J Inf Process Syst, Vol.10, No.3, pp.365~383, September 2014
http://dx.doi.org/10.3745/JIPS.02.0003
ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

A Novel Framework for Defining and Submitting Workflows to Service-Oriented Systems

Hayat Bendoukha*, Yahya Slimani**, and Abdelkader Benyettou*

Abstract—Service-oriented computing offers efficient solutions for executing complex applications in an acceptable amount of time. These solutions provide important computing and storage resources, but they are too difficult for individual users to handle. In fact, Service-oriented architectures are usually sophisticated in terms of design, specifications, and deployment. On the other hand, workflow management systems provide frameworks that help users to manage cooperative and interdependent processes in a convivial manner. In this paper, we propose a workflow-based approach to fully take advantage of new service-oriented architectures that take the users' skills and the internal complexity of their applications into account. To get to this point, we defined a novel framework named JASMIN, which is responsible for managing service-oriented workflows on distributed systems. JASMIN has two main components: unified modeling language (UML) to specify workflow models and business process execution language (BPEL) to generate and compose Web services. In order to cover both workflow and service concepts, we describe in this paper a refinement of UML activity diagrams and present a set of rules for mapping UML activity diagrams into BPEL specifications.

Keywords—Service Composition, Service-Oriented Computing, Service-Oriented Workflow, UML2BPEL, Workflow

1. Introduction

Industrial, business, and scientific applications are becoming complex and require more and more storage and computing resources. Service-oriented and large-scale distributed systems provide efficient environments for resource sharing and distributed computing. They offer a variety of resources to satisfy the requirements of applications [1]. Usually, processes requiring important storage and/or computing resources are composed of a set of sub-processes, which are interdependent, share the same workspace (data, objects and/or resources, etc.), and have to respect a particular execution scheme to achieve a single objective.

We believe that simple and atomic processes rarely require important resources to be executed. To be efficient, new environments must not only provide all of the needed resources but also offer strong tools that are able to specify the internal complexity of applications.

As service-oriented technology increases in popularity [2], it is obvious that researchers try to design large-scale solutions, which incorporate web services, with the best tools for service composition and orchestration. Therefore, users are constrained in being able to master all the tools and languages employed in service technology. These users often come from different

Manuscript received April 23, 2013; accepted September 16, 2013; onlinefirst August 25, 2014.
Corresponding Author: Hayat Bendoukha (bendoukha@univ-usto.dz)
* Department of Computer Science, SIMPA Laboratory, University USTOMB Oran, Algeria ({bendoukha, a_benyettou}@univ-usto.dz)
** Department of Computer Science, University El Manar Tunis, Tunisia (yahya.slimani@fst.rnu.tn)

fields and have to know the fundamentals of their respective application areas. Forcing them to learn new concepts, in addition to all the specific tools used in their fields, increases the level of difficulty for them. We strongly believe that the best way to convince the majority of users from economic, industrial, research, and business fields to move their applications from traditional centralized or locally parallel/distributed computing systems to new large scale computing systems, such as service-oriented grids, is to hide all of the complexities of these systems.

Usually, users must deal with formal languages to compose and deploy services on service-oriented systems. These languages are based on extensible markup language (XML), such as Web service description language (WSDL), Web service deployment descriptor, Java naming and directory interface, hypertext transfer protocol (HTTP), etc. [3]. We believe that it is increasingly necessary to reduce the complexity of managing these tools by associating service-oriented execution environments with user-oriented interfaces. These users-oriented interfaces are able to 1) attract a large number of users, 2) make easy communication between these users and the execution systems, and 3) increase the benefits in terms of performance and workability.

In this paper, we propose an approach that links the efficiency of service-oriented systems and the conviviality of user-friendly composition tools, such as workflows [4]. We propose a novel framework called JASMIN, which allows for applications to be submitted and visualized in a service-oriented distributed system. JASMIN supports workflow specifications with unified modeling language (UML) activity diagrams and uses business process execution language (BPEL) for service composition. In order to cover both the workflow and service concepts, we have defined a set of refinements on the notations of UML activity diagrams. We also present some mapping rules from UML activity diagrams into BPEL specifications.

The remainder of this paper is organized as follows: in Section 2, we highlight the concepts of service-oriented workflows through some existing systems. Section 3 presents our proposal, describes the architecture of the proposed framework, and presents the lifecycle of a service-oriented workflow. Section 4 describes the JASMIN components that are essentially related to the refinements of UML notations, the mapping rules from UML activity diagrams into BPEL documents, and the deployment of services on a distributed system. We present a case study in Section 5 presents. In Section 6, we compare our framework with some other related works in order to highlight the characteristics of our framework. Section 7 concludes the paper and proposes future research on this topic.

2. RELATED WORK

Workflow has emerged as a useful paradigm to describe, manage, and share complex scientific analysis and business processes [5]. Workflows represent the codes or components that are to be executed in a complex application, as well as the data dependencies among these components [4]. Compared to traditional business workflows, new scientific workflows have two main characteristics: 1) they represent scientific processes in which the complexity is not fixed by human interactions, but by the interdependencies and the requirements of the involved activities and 2) they are mostly service-oriented. While composing new complex applications as scientific workflows, the concepts (rules, routes, and roles) presented by the trilogy of Marshak '3R' [6] change. In scientific workflows, rules expressing constraints and activity characteristics are more important than they are in business workflows. Contrary to rules, roles almost lose sense within new scientific processes. In addition, activities are often related to both stateless and stateful services, unlike the common workflows of business processes where only web stateless services are composed [7]. We will now talk about service-oriented workflows several projects and approaches for modeling, composing, executing, and monitoring workflows on service-oriented systems are being developed. These projects achieved to a variety of frameworks [8]. In the following sections, we present a survey on some projects. In this paper, we focus on modeling, submission, and visualization. Other aspects, such as the enactment of services, are left out.

ASKALON [9] is a grid application development and computing environment, which provides services for composing, scheduling, and executing scientific workflows in a grid system. Services in ASKALON implement Web services resource framework [10] by using the Globus toolkit 4.0 [3]. Applications can be composed using an UML-based workflow composition with the Teuta workflow environment [11] or by using the XML-based abstract grid workflow language (AGWL) [12].

Kepler [13] is one of the most popular workflow systems. It has advanced features for composing scientific applications. Kepler allows for drag-drop creations and the execution of workflows for distributed applications. Workflows are modeled in modeling markup language (MoML) [14].

Taverna [15] is a collaboration between the European Bioinformatics Institute, IT Innovation, and the Human Genome Mapping Project Resource Centre. The Taverna project aims to provide language and software tools to facilitate the use of workflow and distributed computer technology within the e-Science community. In Taverna, data models can be represented in the XML-based language, which is called the simple conceptual unified flow language (SCUFL) [16].

Triana [17] is a workflow-based graphical problem-solving environment for data mining applications developed at Cardiff University. Triana provides an UML-based visual programming interface with a functionality represented by units. Applications are written by dragging and connecting the required units onto the workplace to construct a workflow.

The above projects provide workflow-based frameworks for defining and composing applications [18]. Each project has adopted a particular approach for designing workflows but presents at least one flaw. Most of them are exclusively dedicated to a particular application domain. Only Kepler is generic and it provides an application-free solution. Also, few frameworks consider users' skills. Although ASKALON, Triana, and Kepler provide graphical interfaces to define workflows, users still need to master the fundamentals of workflow concepts and some specific languages to interact with execution environments.

The objective of our proposal is to make service-oriented systems more efficient and more transparent to individual users. This is possible if we make it easy for users to interact with the execution environments. In order to gain in both efficiency and transparency, we are proposing the generic framework of JASMIN, which is responsible for submitting and visualizing service-oriented workflows [19]. This framework represents the user front-end of a distributed system. It interacts with other service-based and workflow enactment engines to accomplish the execution of applications. In order to facilitate the use of service-oriented architectures to deploy and execute workflows, the proposed framework covers the three main aspects that are listed below.

1) **The user aspect**. To make our framework accessible for a large community of users, we

propose a graphical user interface (GUI) that is easy for both expert and non-expert users to handle.

2) **The design aspect.** The basic UML notations are well adapted to specify complex and collaborative applications, but they are not able to support service concepts. For this purpose, we define some refinements on the notations of UML activity diagrams to cover both workflows and services.
3) **The service aspect.** To make UML specifications executable on a service-oriented architecture, we need to transform them. For this purpose, we propose a set of rules that map UML into BPEL. BPEL is, at the same time, a service composition and a workflow execution language.

The architecture of the JASMIN framework is widely based on the three aspects explained above. The architecture and the functionalities of our framework are illustrated below.

3. JASMIN: A VISUAL FRAMEWORK FOR SUBMITTING AND VISUALIZING SERVICE-ORIENTED WORKFLOWS

Our proposal takes into account both the physical constraints of the execution platform and users' skills. We took into consideration the fact that the execution environment is service-oriented and we assumed that the users would not necessarily be experts but that instead will need to be assisted during the submission and the execution of their complex applications. In addition, workflow concepts allow us to see applications through the flow of performed actions. Each application is defined as a set of interdependent activities. The routing rules describe the interdependencies as the control flow of a workflow model. They define a formal view of a coordinated set of activities to accomplish the same goal.

3.1 The Architecture of the JASMIN Framework

The JASMIN framework is designed in two main layers: the workflow definition layer and the service composition layer. Each layer is responsible for one or more specific tasks in the entire process of managing applications within a service-oriented system. JASMIN is workflow-oriented in the first layer and service-oriented in the second one.

Workflow definition. In this layer of JASMIN, we focus on the workflow. Its components are user-oriented. It is responsible for specifying users' applications as workflow models without any representation of the physical infrastructure of the execution environment. All the interactions between users and JASMIN are supported by the GUI of the workflow model editor, which is the key component of this layer.

Service composition. Here, we take into account the execution platform and the physical resources. To be executed, workflow models need to be transformed into formal specifications so that they can be handled as services. The transformation tool responsible for the service generation includes the UML2BPEL library and the workflow instance generator. In the service enactment step, a workflow engine and a service container are needed to deploy obtained services on a distributed environment.

The separation between the specification of applications and their execution provides an easy management of repetitive processes. Each time users want to execute an instance of a repetitive process; they use a predefined abstract model of the given process and do not need to redefine it. Also, users do not specify conditions on the physical nodes. This introduces a high degree of transparency and facilitates the interaction between users and the execution environment. Users submit their applications and follow the evolution of their execution through a graphical interface that is easy to handle. This will undoubtedly attract more users.

The architecture of JASMIN and its interactions are shown in Fig. 1.

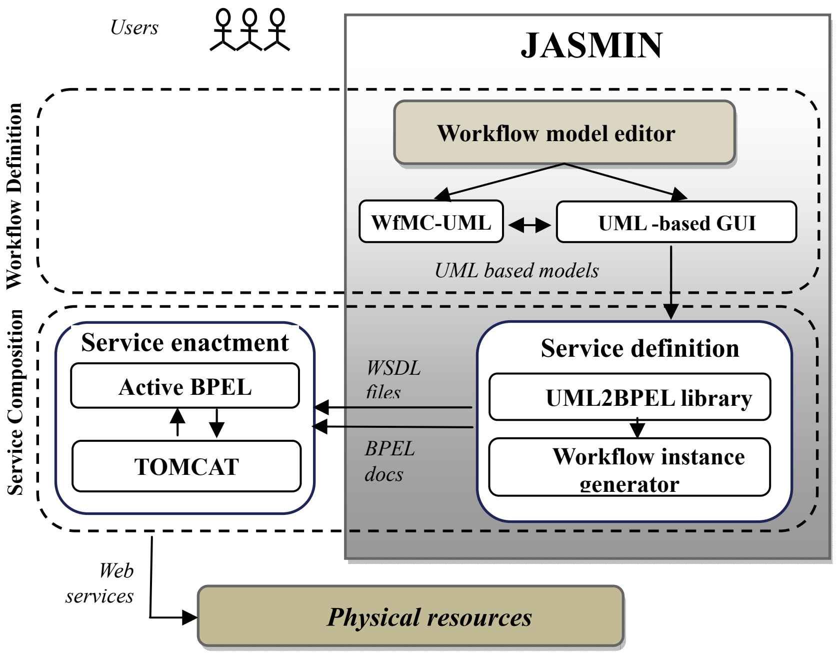


Fig. 1. The architecture of the JASMIN framework. UML=unified modeling language, BPEL=business process execution language, GUI=graphical user interface, WSDL=Web service description language, WfMC=workflow management coalition.

3.2 The Lifecycle of a Service-Oriented Workflow

The execution process of a distributed application includes many steps that involve the different components of JASMIN and some other environments.

Initially, users have their applications as a set of programs written mostly in C or Java. First, users start by extracting interdependencies and routing rules between sub-processes in order to define the workflow model of their application in the form of a UML activity diagram [20]. This task is accomplished by the workflow model editor, which uses a library named WfMC-UML, for enabling the specification of all the coordination rules defined by the WfMC [21] with UML notations. At the end of the Workflow Definition step, users have a UML diagram that corresponds to the entire application. It is composed of a set of sub-processes representing the interdependent programs.

Once created, the workflow models need to be transformed into services to proceed to their execution. The workflow instance generator, using the UML2BPEL library, maps UML models into BPEL specifications [22]. It is important to note that workflow models defined as UML diagrams and BPEL documents do not give the internal description of the actions to be performed but instead provide the coordination between them. They describe the behavior and define the execution order of the involved services and their interdependencies. In order to deploy these services on a BPEL-based workflow engine, the information available in the BPEL document is complemented with a description of the static characteristics of the entire service and all other services that are involved. This description is usually given in WSDL [23]. A WSDL file includes the name of each service (which program has to be executed when a given service is invoked), its location (which service is available on which node), the objects manipulated by services, their inputs and outputs, etc.

The service composition step finishes with the enactment of the application on a distributed environment. In order to deploy services that take the workflow aspects of the application into account, a workflow engine is needed. In our case, we chose ActiveBPEL [24] based on Apache Tomcat [25]. Fig. 2 describes the different for managing a user's application from its submission to its enactment. It shows all the transformations of the service-oriented workflow process X, which is composed of a set of programs (x_1, x_2, ..., x_n).

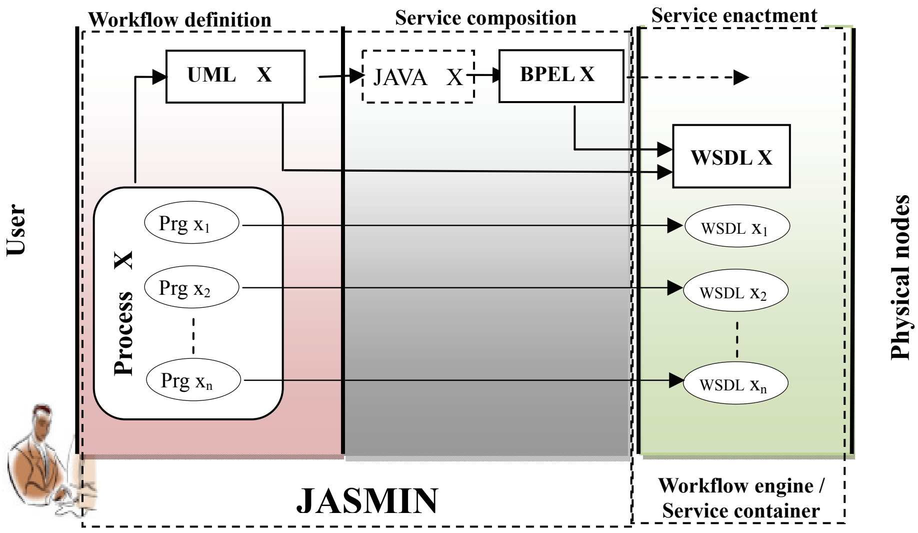


Fig. 2. The lifecycle of a service-oriented workflow. UML=unified modeling language, BPEL= business process execution language, WSDL=Web service description language.

4. JASMIN Components

Our greatest concern was making sure to introduce more transparency and ease of work on service-oriented architectures, while taking into account the different kinds of users in various application domains. In order to hide the complexity of the execution environment, the workflow model editor is mainly composed of a GUI. This GUI is based on the UML formalism. It is responsible for specifying and submitting applications. Currently, several editors of UML diagrams exist, such as ArgoUML [26]. The main role of our contribution over these editors is that we focused on both workflows and services. Compared to XML-based languages, such formalism is relatively easy to handle by users, especially if these formalisms are guided while generating models.

For this purpose, we introduced some refinements on the UML formalism. Furthermore, UML formalism does not specify service concepts. To be handled within a service-oriented environment, UML models must be transformed into a common service specification. There are many workflow formal languages [27], but BPEL is the standard for describing service composition. BPEL contains constructs for control flow and data manipulation, as well as interactions with Web services that implement tasks in a workflow [22]. We are proposing a set of mapping rules from UML activity diagrams to BPEL documents.

4.1 UML Refinements for Users

Our objective through this first set of refinements is to allow users to specify their most complex applications. We took two different types of users into consideration: users who are familiar with workflow languages for process management (expert users) and those who have no expertise on working with UML and workflow (non-expert users). For each type, we provided a set of patterns to be used while modeling processes. This simplifies the work of users and minimizes the users' intervention while submitting and deploying their applications on a service-oriented system [28].

4.1.1 JASMIN nodes for expert users

Expert users represent users from scientific or business fields who already deal with workflow technology and UML formalism. These users handle only standard patterns of UML activity diagrams with no additional patterns or specifications. Our tool provides a toolbar with all of the necessary patterns to specify different applications as UML activity diagrams, such as activity, transition, condition edge, synchronization bar, begin, and end nodes. These patterns are given in Fig. 3.

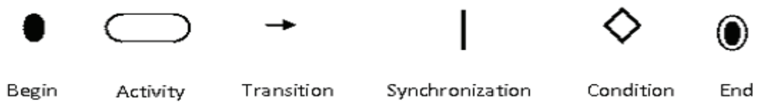


Fig. 3. Toolbar nodes for expert users.

A user can drag any node and drop it and can create a workflow by matching these different patterns together. Even for expert users, the probability of obtaining an incorrect model is very low, since many of the control rules are already programmed so that users respect the UML formalism. These control rules are available in two ways: 1) dialog boxes to indicate to users every step (pattern suspected, information missing, possible routing rules, etc.) and 2) automatic generation of the suspected pattern when only one node can be matched to the last pattern.

4.1.2 UML prototypes for non-expert users

We believe that users coming from different application areas do not necessarily have expertise regarding workflow and UML. Since transparency has been our primary concern, we offered through the workflow model editor a set of 'prototypes' corresponding to the recurrent workflow routing rules. Users from this class are able to identify the different activities that make up the entire process and their interdependencies. They can use the predefined sub-workflows corresponding to different types of interdependencies to build their entire workflow. The prototypes include sequential routing, parallel routing (fork, join), and selective routing (switch) [29]. Fig. 4 shows the prototypes that are available in our workflow model editor.

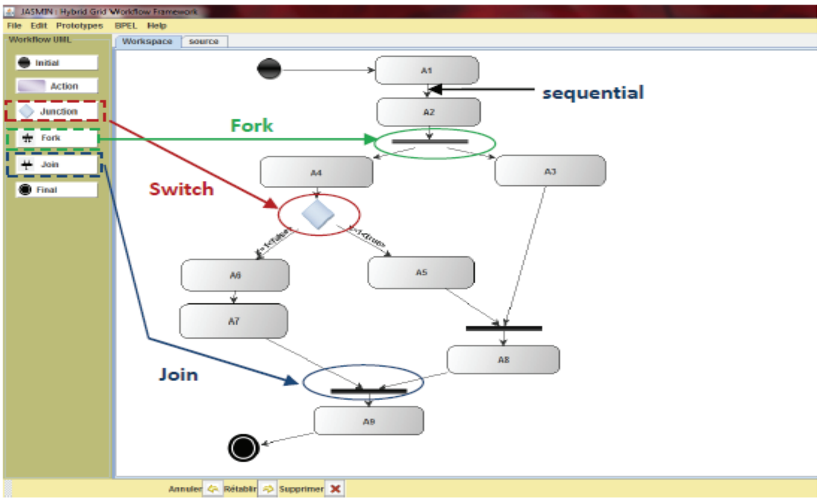


Fig. 4. Unified modeling language prototypes for non-expert users.

4.2 UML Refinements for BPEL Generation

Since we plan to execute our applications on a distributed service-oriented environment, workflow modeling tools have to be able to interact with service composition tools in order to extract services from the obtained UML models. The standard formalism of UML does not cover service-oriented concepts. UML-based user interfaces usually provide information on activities like names, shared objects, routing rules, dependencies, etc. In order to specify service-oriented workflow applications, our framework must provide additional information about the type of activity, activity communication ports with other activities, etc. These additional pieces of information allow the workflow instance generator to compose a concrete workflow of activities as services to be deployed in coordination with other services.

In BPEL notation, both activities and interdependencies are supported. Compared to traditional UML, more notions are present in a BPEL definition. Once made, UML activity diagrams have to be managed by service tools. This is made possible by enhancing UML notations, such as activities, transitions, and routing rules, with some other patterns like activity properties including types, variables, port types, and partner links [22]. These new patterns are introduced in the UML models to make it easier to generate BPEL instances. In JASMIN, each time an activity is inserted, a set of properties related to BPEL notions must be introduced, as shown in Fig. 5.

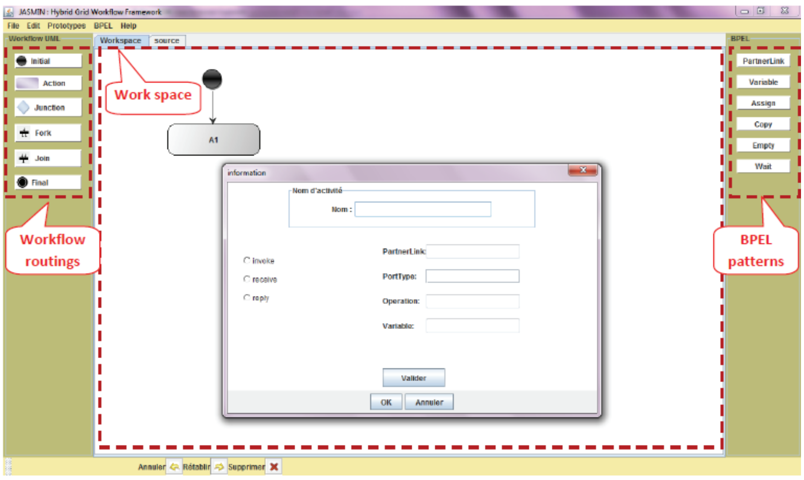


Fig. 5. The properties of business process execution language (BPEL) activities.

4.3 Mapping UML Activity Diagrams into BPEL Documents

The workflow model editor helps to generate the UML models that correspond to a given process. However, even by refining UML diagrams, the execution of the abstract workflows obtained by this interface is impossible, unless we translate these models into executable instances. Service composition tools are responsible for extracting the executable jobs of workflow instances from the initial graphical models. In other words, these tools generate a set of services written in a specific formal language from the UML activities flow. These services can then be deployed within a service-oriented system. For this purpose, the workflow instance generator ensures the mapping of UML activity diagrams into BPEL documents, using the UML2BPEL library.

4.3.1 UML2BPEL library

The workflow instance editor generates the BPEL specifications. This is possible thanks to the UML2BPEL library and the information introduced by the user while editing the UML diagram. The UML2BPEL library includes information corresponding to both the so-called basic activities and complex activities. The basic activities include the invoke activity, the receive activity, and the reply activity. The complex activities represent a set of basic activities grouped by workflow routing rules, such as the flow and the sequence.

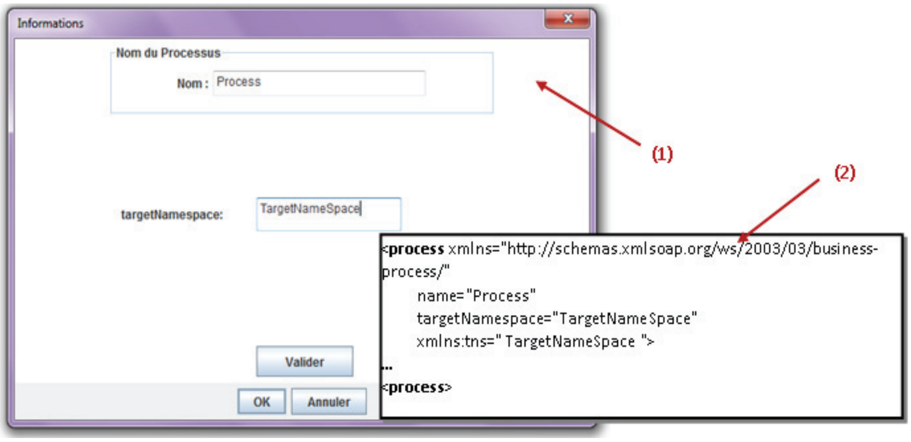
For the purpose of portability, the mapping of UML diagrams into BPEL documents is divided into two steps. The first step consists on mapping UML diagrams into Java code, while the second one consists of mapping the obtained Java code into BPEL documents. The mapping from UML to Java is transparent to users. The intermediate Java code corresponding to the behavior of the sub-processes and their interdependencies may facilitate a future mapping of UML models into another formal language or creating BPEL documents from other semi-formal notations if they are coded in Java.

4.3.2 Workflow instance generator

The workflow instance generator is responsible for generating the BPEL documents that correspond to UML models. When users define a new activity or introduce a new pattern related to any activity, the workflow instance generator produces the corresponding code in BPEL, using the UML2BPEL library. A BPEL document is filled gradually while users are editing UML diagrams on the workflow model editor. In addition to generating BPEL patterns related to basic activities, we also made the generation of BPEL routings automatic. At this level, we implemented some rules to map workflow routings from UML activity diagrams into the control flows in BPEL documents. The JASMIN framework provides a transparent way for generating the BPEL tags corresponding to the most recurrent WfMC routing rules, which we already presented in the sections above (sequential routing, parallel routing, selective routing, and iterative routing).

4.3.3 Mapping rules corresponding to the most recurrent BPEL activities

As mentioned above, the graphical interface of JASMIN assists the user while editing his/her UML model. First, when the user chooses the ‘begin’ pattern, he/she gets a dialog box allowing him/her to insert information about the entire process, such as its name and the target namespace. Fig. 6 shows, in ‘(1)’ the dialog box for the process definition in the UML model, and in ‘(2)’ the heading of the BPEL document that is generated by the workflow instance generator, corresponding to a process named ‘process.’

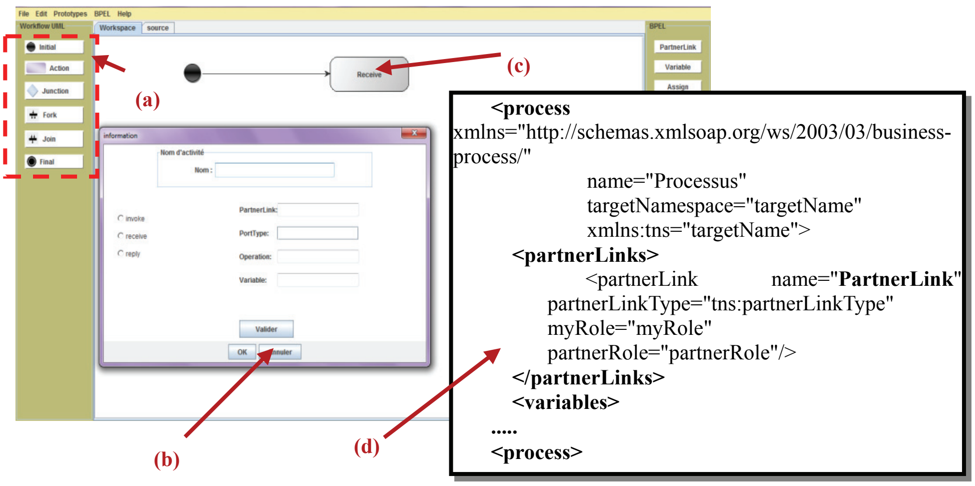


```
<process xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
    name="Process"
    targetNamespace="TargetNameSpace"
    xmlns:tns=" TargetNameSpace ">
...
<process>
```

Fig. 6. The heading of a business process execution language (BPEL) document.

The user gradually composes his/her process through the graphical interface of JASMIN. The BPEL specification corresponding to the resulted UML diagram of the process is then generated. Fig. 7 shows the main steps of the modeling and mapping processes.

Each time the user wants to insert a basic activity or a structured one, he/she starts by choosing the corresponding pattern from the toolbar on the left of the graphical interface (labeled by ‘(a)’ in Fig. 7). He/she gets the dialog box ‘(b)’, which allows the user to introduce the activity properties, including the type, name, partnerLink, variables, etc. These properties change from one type of activity to another. When the user validates the properties, the corresponding activity ‘(c)’ is added to the diagram. The activity properties are not visible in the UML diagram, but are saved in the corresponding Java code to generate the BPEL tags. In addition to the tags inserted in the activities of the BPEL document, variables and partnerLinks of every new activity are defined in the heading of the BPEL document ‘(d).’



```
<process
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-
process/"
                name="Processus"
                targetNamespace="targetName"
                xmlns:tns="targetName">
   <partnerLinks>
                <partnerLink            name="PartnerLink"
           partnerLinkType="tns:partnerLinkType"
           myRole="myRole"
           partnerRole="partnerRole"/>
   </partnerLinks>
   <variables>
.....
<process>
```

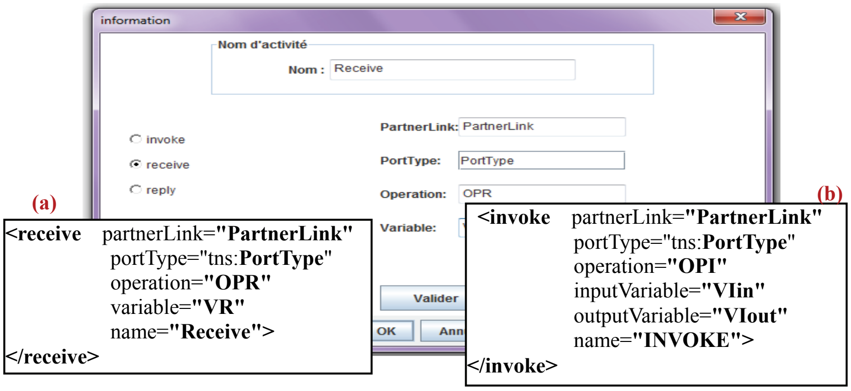
Fig. 7. The main steps of the modeling and mapping processes.

In this section, we present some mapping rules corresponding to the most recurrent BPEL activities. We also show how their respective UML specifications are mapped into BPEL.

1) Receive: The receive activity is often used to initiate a process. It is typically the first construct that appears in a BPEL process. It accepts the data from the incoming service message and places it into a variable that is accessible for the remaining services. It specifies a partner link, a port type, an operation to be invoked, and a variable where the received data can be placed.

2) Invoke: It invokes a particular service as requested. With an invoke activity a process can call another Web service that has been defined as a partner. The invoke can be either asynchronous, which needs to specify only an input variable, or synchronous, which needs both input and output variables.

Fig. 8 shows part of a BPEL document as it is generated by JASMIN for a receive activity in (a) and an invoke activity in (b).



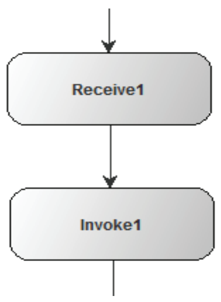
Nom : Receive
PartnerLink: PartnerLink
PortType: PortType
Operation: OPR

```
<receive    partnerLink="PartnerLink"
             portType="tns:PortType"
             operation="OPR"
             variable="VR"
             name="Receive">
</receive>
```

```
<invoke    partnerLink="PartnerLink"
            portType="tns:PortType"
            operation="OPI"
            inputVariable="VIin"
            outputVariable="VIout"
            name="INVOKE">
</invoke>
```

Fig. 8. (a) The receive activity, (b) the invoke activity.

3) Sequence: A sequence activity has one or more activities that are executed one after another in the order they are placed within the sequence element. The sequence activity stops when all activities within it are done. Fig. 9 presents a sequence of two activities Receive1 then Invoke1.

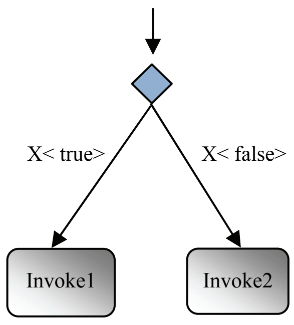


```
<sequence>
    <receive  partnerLink="PartnerLink"  portType="tns:PortType"
                        operation="OPR1"          variable="VR1"
name="Receive1">
    </receive>
    <invoke  partnerLink="PartnerLink"  portType="tns:PortType"
                                    operation="OPI1"
inputVariable="VI1in"
            outputVariable="VI1out" name="Invoke1">
    </invoke>
    </sequence>
    …
```

(a) Sequence in UML **(B) Sequence in BPEL**

Fig. 9. The sequence activity. UML=unified modeling language, BPEL=business process execution language.

4) Switch: The switch activity specifies a conditional behavior. This activity consists of an ordered list of conditional branches. Every branch is specified by a case element followed by one optional otherwise element. The activities specified in the case are executed when the condition of the case is true. When none of the cases are true, the activities in the otherwise element are executed. The switch activity is done when all the activities of one of the branches are completed. The switch in Fig. 10 is triggered by the condition X. If X is true, the activity Invoke1 is launched, which defines the case branch. If not, the otherwise branch and the activity Invoke2 are launched.

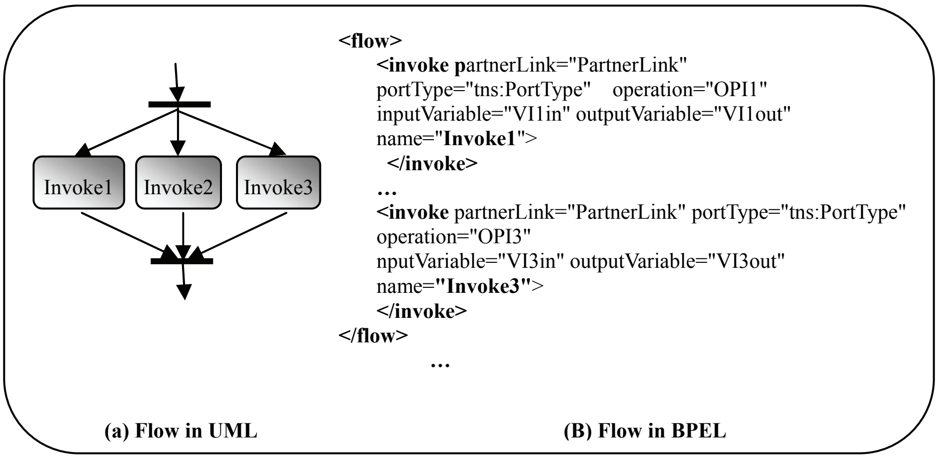


```
< switch >
<case condition="X">
    <invoke  partnerLink="PartnerLink" portType="tns:PortType"
    operation="OPI1"                        inputVariable="VI1in"
    outputVariable="VI1out" name="Invoke1">
    </invoke>
</case>
<otherwise>
      <invoke partnerLink="PartnerLink" portType="tns:PortType"
    operation="OPI2"     inputVariable="VI2in"     outputVariable=
    "VI2out" name="Invoke2">
      </invoke>
  </otherwise>
</switch>
…
```

(a) Switch in UML **(B) Switch in BPEL**

Fig. 10. The switch activity. UML=unified modeling language, BPEL=business process execution language.

5) Flow: The flow activity allows for the execution of several activities in the same time. A flow activity is completed when all its activities are completed. One of the possibilities offered by a flow activity is the synchronization of activities within the flow. In JASMIN, a flow activity is modeled by inserting a set of activities between the fork and the join patterns available in the toolbar of the graphical interface. Fig. 11 represents three activities: Invoke1, Invoke2, and Invoke3, which are defined in a flow activity.



```
<flow>
     <invoke partnerLink="PartnerLink"
     portType="tns:PortType"    operation="OPI1"
     inputVariable="VI1in" outputVariable="VI1out"
     name="Invoke1">
       </invoke>
     ...
     <invoke partnerLink="PartnerLink" portType="tns:PortType"
     operation="OPI3"
     nputVariable="VI3in" outputVariable="VI3out"
     name="Invoke3">
     </invoke>
</flow>
         ...
```

Fig. 11. The flow activity. UML=unified modeling language, BPEL=business process execution language.

4.4 Service Deployment and Enactment

The deployment and enactment of a service-oriented workflow require the existence of a BPEL-based workflow engine and a service container. Many workflow engines exist and the freely available ActiveBPEL engine is the most popular one [27]. ActiveBPEL is deployed as a servlet into the Jakarta Tomcat container of Apache. It has extensive documentation and it is totally managed on a Web interface [24]. Users gradually introduce all of the needed patterns, like the BPEL document of the entire process and the WSDL descriptions of all the involved services. This step requires a service container. In our case, services are deployed on the Apache Tomcat since ActiveBPEL is also based on it.

5. Case Study: Atmospheric Event Prediction Workflow

In this section, we present a workflow corresponding to a sub-process that is called the event handling workflow. It is from the atmospheric phenomena prediction by the Linked Environments for Atmospheric Discovery (LEAD) project [30]. LEAD aims to detect, analyze, and predict atmospheric phenomena. The process managed by the event handling workflow manages the atmospheric events according to their occurrence probability. The process may finish quickly when the event is considered uninteresting or it may continue running for long time to determine if anything interesting happens. Each time the event is received, the probability of its occurrence is computed. According to this probability, the event is considered either frequent or ignored. In this last case, the event can generate other events. These events require the execution of other workflows and eventually need to send an urgent notification to users to inform them that an important event, such as a tornado, is most likely to happen.

5.1 Workflow Model with UML

In order to be managed by JASMIN, the sub-process of the atmospheric event prediction presented above has to be modeled in the form of an UML activity diagram, as shown in Fig. 12. The UML specification of the above workflow is accomplished by users through the GUI of the workflow model editor. Users should begin with the extraction of the activities involved in the whole workflow and then move onto defining their interdependencies that represent workflow routings. The first activity, which starts the workflow, is the event occurrence named ReceiveEvent. It is followed by a conditional edge representing the probability of the event's occurrence. If the probability Event. Probability is less than 50%, a sequential branch should be launched to include two activities: the weather simulation of WeatherSim and the transfer of results to users SendResults. In the other case, a verification activity Verif should be launched to execute three parallel activities: the verification activity ModelVerication, the weather simulation WeatherSim, and the user notification activity NotifyUser.

5.2 Service Composition with BPEL

While editing the UML workflow model corresponding to the event handling workflow, users give additional information about the activities in order to prepare the deployment of the whole workflow. Fig. 13 illustrates the simplified syntax of a part of the BPEL document corresponding to the event handling workflow UML activity diagram presented in Fig. 12.

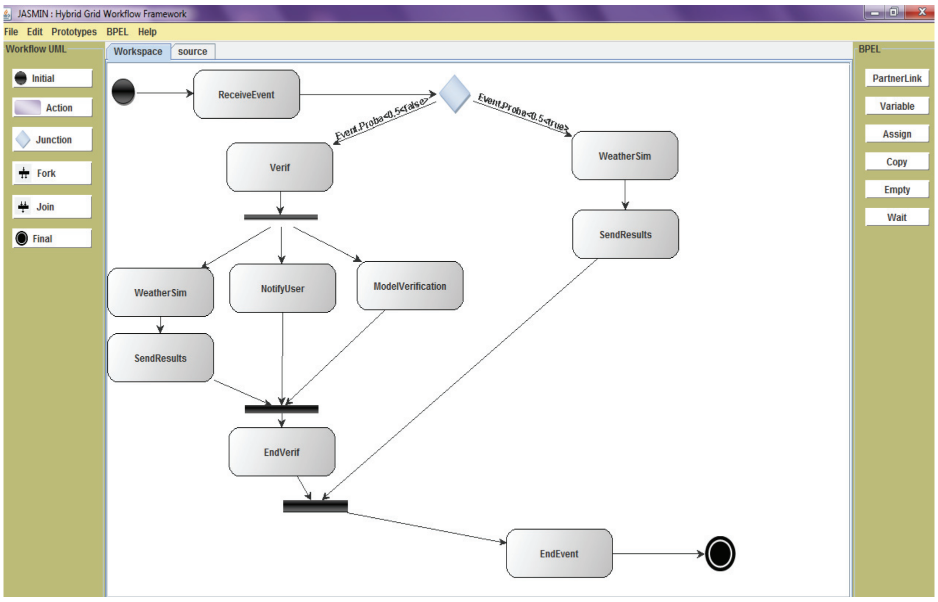


Fig. 12. The Unified modeling language (UML) activity diagram corresponding to the event handling workflow.

6. Discussion and Related Works

In this paper, we proposed the novel framework of JASMIN, which can be integrated as an outer layer in a service-oriented execution system. JASMIN is based on a GUI. It is responsible for the specification, the submission, and the visualization of users' applications in a convivial manner.

Compared to most of the other related works, our proposal essentially has three main characteristics. First, the architecture of JASMIN is application-free. It is not related to any particular application domain and does not require any specific environment to work, as opposed to others like Triana [17] and Knowledge Grid [31], which are oriented to work within distributed data mining applications on grids. Other frameworks like Taverna [15] and Kepler [13] provide workflow specifications and enactment tools for bioinformatics.

```
<process xmlns=http://schemas.xmlsoap.org/ws/2003/03/business-process/   name=
               "EventHandlingWorkflow" targetNamespace="tns"   xmlns:tns="tns">
…
<sequence>
     <receive partnerLink="WorkflowCaller" portType="tns:PortType" operation="ReceiveStatus"
                              variable="Event" name="ReceiveEvent">        </receive>
     <switch>
     <case condition="Event.Proba < 0.5">
          <sequence>
     <invoke partnerLink="WeatehrSimulationExecution" portType="fw:FastWeatherSim" operation= "Run
FastCheck" inputVariable="Event" outputVariable="RunResults" name="WeatherSim">   </invoke>
    <invoke partnerLink="DataMiningService" portType="dm:DataMining" operation="RunDataMining"
inputVariable="RunResults" outputVariable="SendStatus" name="SendResults">       </invoke>
          </sequence>
      </case>
     <otherwise>
          <sequence>
    <receive partnerLink="VerificationStarter" portType="tns:PortType" operation="VerifStart"
                            variable="Event" name="Verif">       </receive>
       <flow>
       <sequence>
  <invoke partnerLink="WeatehrSimulationExecution" portType="fw:FastWeatherSim" operation=
"RunFastCheck" inputVariable="Event" outputVariable="RunResults" name="WeatherSim"> </invoke>
  <invoke partnerLink="DataMiningService" portType="dm:DataMining" operation="RunDataMining"
inputVariable="RunResults" outputVariable="SendStatus" name="SendResults">       </invoke>
         </sequence>
<invoke partnerLink="NotificationService" portType="dm:UserNotificationService" operation="Notify"
inputVariable="NotifyMsg" outputVariable="null" name="NotifyUser">       </invoke>
<invoke partnerLink="ModelVerificationService" portType="dm:ModelVerification" operation="Verify"
  inputVariable="Event" outputVariable="VerificationResults" name="ModelVerification">     </invoke>
         </flow>
<reply partnerLink="VerifEndSce" portType="tns:PortType" operation="VerifEnd" varia-
ble="VerificationResults " name="EndVerif">       </reply>
     </sequence>
     </otherwise>
     </switch>
<reply partnerLink="WorkflowEndService" portType="tns:PortType" operation="WorkflowEnd" variable=
"SendResults" name="EndEvent">        </reply>
</sequence>
…
</process>
```

Fig. 13. A simplified business process execution language (BPEL) document corresponding to the event handling workflow.

Second, many frameworks like Triana, Knowledge Grid, or GridAnt are based on self-defined notations to compose their workflow models. In order to handle these notations, a learning phase is required for users to master them. Furthermore, models generated by self-defined notations are difficult to verify and validate since corresponding toolkits do not provide any verification tool. For example, the Knowledge Grid in its recent workflow-based version proposes a workspace where nodes can be data sources, algorithms, tools, or models [31]. These kinds of notations are not compliant with the workflow basics defined by the workflow management coalition (WfMC). According to the WfMC [21], workflow nodes have to represent tasks (or activities) and the edges represent their interdependencies while any other important information, such as requirements and manipulated objects, are considered as properties stored as workflow rules. We believe that it is safe to use standard tools. We opted for the Object Management Group standard UML to specify the workflow models and BPEL to compose services.

Third, our framework is mainly user-oriented. Users interact with JASMIN through a friendly graphical interface. This interface does not require any specific expertise in formal languages or sophisticated, large-scale, distributed execution environments. JASMIN is based on UML for submitting applications. It takes into account two kinds of users: those who are familiar with workflow technology and UML notations and those who do not have enough knowledge about these features. This characteristic related to workflow and UML makes our framework much easier to handle then other frameworks that are completely based on XML-based languages. For example, generic workflow execution service [32] implements dynamic workflow concepts by means of the generic workflow description language (GWorkflowDL) [33] and GridAnt [34] is an extension of the Apache Ant build tool residing in the Globus COG kit and is based on grid service flow language (GSFL) [35].

7. Conclusion

The increasing demand of today's applications leads developers to propose sophisticated solutions, such as service-oriented computing systems. These solutions provide environments to deploy and execute complex applications on heterogeneous and geographically distributed nodes. Currently, with the large use of service-oriented technologies, workflow tools and the languages of service composition are converging more and more [8]. In this paper, we proposed a hybrid framework called JASMIN, which allows for users to specify, submit, and visualize their applications so that they can be executed on a service-oriented distributed environment. The main challenge we tackled was to take advantage of both workflow and service techniques in an easy and transparent way. JASMIN is based on UML and BPEL for workflow and service specifications. In this paper we have presented a set of refinements that we defined on UML activity diagrams, in order to cover service concepts and some mapping rules from UML diagrams into BPEL documents. We intend to deploy our framework as a service in a service-oriented system like Taverna or caGrid [36].

References

[1] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, 2nd ed. Boston, MA: Morgan Kaufmann, 2004.

[2] G. Baryannis, O. Danylevych, D. Karastoyanova, K. Kritikos, P. Leitner, F. Rosenberg, and B.

Wetzstein, "Service composition," in *Service Research Challenges and Solutions for the Future Internet, Lecture Notes in Computer Science Volume 6500*, M. Papazoglou, K. Pohl, M. Parkin, and A. Metzger, Eds., Heidelberg: Springer Berlin, 2010, pp. 55-84.

[3] B. Sotomayor, "The Globus toolkit 4 programmer's tutorial," 2005; http://www.jpgrid.org/ documents/ pdf/document/GT4Progtutorial_E.pdf.

[4] W. van der Aalst and K. M. van Hee, *Workflow Management: Models, Methods, and Systems*. Cambridge, MA: MIT Press, 2002.

[5] M. Sonntag, D. Karastoyanova, and F. Leymann, "The missing features of workflow systems for scientific computations," in *Software Engineering 2010, Lecture Notes in Informatics Volume P-160*, G. Engels, M. Luckey, A. Pretschner, and R. Reussner, Eds., Bonn: Gesellschaft für Informatik, 2010, pp. 209-216.

[6] R. T. Marshak, "Workflow white paper: an overview of workflow software," in *Proceedings of the Workflow'94 Conference,* San Jose, CA, August 28, 1994.

[7] W. Dou, J. L. Zhao, and S. Fan, "A collaborative scheduling approach for service-driven scientific workflow execution," *Journal of Computer and System Sciences*, vol. 76, no. 6, pp. 416-427, Sep. 2010.

[8] G. C. Fox and D. Gannon, "Special issue: workflow in grid systems," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1009-1019, Aug. 2006.

[9] T. Fahringer, R. Prodan, D. Rubing, F. Nerieri, S. Podlipnig, Q. Jun, M. Siddiqui, T. Hong-Linh, A. Villazon, and M. Wieczorek, "ASKALON: a Grid application development and computing environment," in *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, Seattle, WA, November 13-14, 2005, pp. 122-131.

[10] OASIS Web Services Resource Framework (WSRF) Technical Committee [Online]. Available: http://www.oasis-open.org.

[11] T. Fahringer, S. Pllana, and J. Testori, "Teuta: tool support for performance modeling of distributed and parallel applications," in *Computational Science-ICCS 2004, Lecture Notes in Computer Science Volume 3038*, M. Bubak, G. van Albada, P. A. Sloot, and J. Dongarra, Eds., Heidelberg: Springer Berlin, 2004, pp. 456-463.

[12] T. Fahringer, Q. Jun, and S. Hainzer, "Specification of grid workflow applications with AGWL: an Abstract Grid Workflow Language," in *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid*, Cardiff, UK, May 9-12, 2005, pp. 676-685.

[13] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the Kepler system," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1039-1065, Aug. 2006.

[14] A. E. Lee and S. Neuendorffer, "MoML a modeling markup language in XML version 0.4," University of California Berkeley, Berkeley, CA, Technical Memorandum UCB/ERL M00/12, 2000.

[15] D. Turi, P. Missier, C. Goble, D. De Roure, and T. Oinn, "Taverna workflows: syntax and semantics," in *Proceedings of the IEEE International Conference on e-Science and Grid Computing*, Bangalore, India, December 10-13, 2007, pp. 441-448.

[16] G. Hobona, D. Fairbairn, H. Hiden, and P. James, "Orchestration of grid-enabled geospatial web services in geoscientific workflows," *IEEE Transactions on Automation Science and Engineering*, vol. 7, no. 2, pp. 407-411, Apr. 2010.

[17] A. Harrison, I. Taylor, I. Wang, and M. Shields, "WS-RF workflow in Triana," *International Journal of High Performance Computing Applications*, vol. 22, no. 3, pp. 268-283, Aug. 2008.

[18] J. Yu and R. Buyya, "A taxonomy of scientific workflow systems for grid computing," *ACM SIGMOD Record*, vol. 34, no. 3, pp. 44-49, Sep. 2005.

[19] H. Bendoukha, Y. Slimani, A. Benyettou, "JASMIN: a visual framework for managing applications in service-oriented grid systems," in *Proceeding of the 7th International Conference on Internet and Web Applications and Services*, Stuttgart, Germany, May 27-June 1, 2012, pp. 46-51.

[20] D. Skogan, R. Groenmo, and I. Solheim, "Web service composition in UML," in *Proceedings of the 8th IEEE International Enterprise Distributed Object Computing Conference*, Monterey, CA, September 20-24, 2004, pp. 47-57.

[21] L. Fisher, *Workflow Handbook 2004: Published in Association with the Workflow Management Coalition*. Lighthouse Point, FL: Future Strategies Inc., 2004.
[22] Web services business process execution language version 2.0 [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.
[23] Web Services Description Language (WSDL) 1.1 [Online]. Available: http://www.w3.org/TR/wsdl.
[24] ActiveVOS, Open source vs. open standards [Online]. Available: http://www.activevos.com/learn/open-source.
[25] Apache Tomcat [Online]. Available: http://tomcat.apache.org.
[26] ArgoUML [Online]. Available: http://www.argouml.org.
[27] M. Sonntag, D. Karastoyanova, and E. Deelman, "BPEL4Pegasus: combining business and scientific workflows," in *Service-Oriented Computing, Lecture Notes in Computer Science Volume 6470*, P. Maglio, M. Weske, J. Yang, and M. Fantinato, Eds., Heidelberg: Springer Berlin, 2010, pp. 728-729.
[28] H. Bendoukha, Y. Slimani, and A. Benyettou, "UML refinement for mapping UML activity diagrams into BPEL specifications to compose service-oriented workflows," in *Networked Digital Technologies, Communications in Computer and Information Science Volume 294*, R. Benlamri, Ed., Heidelberg: Springer Berlin, 2012, pp. 537-548.
[29] W. M. P. van der Aalst, "Workflow patterns," in *Encyclopedia of Database Systems*, L. Liu and M. T. ÖZsu, Eds., New York, NY: Springer US, 2009, pp. 3557-3558.
[30] LEAD–Linked Environments for Atmospheric Discovery [Online]. Available: http://vgrads.rice.edu/research/applications/lead.
[31] E. Cesario, M. Lackovic, D. Talia, and P. Trunfio, "A visual environment for designing and running data mining workflows in the knowledge grid," in *Data Mining: Foundations and Intelligent Paradigms, Intelligent Systems Reference Library Volume 24*, D. Holmes and L. Jain, Eds., Heidelberg: Springer Berlin, 2012, pp. 57-75.
[32] The Grid Workflow Execution Service (GWES) [Online]. Available: http://www.gridworkflow.org/kwfgrid/gwes.
[33] S. Pellegrini, F. Giacomini, A. Ghiselli, and A. Hoheisel, "Using GWorkflowDL for middleware-independent modeling and enactment of workflows," in *Proceedings of the CoreGRID Integration Workshop*, Crete, Greece, April 2-4, 2008.
[34] K. Amin, G. von Laszewski, M. Hategan, N. J. Zaluzec, S. Hampton, and A. Rossi, "GridAnt: a client-controllable grid workflow system," in *Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, Big Island, HI, January 5-8, 2004.
[35] S. Krishnan, P. Wagstrom, and G. von Laszewski, "GSFL: a workflow framework for grid services," Argone National Laboratory, Technical Report ANL/MCS-P980-0802, 2002.
[36] J. Saltz, S. Oster, S. Hastings, S. Langella, T. Kurc, W. Sanchez, M. Kher, A. Manisundaram, K. Shanbhag, and P. Covitz, "caGrid: design and implementation of the core architecture of the cancer biomedical informatics grid," *Bioinformatics*, vol. 22, no. 15, pp. 1910-1916, Aug. 2006.

Hayat Bendoukha

She received her Magister in Computer Science, option Artificial Intelligence and Patterns' Recognition from the University USTOMB Oran, Algeria, in 2005. She is currently pursuing his Ph.D. in Computer Science at the same university where she is also a Teacher Researcher since 2005. Her current research interests include service-oriented systems, distributed and grid-based systems, workflow management and web services.

Yahya Slimani

He studied at the Computer Science Institute of Alger's (Algeria) from 1968 to 1973. He received the B.Sc. (Eng.), Dr. Eng and Ph.D. degrees from the Computer Science Institute of Alger's (Algeria), University of Lille (French) and University of Oran (Algeria), in 1973, 1986 and 1993, respectively. He is currently Full Professor at the Department of Computer Science of Faculty of Sciences of Tunis. His research activities concern data mining, parallelism, distributed systems and Grid computing. Dr. Yahya Slimani has published more than 200 papers from 1986 to 2010. He contributed to Parallel and Distributed Computing Handbook, McGraw-Hill, 1996. He is currently Scientific Expert for the European Union. He joined the Editorial Boards of the Information International Journal in 2000, the J.UCS and others journals.

Abdelkader Benyettou

He received the engineering degree in 1982 from the Institute of Telecommunications of Oran and the M.Sc. degree in 1986 from the University USTOMB Oran, Algeria. In 1987, he joined the Computer Sciences Research Center of Nancy, France, where he worked until 1991 on Arabic speech recognition and received the Ph.D. in electrical engineering in 1993 from the USTOMB Oran University. From 1988 to 1990, he has been an assistant Professor in the Metz University, and Nancy-I University. He is actually professor at USTOMB Oran University since 2003. He is a researcher director of the Signal-Speech-Image–SIMPA Laboratory; USTOMB Oran, since 2002. His current research interests are in the area of speech and image processing, automatic speech recognition, neural networks, artificial immune systems, genetic algorithms, neuro-computing, machine learning, neuro-fuzzy logic, handwriting recognition, electronic/electrical engineering, signal and system engineering.