A FRAMEWORK FOR THE MANAGEMENT OF CONTEXT-AWARE WORKFLOW SYSTEMS

Liliana Ardissono, Roberto Furnari, Anna Goy, Giovanna Petrone and Marino Segnan
Dipartimento di Informatica - Università di Torino, Corso Svizzera 185, 10149 Torino, Italy

Keywords: Web Services and Web Engineering, Personalized Web Sites and Services.

Abstract: This paper presents the CAWE framework for the context-aware management of applications based on the composition of Web Services in complex workflows. We introduce a representation of context-dependent activities based on an abstraction hierarchy which supports the specification of synthetic and clear workflows. Moreover, we propose a framework architecture which enriches the capabilities of a workflow engine in order to support the execution of possibly complex adaptation rules. We have exploited the CAWE framework to develop a prototype application handling a medical guideline which specifies the activities to be performed in order to monitor patients treated with blood thinners from their home. The application coordinates actors playing different roles (e.g., patient, doctor, etc.) and can be accessed by using diverse client devices.

1 INTRODUCTION

Up to now, the introduction of context awareness in workflow systems has been mainly focused on Quality of Service management and on the adaptation to the user's device; e.g., see (Benlismane et al., 2005) and (Keidl and Kemper, 2004), respectively. In contrast, little effort has been devoted to support context-awareness in composed applications whose business logic imposes the management of complex and long-term interactions with the service suppliers. Web Services having complex interaction protocols have been integrated in standard workflows by utilizing Web Service composition languages, such as WS-BPEL (OASIS, 2005). However, the context-aware activity executions have been specified by directly modeling all the decision points and the alternative execution paths in flat workflows, which turn out to be hardly readable for the workflow designer and are little extensible (e.g., to handle a new contextual condition and the associated course of action).

In order to enhance the flexibility of workflow and Web Service composition systems, context information and context-adaptation rules should be explicitly represented in the application logic. As a first step in this direction, we propose the CAWE (Context Aware Workflow Execution) framework for the management of context-aware applications whose business logic is implemented as a context-sensitive workflow. CAWE is based on the following features:

- A hierarchical representation of the workflow, which can include *abstract activities* hiding the context-dependent details.
- The association of each abstract activity with its implementations. These are a set of alternative, context-dependent parts of the workflow which can be performed in different context states.
- The declarative specification of the context conditions determining the selection of the appropriate context-dependent part of the workflow at execution time.

These features support the execution of alternative courses of action and the context-aware invocation of Web Service suppliers.

In order to test the functionality offered by our framework, we have developed a proof-of-concept prototype and we have instantiated it on the execution of a medical guideline for the home assistance of patients affected by heart diseases and treated with blood thinners. The application, which coordinates the activities to be performed by the patient, by the

Ardissono L., Furnari R., Goy A., Petrone G. and Segnan M. (2007).
A FRAMEWORK FOR THE MANAGEMENT OF CONTEXT-AWARE WORKFLOW SYSTEMS.
In *Proceedings of the Third International Conference on Web Information Systems and Technologies - Internet Technology*, pages 80-87
DOI: 10.5220/0001279800800087
Copyright © SciTePress

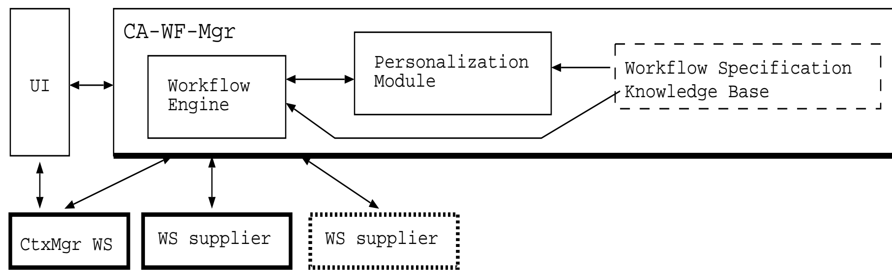

Figure 1: Architecture of the CAWE framework. Web Service interfaces are depicted as thick lines.

doctor and by other personnel, may be accessed from the internet, by using a PC or a Smart Phone client.

In the rest of this paper, we describe the architecture of the CAWE framework and the proposed hierarchical representation of context-sensitive workflows. Moreover, we describe in which way a standard workflow engine can be exploited to execute this kind of workflow.

2 THE CAWE FRAMEWORK

The CAWE framework supports the development of context-aware applications adapting the user interface and the workflow execution to the context. In this paper, we focus on the second aspect, which is based on the introduction of a Personalization Module selecting, at each stage of execution, the most appropriate activity path to be performed by the workflow engine.

2.1 Architecture

Figure 1 shows the framework architecture.

A *Context Manager service (CtxMgr WS)* provides the contextual information during the execution of the application. In line with current approaches to context-aware workflow management, we have designed the Context Manager as a Web Service which might exploit multiple information sources to synthesize the required information.

The *Context-Aware Workflow Manager (CA-WF-Mgr)* runs a workflow engine on the process specification which defines the business logic of the composed application. The engine executes an abstract workflow (defined later on) as if it were a standard workflow. However, each time it encounters an abstract activity, the engine invokes the Personalization Module to retrieve the appropriate context-dependent part of the workflow and executes it. The Personalization Module wraps the adaptation logic, which is represented by the applicability conditions of the context-sensitive parts of the workflow. The CA-WF-Mgr invokes the CtxMgr WS for retrieving the contextual information.

A device-dependent *User Interface* (UI) enables the user to interact with the application while carrying out the tasks assigned to her/him. The UI component retrieves the context information needed to adapt the UI pages from the CtxMgr WS.

The CA-WF-Mgr and the CtxMgr WS may invoke some *Web Service suppliers* to receive external services. E.g., in our sample scenario we consider a Nurcery Service, a Clinical Record Manager WS storing the clinical records of the patients, and the WS interface of the lab which performs the medical tests.

2.2 Representation of the Context-Sensitive Workflow

In order to represent the context-dependent parts of the workflow in a declarative and synthetic way, we introduce an abstraction hierarchy whose higher-level elements describe the activities to be performed in generic way. Specifically:

- We introduce the concept of *abstract activity* to denote an activity schema which does not directly specify a piece of business logic of the application (e.g., starting a task to be performed by a human actor, sending or waiting to receive a message from a Web Service, or carrying out some internal computation). In fact, the actions to be executed in order to complete the activity are selected at runtime, depending on the context state. An abstract activity is characterized by the following features:
  - *Abstract activity name*.
  - *Input and Output arguments*. The input arguments must have been set before invoking the abstract activity. The output arguments are set as a result of the completion of the activity, i.e.,

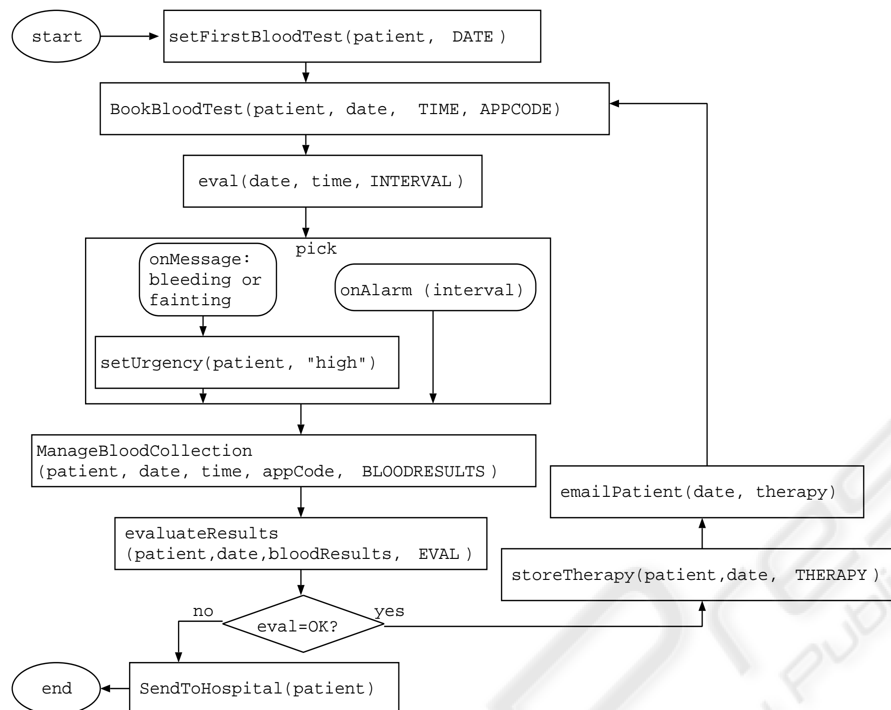

Figure 2: Abstract workflow of our medical application.

after one of its implementations has successfully terminated.

- *Input variables of the applicability conditions*. This is a list of variables occurring in the applicability conditions associated to the implementations of the abstract activity. The variables are a subset of the input arguments of the activity.

- We define *abstract workflow* a workflow schema including at least one abstract activity: the workflow abstracts from the details of execution of at least one (context-dependent) activity. If a workflow does not contain any abstract activity, it represents a *concrete workflow* and it can be executed by the workflow engine in straightforward way.[1]

Each abstract activity has an associated a set of *context-dependent implementations* representing the alternative courses of action which the workflow engine should actuate, depending on the context. Each *context-dependent implementation* is characterized by the following features:

- *Identifier of the implementation*.
- *Implements* field. This feature specifies the name of the implemented abstract activity.
- *Workflow specification*. Each context-dependent implementation consists of a workflow and represents a subprocess to be completed in order to achieve the results of the abstract activity.[2] The workflow may be elementary, i.e., composed of a single activity.
- *Input and output arguments*. The input and output arguments are set according to the values of the corresponding input/output arguments of the abstract activity; the correspondence is by name. If the workflow is elementary, the input and output arguments are the input and output arguments of its included activity.
- *Context-dependent variables*. This is a list of process variables whose value has to be set according to the context, when the implementation is selected for execution. We introduced this feature in order to support a reactive adaptation to context parameters whose value can change abruptly during the execution of the application. In fact, although such parameters might have already been evaluated during the execution of the

[1]In some work about workflow systems, the terms abstract and concrete workflow denote, respectively, the schema of a workflow and its instances. Notice that in this paper we adopt a different meaning.

[2]The implementation of an abstract activity might itself be an abstract workflow.

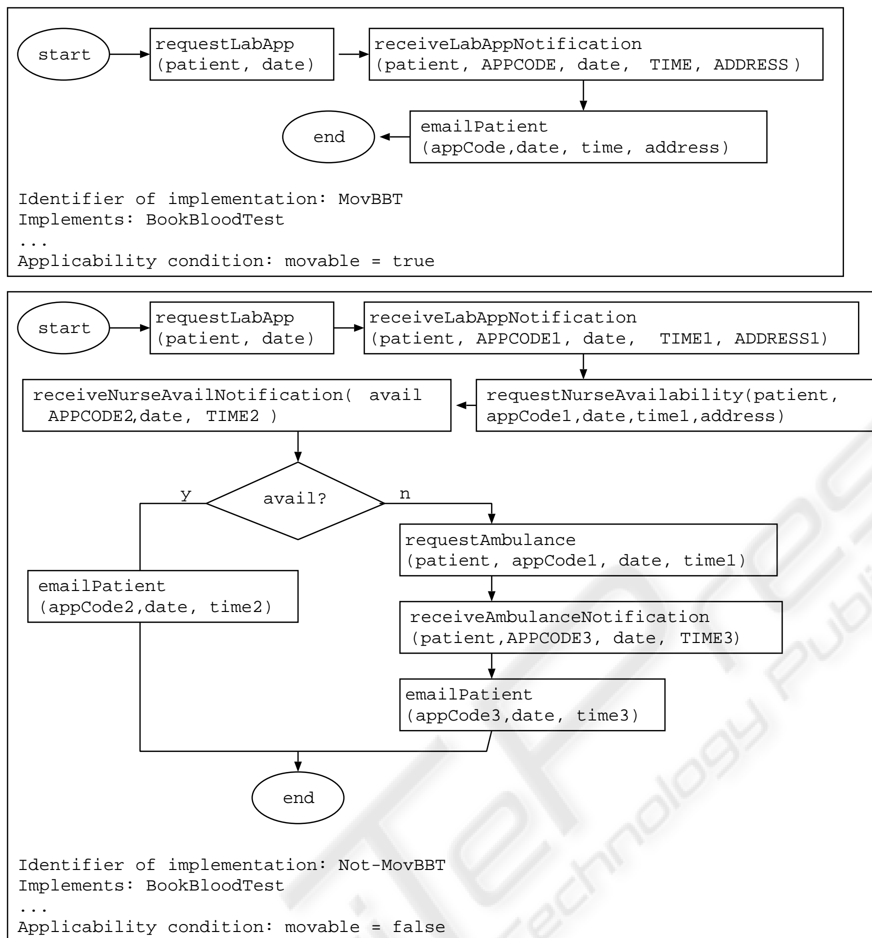

Figure 3: Context-dependent implementations of abstract activity `BookBloodTest.` (for movable and not movable patients).

application, they need to be refreshed in order to suitably influence the system behavior.

- *Applicability condition*. A boolean condition which is true if the context-dependent implementation can be applied in the current context, false otherwise. The applicability conditions are characterized by *input variables* (bound to the values of the corresponding input arguments of the abstract activity) and *context-dependent variables* (bound according to the context).

Figure 2 depicts the abstract workflow of our prototype application. Abstract activity names start with uppercase letter and concrete activity names start with lowercase letters. Moreover, the output arguments of the activities are uppercased. We describe only the first part of the abstract workflow for space reasons:

1. A doctor starts the workflow by setting the first blood test to be performed (`setFirstBloodTest(patient, DATE)`).
2. The application reserves a blood test with a lab at the specified date and evaluates the time interval before the test (`BookBloodTest(patient, date, TIME, APPCODE)`, `eval(date, time, INTERVAL)`).
3. If the patient's health state is good, she waits until the date of the test (`onAlarm(interval)`). However, if any warning symptoms occur before that date (`onMessage ...`), the service sets the urgency of the case (`setUrgency(patient, ``high'')`). The `pick` scope includes the competing courses of action.
4. At the specified date, or after a warning symptom, a blood sample is taken from the patient (`ManageBloodCollection(patient, date, time, appCode, BLOODRESULTS)`). The lab analyzes the sample and returns the results, which are evaluated by a doctor. If the results are good, the next blood test and the therapy

are set; then, the application notifies the patient (`emailPatient(date, therapy)`) and the flow restarts from item 2. Otherwise, the patient is advised to go to the hospital.

Figure 3 depicts the context-dependent implementations associated to the `BookBloodTest` abstract activity. Both implementations are concrete workflows:

- The first one may be performed if the patient can be transported by car (`movable(patient)` applicability condition). The workflow specifies that the application must obtain an appointment with the lab (`requestLabApp(patient, date)`, `receiveLabAppNotification(patient, APPCODE, date, TIME, ADDRESS)`) and notify the patient (`emailPatient(appCode, date, time, address)`).
- The second one is suitable to handle patients which cannot be transported by car (`not movable(patient)`) and specifies the activities to be performed in order to request a nurse who collects the patient's blood at her home, or an ambulance to take the patient to the lab.

It should be noticed that our abstraction mechanism enables the context-dependent invocation of alternative Web Service suppliers offering similar services, and the management of different invocations on the same Web Service. Specifically:

- An abstract activity A(arg_1, ..., arg_n) might be implemented as alternative elementary subprocesses, each one including one of the following concrete activities: $C_1(arg_1, ..., arg_x)$, ... $C_m(arg_1, ..., arg_y)$. Suppose that the activities invoke different Web Service suppliers, e.g., offering the same service at different QoS levels. Then, the application can select the one to be invoked on the basis of the applicability conditions associated to the alternative implementations.
- The abstract activity A(arg_1, ..., arg_n) might be implemented as a single concrete activity *C(arg_1, ..., arg_z)*, which can be invoked by setting its context-dependent variables according to the context. For instance, the same supplier might offer a service at different QoS levels and the application might choose the most appropriate one by setting the arguments of the invocation.[3]

It should be noticed that, although the context variables are particularly significant in the last case above, they may occur in any implementation of an abstract activity. In our sample scenario, the context-aware invocation of a Web Service supplier is exemplified by the `ManageBloodCollection` abstract activity. In fact, its implementations include a `notifyUrgency(patient, date, urgency)` concrete activity, which the application executes in order to inform the lab about the degree of urgency of the patient (if the urgency is high, the results should be produced in few hours). The `urgency` variable is stored in the patient's context and can be retrieved by interacting with the CtxMgr WS.

[3] In this case, the applicability condition of the concrete activity is always true, because it is not necessary to discriminate between a set of alternatives.

2.3 Context-aware Workflow Execution

The workflow engine wrapped by the Context-Aware Workflow Manager creates an instance of the abstract workflow each time a user performs the starting activity of the workflow (in our application, when the doctor sets the first blood test for a given patient). The abstract workflow is executed as if it were a standard one; however, when the engine encounters an abstract activity, it works as follows:

1. First, it invokes the Personalization Module on the abstract activity.
2. When the Personalization Module returns the appropriate context-dependent implementation, the workflow engine performs it as a subprocess of the main process instance.
3. At subprocess completion, the control returns to the abstract workflow, in order to perform the next (concrete or abstract) activity.

The Personalization Module, invoked on an abstract activity, works as follows:

1. First, it retrieves the context-dependent implementation(s) of the abstract activity from the Workflow Specification Knowledge Base. If the abstract activity is associated to more than one implementation, the module evaluates the applicability conditions of the alternatives and selects one of the applicable implementations. The module retrieves from the CtxMgr WS the values of the context-dependent variables of the applicability conditions.
2. The module binds the context-dependent variables of the selected implementation to their current values and returns the implementation together with the bindings.

In the following section, we explain how the standard behavior of a workflow engine can be extended to perform the steps described above. Before entering such technical details, we want to point out that this extension does not require any changes to the workflow engine: in fact, it can be implemented by embedding

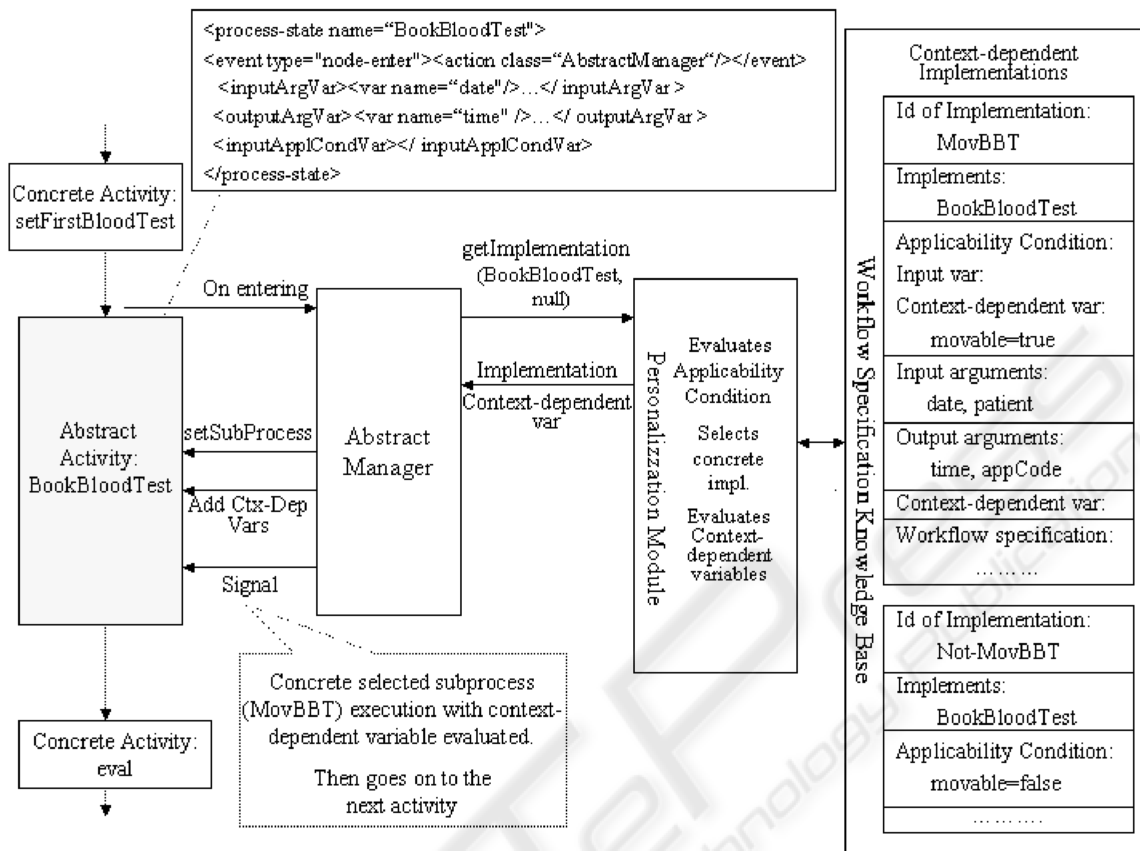

Figure 4: Management of abstract activity.

some executable code in the definition of the abstract activities.[4] In this way, the workflow engine can perform each abstract activity as if it were a concrete one; however, the execution results in the invocation of the Personalization Module and the enactment of the appropriate implementation.

2.4 Details About the Cawe Prototype

We have developed an initial prototype of the CAWE framework on top of jBPM (Koenig, 2004), a business process management system based on Petri Net model (Jensen, 1976) implemented in Java. jBPM is based on Graph Oriented Programming model, which complements object-oriented programming with a runtime model for executing long lasting workflows activities, represented as graphs.

In the jBPM language, standard workflow activities are represented as `Actions`, i.e., they are pieces of Java code that implement the business logic and are executed upon events in the process. The direct association of `Actions` to pieces of executable code makes the invocation of special purpose operations rather straightforward. jBPM also separates the navigation of the underlying graph (workflow) from the maintenance of the workflow state, which is stored in an object token, similar to the Petri Net model.[5] Furthermore, jBPM offers `process-state nodes` to represent the subprocesses to be performed.

In the CAWE prototype, we have implemented the abstract activity as a process-state node representing the subprocess which has to be performed. It should be noticed that, in a standard jBPM workflow, a `process-state node` stores the name of the workflow definition to be executed when the token (workflow state) reaches the node. However, in our framework, the name of such a workflow is known only after the invocation of the Personalization Module. Therefore, we manage the node as follows (see Figure 4, which shows the management of an abstract activity on `BookBloodTest`):

[4] In turn, this can be achieved at workflow design time, by translating the abstract workflow to a very similar one, where the abstract activities have been modified.

[5] In our framework, we assume that the workflow is represented as a Petri Net.

- The node is set to the same name as the abstract activity; e.g., `BookBloodTest`.
- We use the `Action` (delegation) tool provided by jBPM and we define a java class handler, the `AbstractManager`, that the jBPM engine invokes as soon as the token of execution enters the node (`event:enter-node`). The `AbstractManager` invokes the Personalization Module to get the context-dependent process implementation that has to be performed. Moreover, the handler sets the subprocess definition obtained from the Personalization Module into the process-state node associated to the abstract activity before starting its execution.

The Personalization Module invokes the CtxMgr WS in order to evaluate the context-dependent variables used by the subprocess implementation and passes the context-dependent variables input list to the `AbstractManager`. In turn, the handler sets these variables in the new subprocess instance in order to make them available at execution time.

In the following we provide some details about the execution of the `AbstractManager`.

- At invocation time, the handler gets the name of the abstract activity and retrieves from the main process instance the values of the variables involved in the evaluation of the applicability condition.
- Then, it invokes the Personalization Module passing the name of the abstract activity and the input variables list of the applicability condition (`getImplementation` arrow in the figure).
- The `AbstractManager` receives the following data items from the Personalization Module:
  - The context-dependent implementation to be performed; e.g., `MovBBT`.
  - A list of evaluated input context-dependent variables. Notice that the list of input arguments used in the selected context-dependent implementation is by definition a subset of the input arguments declared at the abstract activity level and is made available to the implementation process by the workflow engine.
- The `AbstractManager` sets the implementation definition as the subprocess definition for this process-node (`setSubProcess` arrow in the figure).
- Moreover, the `AbstractManager` starts the execution of the subprocess (`Signal` arrow).

Similar to the input arguments, output arguments are copied to the parent process by the workflow engine.

3 RELATED WORK

The introduction of hierarchical workflows is not new; for instance, several workflow languages enable the designer to define complex activities which expand in workflows forming a composition hierarchy; e.g., see WS-BPEL (OASIS, 2005) and process languages such as Petri Nets (Jensen, 1976; van der Aalst, 2002). Our proposal differs from such approaches because it introduces a specialization hierarchy supporting the actuation of the same abstract activity in different ways. Indeed, our framework brings to the research in workflow systems some concepts that have been traditionally developed in the research about autonomous agents and reactive planners; e.g., see (Firby, 1987; Georgeff and Lansky, 1987).

In the Semantic Web research, planning technology has been applied to enhance the flexibility in Web Service composition. However, only simple composition plans have been considered up to now. Typically, the interaction with the composed service has a very short duration, so that persistence management is not needed; moreover, the invocation of Web Service suppliers is elementary because their interaction protocols consist of exchanging very few *request* and *response* messages; e.g., see (McIlraith et al., 2001; Balke and Wagner, 2003; Keidl and Kemper, 2004). Indeed, (Laures and Jank, 2005) deals with persistent workflow composition and dynamic selection of service providers. However, only a reference implementation is provided, which does not specify details about the execution model.

As a matter of fact, relying on a standard workflow engine for the management of the business logic of an application has scalability and robustness advantages, which are not guaranteed by other technologies, such as planners. In fact, several proposals for the adoption of planners in Web Service composition have turned out to exploit workflow engines for the service execution; e.g., (Mandell and McIlraith, 2003; Laures and Jank, 2005).

4 CONCLUSIONS

We have presented the CAWE framework for the development of applications composing Web Service suppliers in context-sensitive workflows. The context-aware workflow execution is based on the introduction of abstract activities hiding the context-dependent details; abstract activities can be performed by following different courses of action, depending on the context state.

In order to test the functionality of the CAWE

framework and its applicability in realistic domains, we have developed a proof-of-concept prototype system and we have instantiated it on a medical domain.

The approach proposed in this paper has the following advantages:

- First of all, the introduction of abstraction enhances the readability of the context-sensitive workflow, which could be very hard to understand, if represented as a flat graph including all the alternative courses of action.
- Second, the organization of context-sensitive workflows as a hierarchy of simpler workflows with applicability conditions supports a seamless extension of the business logic of an application to take new contextual conditions and new courses of action into account. In fact, a context-sensitive workflow can be extended by replacing any concrete activity with an abstract activity and its implementations. Moreover, given an abstract activity, a new context-dependent implementation can be added by revising the applicability conditions of the existing implementations and introducing the new one in the workflow specification.
- Finally, the introduction of abstraction enables the designer to define the workflow top-down, starting from a general view of the activities to be managed, and specifying the low-level details later on. In our medical domain, this separation helped us to focus first on the most relevant aspects of the medical guideline (provided by a physician), and to postpone the specification of details such as reserving an ambulance or requesting a nurse.

It should be noticed that the hierarchical workflows we propose can be executed by a standard workflow engine without the Personalization Module. In fact, although abstract activities and contextual conditions extend the syntax of standard workflow languages, a hierarchical workflow could always be translated to a flat one by replacing abstract activities with decision points and by including the alternative courses of action directly into a flat workflow. However, this translation would generate workflows which are very difficult to read and debug. For this reason, we preferred to extend the capabilities of the workflow engine by making it invoke the Personalization Module, specialized in the execution of adaptation strategies. As discussed in the paper, even this extension can be performed by a standard workflow engine without modifying its core, if the context-sensitive workflow is translated to an intermediate representation which maintains its hierarchical structure but expands the code of the abstract activities. We believe that our approach is strategic because the Personalization Module, which now evaluates boolean conditions, might be extended to handle complex adaptation rules, without modifying the rest of the system.

ACKNOWLEDGEMENTS

This work is supported by the EU (project WS-Diamond, grant IST-516933) and by MIUR (project QuaDRAnTIS).

REFERENCES

Balke, W. and Wagner, M. (2003). Towards personalized selection of Web Services. In *Proc. of 12th Int. World Wide Web Conference (WWW'2003)*, Budapest.

Benlismane, D., Maamar, Z., and Ghedira, C. (2005). A view-based approach for tracking composite Web Services. In *Proc. of European Conference on Web Services (ECOWS-05)*, pages 170–179, Växjö, Sweden.

Firby, R. (1987). An investigation into reactive planning in complex domains. In *Proc. 6th Conf. AAAI*, Seattle (WA).

Georgeff, M. and Lansky, A. (1987). Reactive reasoning and planning. In *Proc. 6th Conf. AAAI*, pages 677–682, Seattle.

Jensen, K. (1976). *Coloured Petri nets: basic concepts, analysis methods and practical use*. Springer-Verlag, Berlin.

Keidl, M. and Kemper, A. (2004). Towards context-aware adaptable Web Services. In *Proc. of 13th Int. World Wide Web Conference (WWW'2004)*, pages 55–65, New York.

Koenig, J. (2004). JBoss jBPM white paper. http://www.jboss.com/pdf/jbpm_whitepaper.pdf.

Laures, G. and Jank, K. (2005). Adaptive Services Grid Deliverable D6.V-1. Reference architecture: requirements, current efforts and design. Technical report, http://asg-platform.org/cgi-bin/twiki/view/Public/ReferenceArchitecture.

Mandell, D. J. and McIlraith, S. A. (2003). Adapting BPEL4WS for the Semantic Web: The bottom-up approach to Web Service interoperation. In *LNCS 2870, Proc. 2nd International Semantic Web Conf. (ISWC 2003)*, pages 227–241. Springer-Verlag, Sanibel Island, Florida.

McIlraith, S., Son, T., and Zeng, H. (2001). Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46–53.

OASIS (2005). OASIS Web Services Business Process Execution Language. http://www.oasis-open.org/committees/ documents.php?wg_abbrev=wsbpel.

van der Aalst, W. (2002). Making work flow: on the application of Petri Nets to Business Process Management. In *Proc. of 23rd Int. Conf. on Applications and Theory of Petri Nets*, pages 1–22, Adelaide, South Australia.