a very simple workflow W capturing an emergency plan designed to clean coastal areas affected by an oil spill. Suppose that W is composed of 4 subworkflows: W_1 determines the type of oil spilled, setting the value of a parameter *OilType*; W_2 determines the type of coastal area affected, setting the value of another parameter *CoastalAreaType*; and W_3 and W_4 define cleaning procedures for two different combinations of oil type and coastal area type. Assume that they may all run in parallel, but W_3 and W_4 have pre-conditions that depend on the values of *OilType* and *CoastalAreaType*. Furthermore, assume that W_3 contains an abstract workflow, corresponding, say, to node CB of the ontology shown in Figure 1.

Suppose that an execution engine E is running an instance I_W of W.

As a first example, suppose that E is running W_3 and that E reaches CB. Then, E invokes the *SConcretization* service, which in turn sends a message to the corresponding service of the ontology services module. At the ontology services module, the *SConcretization* service calls the matching module to find the best match between the abstract workflow CB and all concrete workflows related to it by the property *is-implemented-by*.

Suppose that the best matches are $L = (VC, CL)$. This list is returned to the *SConcretization* service of the execution engine E, which then verifies the workflows in L that can indeed be run, i.e., whose preconditions are all true and whose resources are all available, as illustrated in Figure 3.

Suppose that VC is selected. Then, E continues the execution of W by running VC.

As a second example, suppose that E stops executing because W_1 is not responding with the appropriate oil type. Assume that a timeout associated with the pre-conditions of W_3 or W_4 (or both) occurs. The supervisor of the execution engine E then calls the *SFlexibilization* service, which sends a message to the corresponding service of ontology services module. At the ontology services module, the *SFlexibilization* service selects a default value d_o for the missing oil type, using the workflow ontology. The default value d_o is returned to the *SFlexibilization* service of the execution engine E. The supervisor of E then tests the pre-conditions of W_3 and W_4 with this default value.

Suppose that the pre-condition of W_4 is true. Then, E continues the execution of W by running W_4.

When I_W terminates, the supervisor runs consistency rules to verify if the presupposed value, d_o, was confirmed. If a conflict is detected, the *SCompensation* service is invoked and a compensation workflow for W_4 is selected from the workflow ontology, in a process very similar to those already described.

8 Conclusion

We described 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 or resources, in the presence of negative information or when an abstract definition is reached. These two mechanisms use additional semantic information about the workflow definitions and resources involved.

We also outlined an implementation architecture for the workflow management system, pointing out how ontologies are handled. The focus was on the matching module, which is the component responsible for finding alternatives for workflows and resources. The matching module and the OntoAPI are fully operational and a complete implementation of the workflow management module is planned for the end of 2005.

The ontology approach played a central role in our overall strategy in two interrelated aspects. First, it facilitated modeling the application as a collection of workflows and their resources - the application ontology - much in the same way experts define complex tasks in real life - as a structured collection of separate, smaller procedures. Second, it permitted construct-

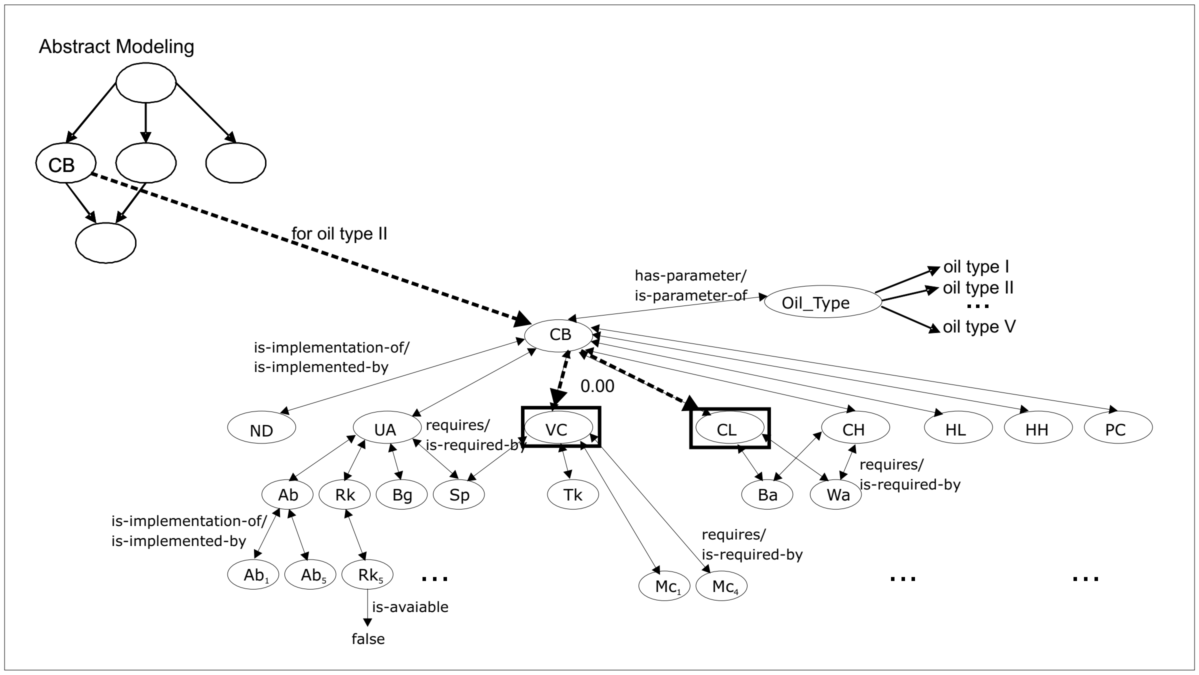

Figure 3: The process realized by the concretization service over the workflow ontology.

ing the workflow management system as a combination of a standard workflow execution engine, a deductive component (the mechanism to handle presuppositions) and a matching component (the mechanism for choosing alternatives), that use the application ontology again much in the same way final users combine the procedures, that experts defined, to achieve their goals, according to the current situation.

To achieve flexibility, we had to pay the price of the extra overhead of the deductive and the matching components. However, for the applications we had in mind, such as disaster response, the added flexibility and the gains in design simplicity far compensate the additional runtime overhead.

Finally, we refer the reader to [22] for a detailed description of the concepts informally introduced here, re-formulated as OWL-S extensions.

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

[1] Gustavo Alonso, Divyakant Agrawal, Amr El Abbadi, and C. Mohan. Functionality and Limitations of Current Workflow Management Systems. *IEEE Expert*, 2(5), 1997.

[2] Gustavo Alonso and Hans-Joerg Schek. Research Issues in Large Workflow Management Systems. Technical Report 1996PA-as96-nsfws, Institute for Information Systems, Switzerland, April 1996.

[3] Valeria De Antonellis, Michele Melchiori, and Pierluigi Plebani. An Approach to Web Service Compatibility in Cooperative Processes. In *2003 Symposium on Applications and the Internet Workshops (SAINT'03 Workshops)*, pages 95-100, Orlando, Florida, January 2003. IEEE.

[4] Ilia Bider and Maxim Khomyakov. 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, Madiera, Portugal, October 2000.

[5] G. Canals, C. Godart, F. Charoy, P. Molli, and H. Skaf. COO Approach to Support Cooperation in Software Developments. In *IEEE Proceedings in Software Engineering*, volume 145, pages 79-84, April/June 1998.

[6] Marco A. Casanova, Tatiana A. S. Coelho, Marcelo Tílio M. de Carvalho, Eduardo T. L. Corseuil, Hérica

Nobrega, Fábio M. Dias, and Carlos H. Levy. The Design of XPAE - An Emergency Plan Definition Language. In *IV Simpósio Brasileiro de Geoinformática (GeoInfo' 2002)*, Caxambu, Minas Gerais, Dezembro 2002.

[7] Marco A. Casanova, Marcelo Tílio M. de Carvalho, and Juliana Freire. The Architecture of an Emergency Plan Deployment System. In *III Simpósio Brasileiro de Geoinformática (GeoInfo' 2001)*, Rio de Janeiro, RJ, 2001.

[8] Fabio Casati, Stefano Ceri, Barbara Pernici, and Giuseppe Pozzi. Workflow Evolution. In Bernhard Thalheim, editor, *International Conference on Conceptual Modeling / the Entity Relationship Approach (15th ER' 96)*, pages 438-455, Cottbus, Germany, October 1996. Lecture Notes in Computer Science.

[9] Wesley W. Chu, Q. Chen, and M. Merzbacher. *Studies in Logic and Computation 3: Nonstandard Queries and Nonstandard Answers*, volume 3, chapter CoBase: a Cooperative Database System, pages 41-72. Oxford University Press, New York, 1994. Edited by R. Demolombe and T. Imielinski.

[10] Wesley W. Chu and Wenlei Mao. CoSent: a Cooperative Sentinel for Intelligent Information Systems, March 2000. Computer Science Department - University of California, LA.

[11] David Garlan, Robert T. Monroe, and David Wile. ACME: An Architecture Description Interchange Language. In *Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON'97)*, pages 169–183, Toronto, Ontario, November 1997. IBM Press.

[12] Dimitrios Georgakopoulos, Mark F. Hornick, and Amit P. Sheth. An Overview of Workflow Management: from Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2):119-153, April 1995.

[13] Daniela Grigori, Franois Charoy, and Claude Gobart. 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, April 2001.

[14] J. J. Halliday, S. K. Shrivastava, and S. M. Wheater. 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, September 2001.

[15] David Hollingsworth. The Workflow Reference Model. The Workflow Management Coalition Specification TC00-1003, Workflow Management Coalition, Hampshire, UK, January 1995.

[16] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission, May 2004. http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/Overview.html.

[17] HP. Jena 2 - A Semantic Web Framework . http://www.hpl.hp.com/semweb/jena.htm, 2004.

[18] G. Joeris. Defining Flexible Workflow Execution Behaviors. In *Enterprise-wide and Cross-enterprise Workflow Management - Concepts, Systems, Applications, GI Workshop Proceedings - Informatik '99*, pages 49-55, 1999. Ulmer Informatik Berichte Nr. 99-07.

[19] Peter Mangan and Shazia Sadiq. 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, Australia, January/February 2002. Australian Computer Society, Inc. Darlinghurst.

[20] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. OWL-S: Semantic Markup for Web Services. W3C Member Submission, November 2004. http://www.w3.org/Submission/2004/SUBM-OWL-S20041122/Overview.html.

[21] Gary J. Nutt. The Evolution Toward Flexible Workflow Systems. In *Distributed Systems Engineering*, volume 3, pages 276-294, December 1996.

[22] Tatiana A. S. C. Vieira. *Mecanismos Baseados em Ontologias para a Execuo Flexvel de Workflows*. PhD thesis, Department of Informatics - Pontifical Catholic University of Rio de Janeiro, Brazil, Rio de Janeiro, RJ - Brazil, 2005. In Portuguese. To be presetend in june.

[23] W3C. OWL Web Ontology Language - Overview. W3C Recommendation, February 2004. http://www.w3.org/TR/owl-features/.

[24] Mathias Weske. Flexible Modeling and Execution of Workflow Activities. In *Proceedings of the Thirty-First Hawaii International Conference on System Sciences*, volume 7, pages 713-722, January 1998.