Scientific Programming 14 (2006) 231–250
IOS Press

An approach for the high-level specification of QoS-aware grid workflows considering location affinity

Ivona Brandic*, Sabri Pllana and Siegfried Benkner
Institute of Scientific Computing, University of Vienna, Nordbergstraße 15, 1090 Vienna, Austria

Abstract. Many important scientific and engineering problems may be solved by combining multiple applications in the form of a Grid workflow. We consider that for the wide acceptance of Grid technology it is important that the user has the possibility to express requirements on Quality of Service (QoS) at workflow specification time. However, most of the existing workflow languages lack constructs for QoS specification. In this paper we present an approach for high level workflow specification that considers a comprehensive set of QoS requirements. Besides performance related QoS, it includes economical, legal and security aspects. For instance, for security or legal reasons the user may express the location affinity regarding Grid resources on which certain workflow tasks may be executed. Our QoS-aware workflow system provides support for the whole workflow life cycle from specification to execution. Workflow is specified graphically, in an intuitive manner, based on a standard visual modeling language. A set of QoS-aware service-oriented components is provided for workflow planning to support automatic constraint-based service negotiation and workflow optimization. For reducing the complexity of workflow planning, we introduce a QoS-aware workflow reduction technique. We illustrate our approach with a real-world workflow for maxillo facial surgery simulation.

Keywords: Quality of Service (QoS), location affinity, workflow reduction, workflow planning, UML-based Grid workflow specification

1. Introduction

The emergence of Grid technology is strongly affecting the way in which information processing tasks are performed. Grid users may specify the tasks that should be performed at several levels of abstraction that directly reflect their role within an organization. At the top level of abstraction the user specifies the tasks that should be performed on the Grid as a *workflow*. This approach has the advantage that the user can map the problem from his/her domain of interest to a workflow in a straightforward manner. As a consequence, the user does not have to be an expert of Grid technology in order to specify the job that should be performed. Therefore, the workflow paradigm is considered to be very relevant for the wide acceptance of Grid technology.

Currently a significant research effort is invested in the development of workflow languages for Grid environments [42]. However, most of the existing workflows are specified in textual form, or are composed based on a self-defined graphical notation. Moreover, there is a lack of adequate tool support for workflow specification, and workflow specification tools are usually not well integrated with Grid environments. Furthermore, many workflow languages do not provide language constructs for QoS specification, or provide only limited QoS support (for instance, only for performance and economical related QoS). We believe that while performance is of paramount importance for time critical applications, the wide acceptance of Grid tech-

*Corresponding author: I. Brandic, Institute of Scientific Computing, University of Vienna, Nordbergstraße 15, 1090 Vienna, Austria. Tel.: +43 1 4277 39408; Fax: +43 1 4277 9394; E-mail: brandic@par.univie.ac.at.

ISSN 1058-9244/06/$17.00

nology depends on security and legal aspects as well. We have experienced that many potential users from industry hesitate to use Grid technology even if the performance and economical benefits are clear because of security and legal concerns.

There is a large number of application domains for Grid workflows, such as life sciences (for instance medical simulation services) and engineering (for instance vehicle development support), that demand a guarantee that workflow activities are performed within the specified *time*, *cost*, *security*, and *legal* constraints. Therefore, we are developing an XML based language for QoS-aware Grid workflows (QoWL), by extending the Business Process Execution Language (BPEL) [27] with language constructs for specification of QoS constraints [8]. A distinctive feature of QoWL is the ability to account for the user's preferences regarding the execution time and price of the activities as well as the execution *location affinity* for activities with specific security and legal constraints. The concept of *location affinity*, introduced in this paper, facilitates the realization of *Virtual Organizations* [16]. Furthermore, *location affinity* enables the accounting of security and legal QoS aspects at workflow specification time. In order to streamline the process of workflow specification, we have extended our graphical editor Teuta [35] for QoWL. With Teuta the user specifies the workflow graphically by using the Unified Modeling Language (UML) [30]. From the UML based workflow representation Teuta automatically generates the corresponding QoWL representation, which serves as input to the QoS-aware Grid Workflow Engine (QWE). QWE performs the necessary steps for the QoS-aware workflow negotiation and execution [9]. During the planning phase a set of QoS-aware service-oriented components is provided that supports automatic constraint-based service negotiation and workflow optimization. During the execution phase, using the information from the planning phase, workflow activities are executed in the manner that the specified requirements in terms of QoS constraints are met.

A prerequisite for QoS-aware workflow execution are QoS-aware services able to give QoS guarantees. A QoS-aware service enables clients to inquire and negotiate about its QoS properties. This kind of support is provided by Vienna Grid Environment (VGE) services [4]. VGE has been utilized within the European Commission funded GEMSS project which developed a testbed for six medical simulation and image reconstruction Grid services [18,3]. We evaluate our approach by specifying a QoS-aware Grid workflow for maxillo facial surgery simulation.

The main contributions of this paper include: (1) development of language support for specification of security and legal QoS constraints; (2) definition of a UML based Domain Specific Language (DSL) for QoS-aware Grid workflows; (3) extension of Teuta for QoWL, and integration of Teuta with QWE; (4) explanation of an approach for QoS-aware workflow reduction that simplifies the planning phase; and (5) explanation of basic mechanisms for QoS-aware planning, negotiation and execution.

The rest of this paper is organized as follows. Section 2 presents VGE services which are prerequisites for QoS-aware Grid workflow execution. Section 3 describes our approach for graphical specification of QoS-aware Grid workflows. Our implementation is outlined in Section 4. In addition Section 4 introduces our approach for QoS-aware Grid workflow reduction that streamlines the workflow planning process. Section 5 demonstrates the application of our approach by modeling a maxillo facial surgery simulation workflow. We compare and contrast the work presented in this paper with related work in Section 6. Section 7 presents our conclusions and describes future work.

2. Preliminaries: Vienna Grid Environment (VGE)

An important prerequisite for the development of QoS support for Grid workflows are QoS-aware Grid services, which are capable of providing service guarantees for the specified QoS requirements. The Vienna Grid Environment (VGE) [4] is a service-oriented Grid infrastructure for the provision of HPC applications as QoS-aware Grid services. VGE relies on standard Web Services technologies such as WSDL, SOAP, WS-Security, Tomcat and Axis. VGE services enclose native HPC applications, usually parallel MPI codes running on a cluster, and expose their functionality via a set of common operations for job execution, job monitoring, data staging and error recovery.

In addition, VGE services may be configured in order to offer QoS guarantees with respect to response time, price and location affinity. VGE services support a dynamic QoS negotiation model where clients may negotiate various QoS guarantees with multiple service providers [5]. VGE services rely on a generic QoS module, which usually comprises: an *application-specific performance model*, a *pricing model*, a *compute resource manager* that supports advance reservation, and a *QoS manager*. An application-specific per-

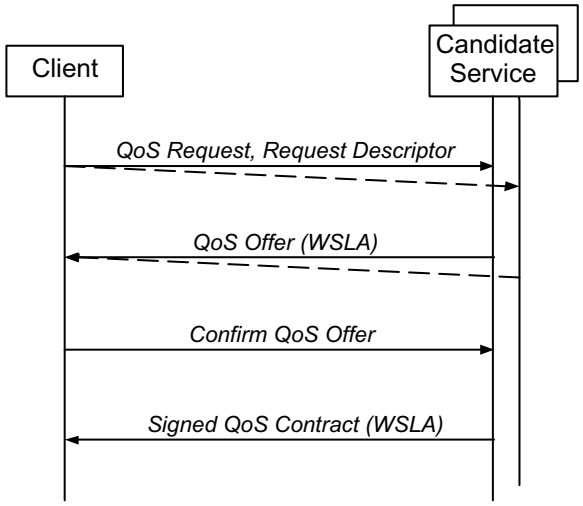


Fig. 1. QoS negotiation scenario.

formance model usually takes as input a *request descriptor*, containing input meta data, and a *machine descriptor* which specifies the amount of machine resources (e.g. number of processors, main memory, etc.) that may be provided for an application. Input *meta data* indicates the information that describes the input data of a service. Commonly, *meta data* is used for the performance evaluation of a service during the generation of QoS offers. *Payload data* denotes input and output data of a service operation.

The basic QoS negotiation scenario, as shown in Fig. 1, is based on a request/offer model, where a client requests offers from services. Upon start of a QoS negotiation, the client specifies the desired QoS constraints (e.g. earliest begin time of a job, latest finish time) within a *QoS request*. Furthermore, the client provides a *request descriptor* containing input meta data for a specific service request. For example, in the case of a finite element method (FEM) simulation, input meta data in the request descriptor would typically include the size of the finite-element model or the number of iterations to be performed.

On the service-side, the input meta data is fed into an application-specific performance model in order to obtain an estimate for the required execution time. In case of parallel MPI applications, the QoS manager, which controls the QoS negotiation on the service-side, uses heuristics to determine the number of processors required to execute a service request within the user-specified time constraints. The QoS manager then interacts with the compute resource manager to check whether a reservation of the required machine resources can be made at the required time. If QoS support for the price of a service request is required, the QoS manager subsequently invokes the pricing model to check whether the client's price constraints can be met. If all QoS constraints can be fulfilled, a temporary resource reservation is made and a corresponding QoS offer in the form of a Web Service Level Agreement (WSLA) [38] document is returned to the client (see Fig. 1). Only when the client, which usually negotiates with multiple service providers to get the best deal, confirms an offer, a signed QoS contract in the form of a WSLA is established. Temporary reservations for offers that are not confirmed by the client expire within a short time frame.

When a client has successfully negotiated the required QoS with a service provider and a QoS contract is in place, the usual job execution phase can be entered, which comprises the invocation of service operations for *upload*ing the input data, for *start*ing the job execution and for *downloading* the result. In order to support direct data transfer between services, corresponding *push* and *pull* operations are supported as well. The generic QoS module of VGE enables the provision of parallel applications as dynamically configurable Grid services. Depending on the requirements of a client, an application may be executed on many processors in short time but for a higher price, or it may be executed on a few processors with a lower price.

The VGE service provision model and QoS support have been successfully applied within the European GEMSS project for the development of six medical Grid applications [22] which utilize computationally demanding methods such as parallel FEM simulation, parallel Computational Fluid Dynamics and parallel Monte Carlo simulation, realized as remote Grid services running on clusters or other parallel computing platforms.

3. High-level specification of QoS-aware workflows

In this section we first briefly describe the QoWL language, which is an XML based language for QoS-aware Grid workflows. Thereafter, we present a UML-based graphical representation of QoWL.

3.1. Quality of service aware grid workflow language (QoWL)

QoWL comprises a subset of the Business Process Execution Language (BPEL) [27] and a set of QoS extensions that is used for specification of the QoS re-

quirements of Grid workflows considering begin time, response time, price and location affinity. The elements of our BPEL subset include: *Process*, *Invoke*, *Copy*, *Sequence*, *Flow*, *Receive*, *Reply*, *Switch*, and *While*. QoWL elements are used for the specification of QoS before QoS negotiation and for the expression of QoS after negotiation.

Figure 2 depicts the structure of a QoWL element. A QoWL element is defined as a BPEL element extended with a set of QoS constraints. The attributes of the `qowl-element`, such as `name` and `portType`, are used as defined in the BPEL specification [27]. The `<qos-constraints>` element specifies the QoS constraints of a specific workflow element. The attribute `reqDescVar` defines the variable which specifies the input meta data. Each `<qos-constraints>` element may contain several `<qos-constraint>` elements. Each element of type `<qos-constraint>` specifies a QoS constraint as a tuple (*name, value, weight*).

QoS extensions are used to express both the *requested* QoS constraints of a workflow before the QoS negotiation (see Fig. 3a) and the *offered* QoS of a workflow after the negotiation with the services (see Fig. 3b). Please note that offered QoS may differ from the requested one.

3.1.1. Specification of QoS constraints

Figure 3 depicts the abstract workflow before the QoS negotiation and the concrete workflow after the QoS negotiation.

Figure 3(a) depicts a fragment of an *abstract* workflow that represents an `<invoke>` activity. The `<qos-constraint>` element named `beginTime` specifies the earliest begin time of the workflow execution. The budget for the activity execution is specified defining the `price <qos-constraint>` element. The `weight` attribute may be used to balance the effect of different QoS constraints on the service selection process. The weight attribute is specified for the `beginTime`, `endTime` and the `price` elements. The `<candidate-registry>` element specifies where potential services may be found. The `<candidate-registry>` element may be specified by the user or automatically mapped from the workflow engine as predefined option. Generally, candidate registries comprise a number of dynamically registered services. Keywords may be used in order to find matching services.

The `<invoke>` element `start` depicted in Fig. 3(a) comprises the set of constraints defined within the `<qos-constraints>` element (see lines 4–16 in Fig. 3(a)). The `<qos-constraints>` element contains one `<candidate-registry>` element (see line 5 in Fig. 3(a)). The meta data necessary for QoS negotiation of that activity is specified using the `startReqDesc` variable. The payload data is set using the `startRequest` variable.

Additionally, the user may express preferences regarding the location of Grid resources where an activity should be executed, by specifying the Grid site, organizational, or geographical affinity. For instance, the QoS constraint `geographicAffinity` with value `at` specifies that the activity should be executed on Grid resources that are geographically located in Austria (see lines 14–15 in Fig. 3(a)).

Figure 3(b) depicts a corresponding *concrete workflow*. Now, in the concrete workflow, the `<invoke>` element `start` contains instead of the `<candidate-registry>` element a `wsdl` attribute with the endpoint of the selected service (see line 3 in Fig. 3(b)). The `<qos-constraints>` element now also contains the `wslaVar` variable specifying the Service Level Agreement between the engine and the particular service. The `<qos-constraints>` element of the `invoke` activity considers the offered QoS of a VGE service.

3.1.2. Specification of location affinity with QoWL

Most of the existing related work focuses on performance (i.e. activity execution time) and economical aspects (i.e. activity price) of QoS [41,43]. We believe that while performance is of paramount importance for time critical applications, the wide acceptance of Grid technology strongly depends on security and legal aspects. We have experienced that many potential users from industry hesitate to use Grid technology even if the performance and economical benefits are clear because of security and legal concerns [20]. Therefore, we consider that it would be useful if the user has the possibility to restrict the location of Grid resources on which certain activities may be executed. For instance, for security or legal reasons the user may specify that an activity should be executed only on Grid resources that belong to the user's organization.

Figure 4 depicts how location affinity can be expressed with QoWL. The user may specify that a certain workflow activity should be executed on a specific *Grid site*, on the Grid resources of a specific *organization*, or on the Grid resources of a specific *geographical region*.

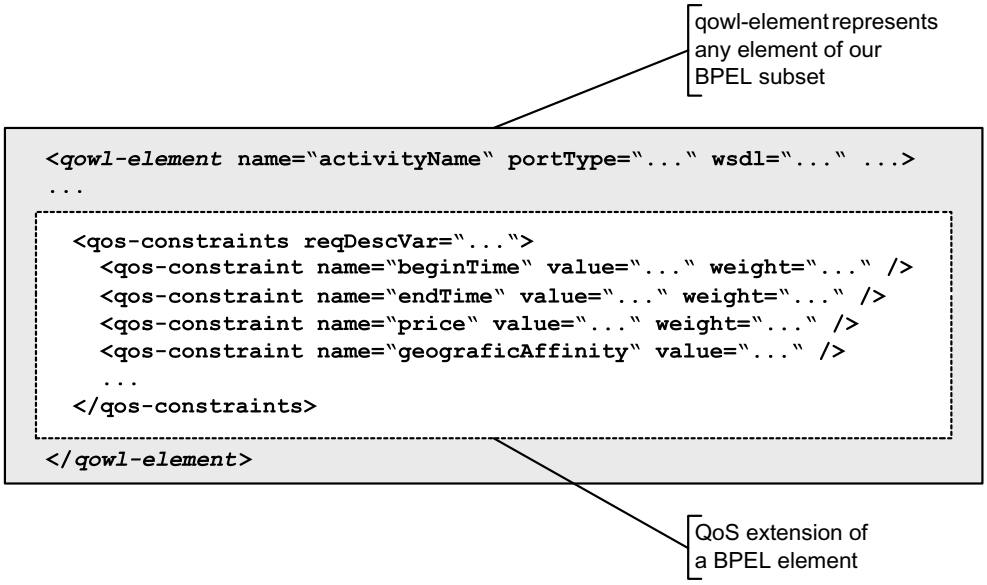


Fig. 2. The structure of a QoWL element.

```
 1. ...
 2. <invoke name="start" portType="appex"
 3.  operation="start" inputVar="startRequest">
 4.  <qos-constraints reqDescVar="startReqDesc">
 5.    <candidate-registry inputVar="queryRequest"
 6.       ...
 7.      wsdl="http://kim:9357/registry/reg?wsdl"/>
 8.    <qos-constraint name="beginTime" weight="0.3"
 9.       value="18-08-2005 12:00:00,0 MET" />
10.      <qos-constraint name="endTime" weight="0.2"
11.        value="18-08-2005 14:00:00,0 MET" />
12.      <qos-constraint name="price" weight="0.5"
13.        value="20.00" />
14.      <qos-constraint name="geographicAffinity"
15.        value="at" />
16.  </qos-constraints>
17. </invoke>
18.  ...
```

(a)

```
 1. ...
 2. <invoke name="start" portType="appex"
 3.   wsdl="http://bridge:9355/SPECT/appex?wsdl"
 4.   operation="start" inputVar="startRequest">
 5.  <qos-constraints reqDescVar="startReqDesc"
 6.     wslaVar="wslaStart">
 7.    <qos-constraint name="beginTime" weight="0.3"
 8.      value="18-08-2005 12:13:06,0 MET" />
 9.    <qos-constraint name="endTime" weight="0.2"
10.      value="18-08-2005 13:45:04,0 MET" />
11.    <qos-constraint name="price" weight="0.5"
12.      value="16.00" />
13.    <qos-constraint name="geographicAffinity"
14.      value="at" />
15.  </qos-constraints>
16. </invoke>
17. ...
```

(b)

Fig. 3. Examples of abstract and concrete QoWL workflows. (a) An excerpt of an abstract QoWL workflow. (b) An excerpt of a concrete QoWL workflow.

Commonly *Grid site affinity* is not specified by the user, but the Grid environment automatically maps workflow activities to Grid resources based on the availability and performance of resources [10]. Usually, the goal is to minimize workflow execution time. But, in the case that the user has the information (related to security or law) which can not be automatically obtained by the Grid environment, then he can manually map the activity to a specific Grid site. Such examples are medical applications with legal restriction considering electronic transfer of patient specific data [17]. Grid site preference is specified by using the QoS constraint named `gridSiteAffinity`. Figure 4 shows that activities `A3` and `A5` encompassed by group `G1` should be on the same Grid site. The reason could be the large data transfer between the activities `A3` and `A5` or some security reasons. The QoWL code of `A3`, `A7` and `A11` depicts the specification of the affinity on the language level (see Fig. 4).

Organization affinity indicates the preference of the user regarding the location where an activity should be executed considering resources that belong to a specific organization. These resources can be geographically distributed. The user's preferences may be based on

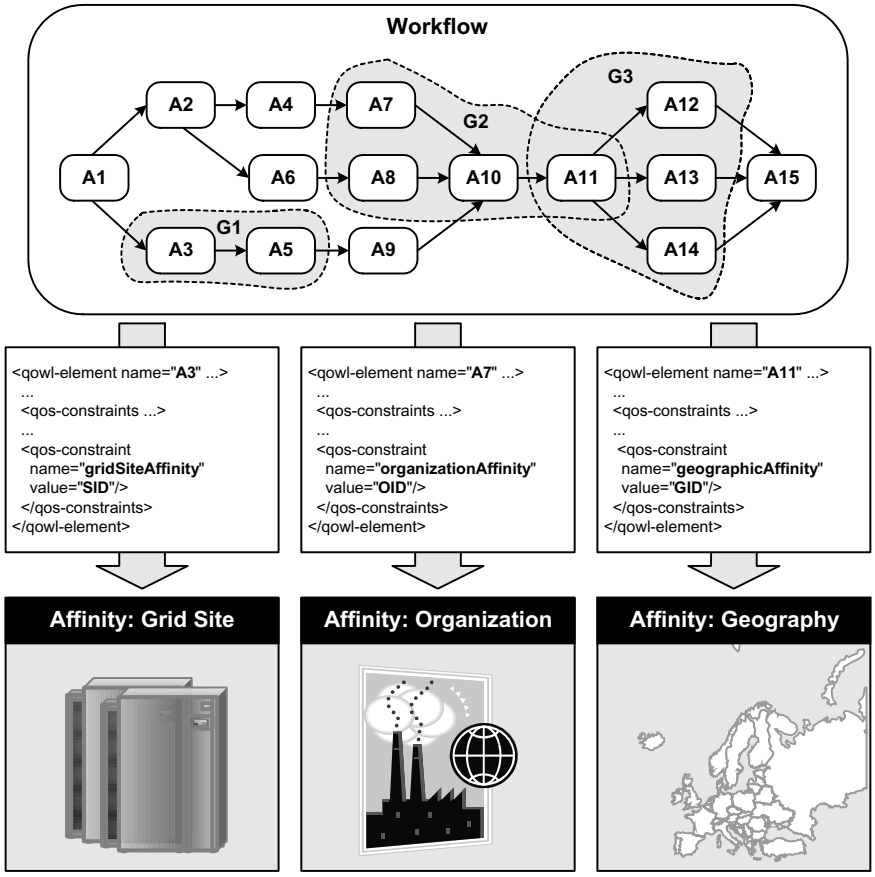


Fig. 4. Specification of location affinity with QoWL.

established trust relationships with other companies. For instance, a vehicle producing company may wish to execute certain critical activities on a subset of the Grid in order to ensure that any relevant information is not visible for competitors. Organization preference is specified by using the QoS constraint named `organizationAffinity`. Figure 4 depicts that activities `A7`, `A8`, `A10` and `A11` encompassed by group `G2` should be executed on resources that belong to the same organization.

Geographical affinity indicates the preference of the user regarding the location of activity execution on Grid resources that belong to a specific geographical region. Examples of geographical region include: country, state, or set of states. For instance several countries which have the same legal conditions for the electronic transfer of medical data may be eligible for execution of certain activities. Geographical preference is specified by using the QoS constraint named `geographicAffinity`. Figure 4 shows that activities `A11`, `A12`, `A13` and `A14` encompassed by group `G3` should be executed on resources that belong to the same geographic region.

The agreement on *time* and *cost* constraints as well as *location affinity* constraints requires a negotiation process with the candidate services as described in Section 4.4.3. The specified location affinity may be integrated with the available security infrastructure, such as *Web Services Policy* [39]. The security and legal related QoS can be used to support the concept of *Virtual Organization* (*VO*). The concept of VO defines policies governing access to resources of the organizations (such as unique authentication, authorization) which form a specific VO [16]. *Location Affinity* can be used to determine which of the available resources of a VO best satisfies legal and security requirements of a specific application (e.g. medical application). In addition, the concept of location affinity permits the user to use selectively Grid resources of several VO's. Therefore, the concept of *location affinity* can be used supplementary to the concepts of VO. The *location affinities* may be mapped to the security credentials of the particular VO.

In the following section we describe the UML-based modeling of QoWL elements.

3.2. The definition of a UML-based DSL for QoS-aware workflows

The UML 2.0 specification [30] provides a large set of modeling elements and diagrams for modeling various types of software and hardware systems. UML has a modular nature, with the *diagram type* being the unit of modularity. From the available 13 UML diagram types, we use only *activity diagrams* for modeling Grid workflows. UML activity diagrams are suitable for *flow modeling* of various types of software or hardware systems. Hierarchical capabilities of UML activity diagrams support modeling of systems at arbitrary levels of detail and complexity. For instance, it is possible to group a set of activities with the corresponding flow into a higher-level activity with a well defined input and output.

In order to enable the modeling of different types of systems, the UML modeling elements are specified in an abstract manner without conceptual connection with a particular domain. However, too generic semantics of UML modeling elements may present an obstacle for using UML in a specific domain. For this reason, the UML specification defines the mechanisms for specializing semantics of modeling elements for a particular domain. We have defined a *Domain Specific Language* (*DSL*) for QoS-aware Grid workflows by using the UML extension mechanisms. The UML may be extended by defining new modeling elements, *stereotypes*, based on existing elements, *base classes* (i.e. metaclasses). A stereotype is defined as a subclass of an existing UML metaclass, with the associated *tagged values* (i.e. meta attributes). Stereotypes are denoted by the stereotype name enclosed in guillemets <<StereotypeName>>, or by a specific graphic icon.

The benefits of definition of a DSL for the domain of QoS-aware Grid workflows include: (1) the user is exposed to only domain-relevant UML modeling elements, (2) the language concepts have domain-specific interpretation, and (3) models may be enriched with information that is used by tools for automatic model transformation (for instance to XML) or model processing (for instance for the purpose of QoS negotiation).

For each element of our XML-based language QoWL we have defined an element of UML-based DSL. Fig. 5 depicts an example for defining elements of our DSL for the domain of QoS-aware Grid workflows. For the *Invoke* QoWL element depicted in Fig. 5(a) the corresponding DSL element is defined by stereotyping the UML metaclass *Action* (see Fig. 5(b)). The *Invoke* activity is used for the invocation of services. The tagged values *portType*, *operation*, *inputVar* and *outputVar* may be used for specification of the information that is needed for service invocation. The graphical notation of stereotype *Invoke* is illustrated with an example in Fig. 5(c).

Figure 6 depicts the structure of type *QTypes*, whose instances are used to specify the tagged value *qosConstraints*. *QTypes* contains the *reqDescVar* attribute, which specifies the meta data that may be used for performance prediction of the service. Additionally, *QTypes* comprises zero or more entities of type *QType* that are used for description of specific QoS constraints (e.g. required execution time). Moreover, *QTypes* comprises zero or more entities of type *RegType* that may be used for the specification of registries where potential services can be found.

The rest of the elements of our UML-based DSL are defined in an analogous manner. Figure 7 depicts the complete list of modeling elements of our DSL for QoS-aware Grid workflows. The first column shows the names of newly defined UML modeling elements (such as *Process*). The second column shows the UML elements that serve as base classes for customization (for instance, *Activity*). Tagged values are shown in the third column. The fourth column provides the description of DSL elements.

3.3. Graphical representation of QoWL elements with the UML-based DSL

Basic elements of QoWL are not further decomposed into other elements. QoWL elements of this kind are: *Invoke*, *Copy*, *Receive*, and *Reply*. The graphical representation of the element *Invoke* is depicted in Fig. 5. Other basic elements of QoWL are represented with the UML-based DSL in an analogous manner.

Complex elements of QoWL may comprise basic and complex elements. According to annotation strategies defined in [9] the QoS of a workflow may be specified (i) locally for basic activities, (ii) globally for the overall workflow or (iii) locally for the critical basic or complex activities. QoWL complex elements are: *Sequence*, *Flow*, *Switch*, *While*, and *Process*. The modeling of QoWL complex elements with the UML-based DSL is described in the following.

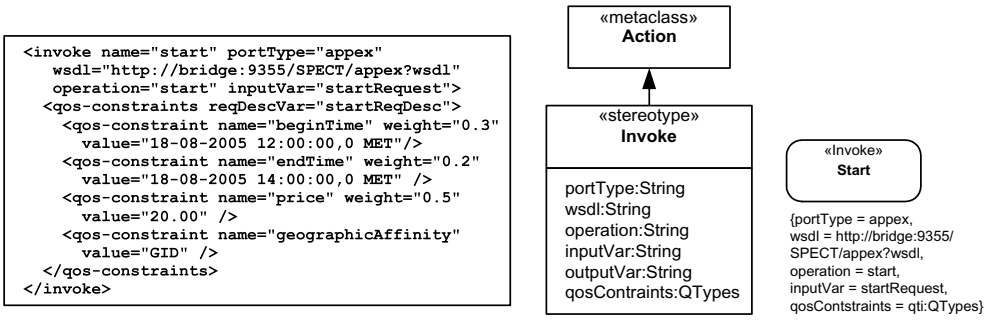


Fig. 5. Stereotype *Invoke*. (a) QoWL. (b) Definition. (c) Usage.

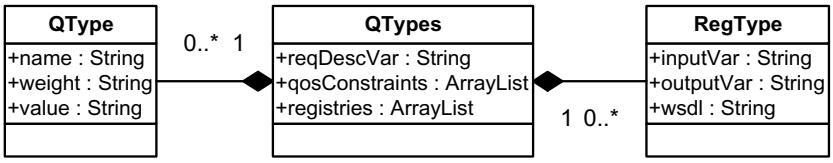


Fig. 6. The structure of *QTypes*.

3.3.1. The Sequence element

The *Sequence* element specifies a set of activities that should be executed sequentially in the predefined order. Usually, the data dependency determines the execution order of activities within the sequence.

Figure 8(a) depicts an instance of the *Sequence* element. The *qti:QTypes* attribute defines the QoS constraints. Figure 8(b) shows the comprised activities within the *SampleSequence* element. Figure 8(c) depicts an object *qti* of type *QTypes*.

QoS information: The user may specify the QoS constraints for the *Sequence* element. Figure 8(c) depicts an example of the *qti* object that comprises four constraints of type *QType* namely $c1, \ldots, c4$. The $c1$ constraint defines the earliest possible begin time of the execution of the *SampleSequence*. $c2$ defines the latest possible end time of the execution of the *SampleSequence*. $c3$ defines the maximum price whereas the *geographicAffinity* constraint specified within $c4$ defines that all comprised activities of the *SampleSequence* should be located in Middle Europe. The execution of comprised activities should comply with the QoS constraints of *Sequence* as follows,

- $\sum_{i=1}^{n} time(A_i)$ should not exceed the specified time for *Sequence*,
- $\sum_{i=1}^{n} price(A_i)$ should not exceed the specified price for *Sequence*,
- the *location affinity* is inherited by all comprised activities.

Let A_i denote the i^{th} activity in the *Sequence* and n the number of comprised activities within the *Sequence*. The execution time of an activity (A_i) can be calculated as follows,

$$time(A_i) = endTime(A_i) - beginTime(A_i). \quad (1)$$

The *geographicAffinity* is defined in the down-right part of Fig. 8(c) (see object $c4$). Location affinity may be defined as set of values. For example if *MiddleEurope* is defined as follows

$$MiddleEurope = \{Austria,\ Czech\ Republic,\ Germany,\ Poland,\ Slovakia,\ Switzerland\} \quad (2)$$

the selected services may be located in any of countries defined in Eq. (2). In the similar way the multiple values for the `gridSiteAffinity` and for the `organizationalAffinity` may be specified.

In the following Sections 3.3.2 – 3.3.5 we use the object *qti:QTypes* for the association of QoS constraints to workflow elements. An example of the content of object *qti:QTypes* is depicted in Fig. 8(c), and therefore, we do not repeat it for the remaining workflow elements. In the remaining part of this document, it is presumed that $time(A_i)$ is calculated using the Eq. (1).

3.3.2. The Flow element

The *Flow* element specifies that a set of activities may be executed concurrently.

Stereotype	Base Class	Tags	Description
Process «Process»	Activity	qosConstraints:QType, variables:VType	Indicates that Activity represents a workflow process
Invoke «Invoke»	Action	qosConstraints:QType, portType:String, operation:String, inputVar:String, outputVar:String, wsdl:String	Indicates that Action represents the operation invocation of an external Grid Service
Copy «Copy»	Action	From:FromType, To:ToType	Indicates that Action represents the data value assignment
Sequence «Sequence»	SequenceNode	qosConstraints:QType	Indicates that SequenceNode represents a series of actions which are executed sequentially
Flow «Flow»	StructuredActivityNode	qosConstraints:QType	Indicates that StructuredActivityNode represents a set of actions which may be executed concurrently
Receive «Receive»	AcceptEventAction	portType:String, operation:String, variable:String, wsdl:String	Indicates that AcceptEventAction represents a blocking message receive
Reply «Reply»	SendSignalAction	portType:String, operation:String, variable:String, wsdl:String	Indicates that SendSignalAction represents the reply message to a message that was received through a «Receive»
Switch «Switch»	DecisionNode	qosConstraints:QType	Indicates that DecisionNode represents the conditional execution
While «While»	LoopNode	condition:Boolean	Indicates that LoopNode represents a while loop. The loop body is executed until the condition is violated.

Fig. 7. Elements of the UML-based DSL for QoS-aware workflows.

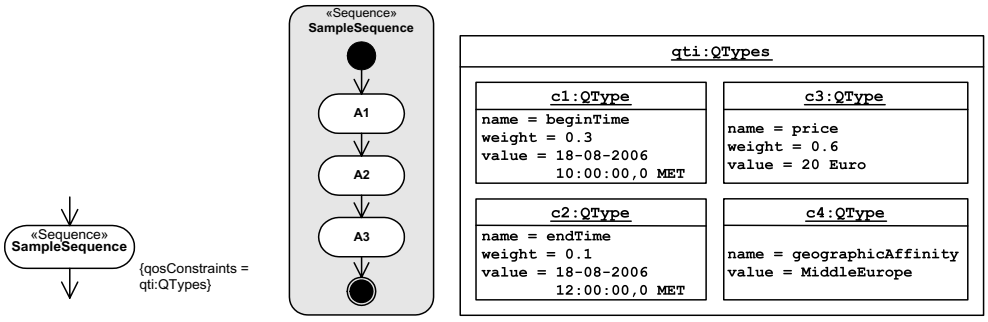


Fig. 8. QoWL element *Sequence*.

Figure 9(a) depicts an instance of the *Flow* element. The *qti:QTypes* attribute defines the QoS constraints. Figure 9(b) shows the comprised activities, A_1 and A_2, within the *Flow* element that should be executed concurrently.

QoS information: The user may specify the QoS

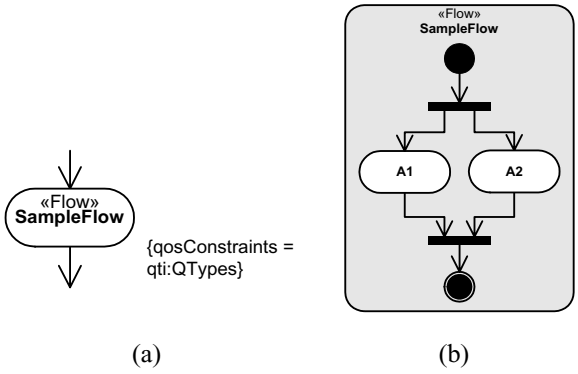


Fig. 9. QoWL element *Flow*.

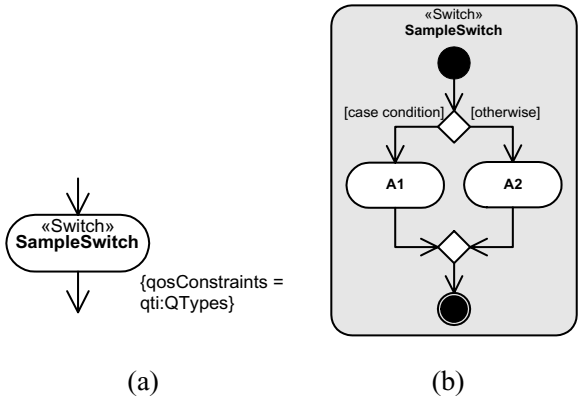


Fig. 10. QoWL element *Switch*.

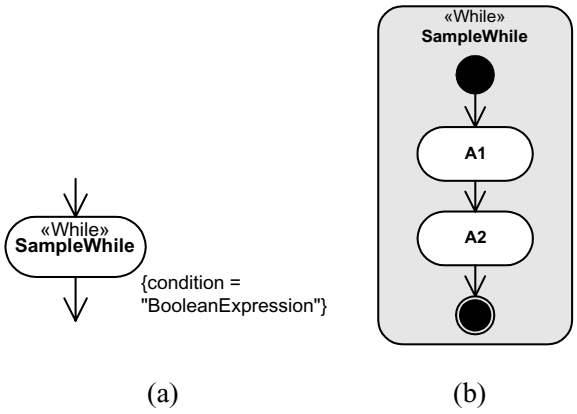


Fig. 11. QoWL element *While*.

constraints for the *Flow* element. The execution of the comprised activities should comply with the QoS constraints of *Flow* as follows,

- *Max*$\{$*time*$(A_i)|i = 1, \ldots, n\}$ should not exceed the specified time for *Flow*,
- $\sum_{i=1}^{n}$ *price*(A_i) should not exceed the specified price for *Flow*,
- the *location affinity* is inherited by all comprised activities,

where n is the number of comprised activities within the *Flow*, A_i is the i^{th} activity of the *Flow*, and *time*(A_i) is calculated as defined in Eq. (1).

3.3.3. The Switch element

The *Switch* element specifies that one of the alternate execution paths is selected based on a condition. The condition is specified as a Boolean expression.

Figure 10(a) depicts an instance of the *Switch* element. The *qti:QTypes* attribute defines the QoS constraints. Figure 10(b) shows the comprised activities, A_1 and A_2, within the *Flow* element. If the Boolean *case condition* evaluates to *true*, then activity A_1 is executed, otherwise A_2.

QoS information: The user may specify the QoS constraints for the *Switch* element. The specified QoS constraints have to be satisfied for each possible execution path. The execution of the comprised activities should comply with the QoS constraints of *Switch* as follows,

- *Max*$\{$*time*$(A_i)|i = 1, \ldots, k\}$ should not exceed the specified time for *Switch*,
- *Max*$\{$*price*$(A_i)|i = 1, \ldots, k\}$ should not exceed the specified price for *Switch*,
- the *location affinity* is inherited by all comprised activities,

where k is the number of comprised execution paths within the *Switch* element, A_i is the i^{th} branch of the *Switch*, and $time(A_i)$ is calculated as defined in Eq. (1).

3.3.4. The While element

The *While* element specifies the iterative execution of the comprised activities as long as the specified boolean expression evaluates to true.

Figure 11(a) depicts an instance of the *while* element. The *condition* attribute specifies the Boolean expression which is evaluated before each iteration. Figure 11(b) shows the comprised activities, A_1 and A_2, within the *While* element.

QoS information: Because in the general case it is difficult to determine exactly the number of iterations in advance, commonly the performance related QoS for the *While* element is not specified. However, based on historical data (that is data obtained from previous executions) it is possible to predict the number of iterations for specific cases. Also usage of probabilistic

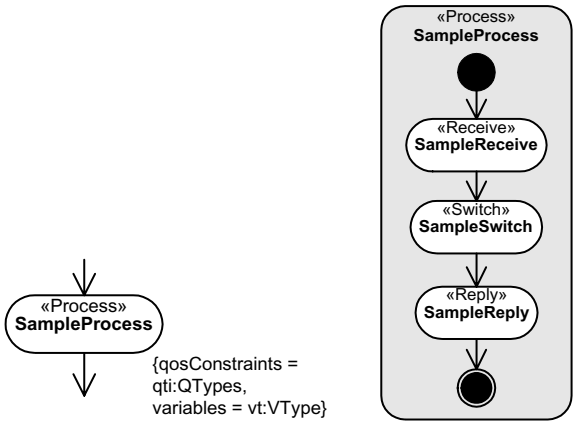


Fig. 12. QoWL element *Process*.

models for the prediction of the number of iterations is conceivable. Therefore, the performance related QoS may be specified but in the general case can not be guaranteed. The QoS constraints that are related to location affinity are specified for the *While* element in the same manner as for other workflow elements.

3.3.5. The Process element

The *Process* element specifies the overall workflow. It comprises all other elements of the workflow.

Figure 12(a) depicts an instance of the *Process* element. The *qti:QTypes* attribute defines the QoS constraints. The *vt:VType* defines the global variables of the workflow. Figure 12(b) shows the comprised activities within the *Process* element.

QoS information: The user may specify the QoS constraints for the *Process* element. The execution of the comprised activities should comply with the QoS constraints of *Process*. Generally, process element comprises one complex element as a root element (e.g. Sequence). The root element may comprise other basic or complex elements.

4. QoS-aware workflow system

In this section we describe our system that provides support for the whole workflow life cycle from specification to execution.

4.1. Architectural overview

Figure 13 shows the architecture of our system for QoS-aware Grid workflows. The main components include: (1) Teuta, which is a UML based graphical editor for workflow specification; (2) QWE, which is a QoS-aware workflow engine; and (3) VGE services, which are QoS-aware Grid services.

A user may specify the workflow with Teuta by composing the predefined elements of UML-based DSL for QoS-aware workflows (see Section 3.3). Furthermore, for each workflow element different parameters (such as execution time, price, location affinity) may be specified that determine the user's QoS requirements. Thereafter, Teuta verifies whether the specified workflow is well defined. In the case that the workflow model is well defined, Teuta generates the corresponding QoWL representation. The QWE engine interprets the QoWL workflow, negotiates with available services, applies the selected workflow planning strategy, selects appropriate services and finally executes the specified workflow. If the specified tasks need QoS guarantees we use VGE services, which are able to give certain QoS guarantees. In other cases (for instance in case that execution time of the service is negligible) the use of other non-VGE services may be considered. In the following the main architectural components are explained in more detail.

4.2. Teuta

Teuta is a UML-based graphical editor. It is designed as a platform independent, configurable and extensible tool. Therefore, it is possible to extend Teuta with new types of diagrams and modeling elements for various domains. Examples of usage of Teuta include performance modeling of high performance programs [35] and specification of scientific workflows within the framework of Askalon project [2]. In order to provide tool-support for our approach described in this paper we have extended Teuta for QoS-aware workflows and integrated with QWE.

The Teuta architecture is shown on the left-hand side of Fig. 13. Teuta comprises three main components: Graphical User Interface (GUI), Model Checker, and Model Traverser. We illustrate the GUI of Teuta with examples of real-world workflows in Section 5.

The *Model Checker* verifies whether the model is well defined. The rules for model checking are specified by using our XML-based Model Checking Language (MCL). The model checker gets the model de-

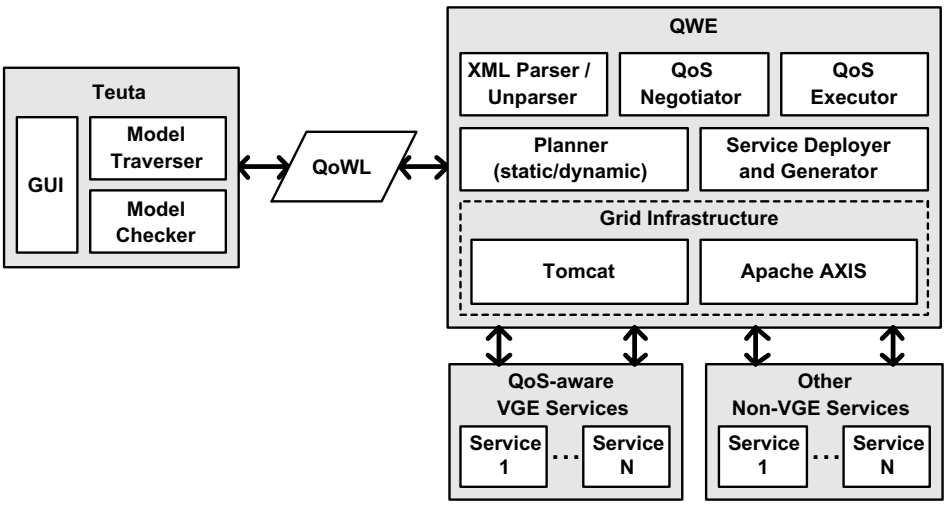


Fig. 13. Architecture of the system for QoS-aware Grid workflows.

scription from an MCL file. This MCL file contains a list of available diagrams, modeling elements and the set of rules that defines how the elements may be interconnected.

The *Model Traverser* provides the possibility to walk through the model, to visit each modeling element, and to access its properties (for instance QoS constraints). We use model traversing for the generation of various model representations; for instance, a QoWL representation serves as input for QWE engine (see Fig. 13).

4.3. QWE

The QoWL documents generated by Teuta can be executed using the QoS-aware Grid Workflow Execution Engine (QWE). In this section we briefly describe the main components of QWE. A more comprehensive description of QWE can be found in [9].

QWE is depicted on the right-hand side of Fig. 13. The main parts of the QWE engine are: (1) *the XML parser and unparser* which generates the intermediary representation of the QoWL workflow; (2) the *QoS Negotiator* queries the registries, generates necessary QoS requests and receives offers from services; (3) the *Planner* component calculates a workflow execution plan considering the selected workflow planning strategy (*static or dynamic*) and the selected workflow planning technique (*Integer Programming, Genetic Algorithm, MCDM, etc.*); (4) the *Service Deployer and Generator* exposes a QoWL workflow as a Web Service and the *QoS Executor* starts the execution of the QoWL workflow.

4.4. Planning

The workflows, which are specified using Teuta, are transformed into a concrete workflow using the *Workflow Planner* component. The aim of this component is to automate the selection of services in accordance with the *requested QoS*. *Workflow planning* comprises the following phases: (1) selection of the *workflow optimization strategy*, (2) *QoS-aware workflow reduction*, (3) *negotiation* and (4) the *workflow optimization*.

4.4.1. Workflow optimization strategy selection

We distinguish between *static* and *dynamic* workflow planning strategies. The decision whether static or dynamic planning techniques should be used, depends on the *meta data* of the invoked services. If all *meta data* required for QoS prediction is known before the workflow execution, the static planning strategy can be selected. If the *meta data* is generated or changed during workflow execution, the dynamic planning approach has to be used. Static planning implies the generation of the concrete workflow before the execution of the first workflow activity. In the case of the dynamic planning approach the concrete parts of the workflow are created for the ready-to-start activities during the workflow execution. The *Meta Data Flow Analyzer* (*MDFA*) verifies whether the static planning approach is feasible. We distinguish two types of variables: (1) the *payload variables* (*PV*), such as input data of the invoked services, and (2) the *meta data variables* (*MDV*), that describe the service input data (for instance the size of the input file, matrix size, etc.). The *MDFA* checks whether any *MDV* appears as output of any *invoke*, *re-*

ceive or *copy* activity. In this case, only the dynamic planning approach can be used.

4.4.2. Workflow reduction

The planning for real-world workflows is an NP-hard problem. But, the task of workflow planning may be alleviated by reducing the complexity of the workflow. Commonly, Grid workflows are composed of several activities where some of them are *time intensive*, *cost intensive* or *security relevant*. Such activities determine the QoS of the overall workflow. Other kinds of activities (e.g. control tasks) that do not significantly affect the overall QoS may be neglected during the optimization phase of the workflow planning. Workflow reduction can be done in several ways either manually or automatically. In the first case the user can manually select services which should be considered for the workflow planning. In the second case the resource consuming activities may be detected by considering all activities which have associated QoS. In the third case, if for example QoS is assigned only on the global level, a trial workflow run may be done in order to figure out which of the specified resources offer/demand QoS. Thus, resource intensive activities may be detected.

Figure 14(a) depicts the Maxillo Facial Surgery Simulation (MFSS) workflow [11]. We provide a detailed description of MFSS in Section 5. Here we use the MFSS workflow for the illustration of our QoS-aware workflow reduction method. The process of elimination of activities without QoS constraints from the planning process is called *QoS-aware workflow reduction*. Workflow reduction is done after workflow specification and before QoS negotiation (see Section 4.4.3). In the case of MFSS the workflow reduction is performed by eliminating all activities which do not have specified QoS constraints. Figure 14(b) represents the reduced QoS model of the MFSS workflow. The QoS model may be specified for: (1) a single activity (e.g. *UploadOperation*) which can be basic or complex, (2) the overall reduced workflow (e.g. activities *UploadOperation*, *StartOperation* and *DownloadOperation*). All aforementioned activities belong to the complex activity *FEMSequence*. The QoS model considers the selected optimization strategy and the aggregation function.

4.4.3. QoS negotiation

Based on the selected *optimization strategy* the *WF Negotiator* queries the specified registries, generates necessary *requested QoS* and receives *offered QoS* from services. Figure 15 depicts the negotiation process and participating components. The *WF Planner* starts the negotiation by initializing one or more instances of *WF Negotiator*. Each instance is responsible for the negotiation process of one activity. In the case of static planning, negotiation starts concurrently for all activities of the reduced QoS-aware model.

After the initialization of *WF Negotiator*s, each *WF Negotiator* supplies each candidate service with a *QoSRequest* (QoSReq) and a *RequestDescriptor* (ReqDesc). A *QoSReq* contains requested QoS, whereas a *ReqDesc* contains meta data about input data necessary for the evaluation of QoS constraints. If the requested QoS is not specified, the WF Negotiator supplies an empty *QoSReq* and the service responses with *offered QoS* that is not optimized for any QoS constraint (i.e. for time or price) (see Fig. 15). If the *QoSReq* is specified, each service tries to meet the specified constraints. For instance if a low price is requested, a service may run the application on fewer nodes to meet the price constraint. After collecting all offers each *WF Negotiator* notifies the *WF Planner* about received offers. Thereafter, the *WF Planner* selects appropriate services which fit into global or local constraints by applying the selected optimization strategy. The selected services are confirmed and the WSLA is generated between services and the QWE.

4.4.4. Optimization

For the *static planning approach* we use *lp_solve*, which is a mixed-integer linear programming (lp) package [24]. *lp_solve* is suitable for solution of global optimization problems. It can be applied to the planning of the whole workflow, or to the planning of complex activities. The optimization process with *lp_solve* involves the association of an *ActivityID* with each activity of the reduced workflow, and a *serviceID* with each candidate service of an activity. Each QoS-constraint (e.g. maximum time, maximum price) of an activity represents an *integer programming* (*IP*) *constraint*. The outcome of the optimization process with *lp_solve* is an array that contains the IDs of the selected services. The order of service IDs within this array corresponds to the execution order of activities of the reduced workflow. The interested reader may find a theoretical elaboration of the IP approach in [9]. Since most of our reduced workflows have the form of a simple straight-line code, IP is a more appropriate approach than complex heuristic techniques (e.g. genetic algorithms [25]).

For the *dynamic planning approach* we use the Multiple Criteria Decision Making (*MCDM*) approach where we select local optima for each activity sepa-

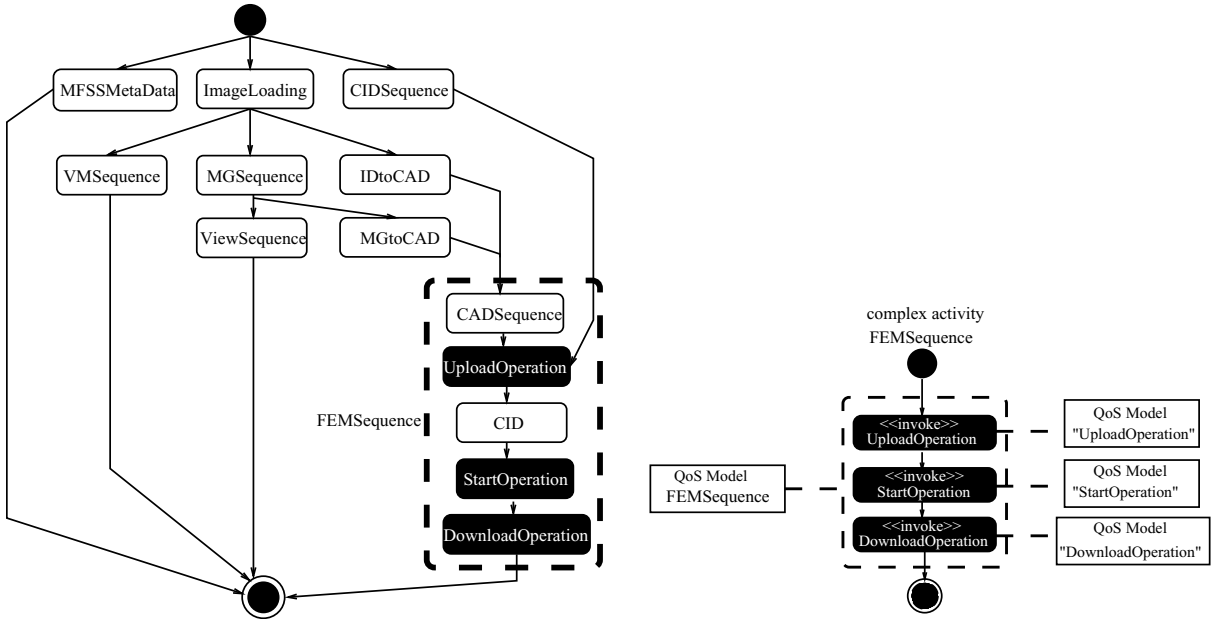


Fig. 14. QoS-aware Grid workflow reduction. (a) MFSS workflow. (b) Reduced form of the MFSS workflow with QoS models of the activities.

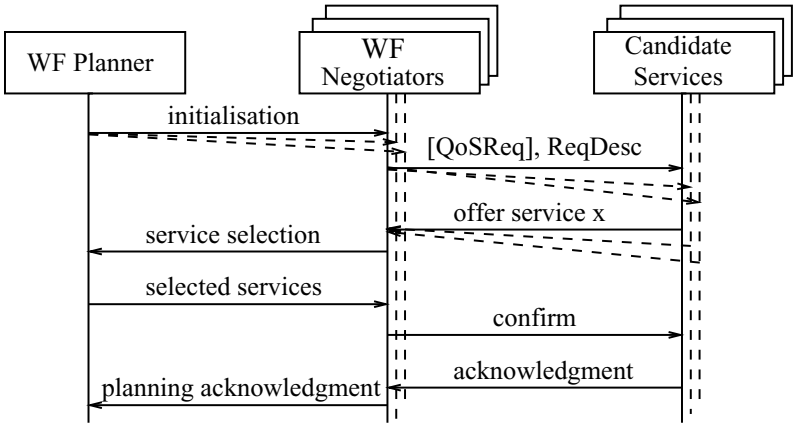


Fig. 15. WF Negotiation process.

rately. In case of the dynamic approach, planning, negotiation and execution processes are invoked in an iterative fashion.

4.5. Execution

The outcome of the workflow planning phase is a concrete workflow which is ready for execution. In the case of static planning approach the workflow execution phase is started after the completion of workflow planning phase. But, in the case of dynamic planning the execution and planning phases are performed for each activity of the workflow in an alternate fashion. Figure 16 illustrates the relationship of workflow planning and execution phases.

Figure 16(a) depicts the relationship between *static workflow planning* and *workflow execution*. Static workflow planning involves the following steps: *Strategy Selection*, *WF Reduction*, *WF Negotiation* and *WF Optimization*. As we described in Section 4.4, for the static planning approach global constraints (GC), such as maximum price of the workflow, are specified for the

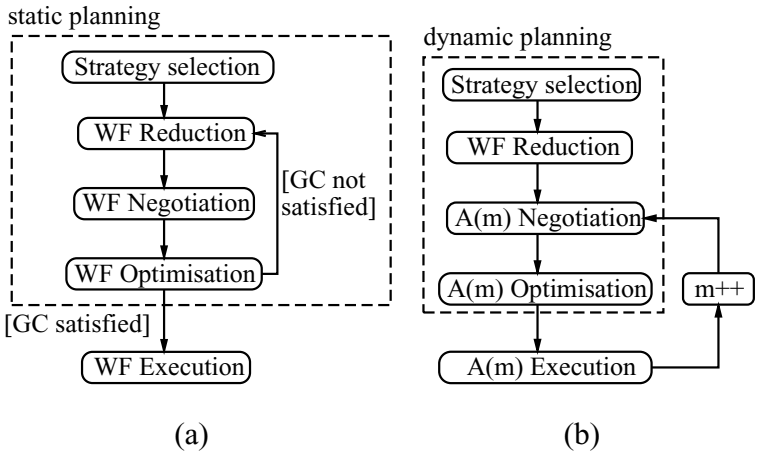


Fig. 16. Workflow execution models: (a) with static planning, (b) with dynamic planning.

whole workflow. If GCs can not be satisfied after the *WF Negotiation* and *WF Optimization*, the workflow is reduced by excluding the first activity of the workflow. The excluded activity is optimized separately from the rest of the workflow by searching for the candidate service that offers the best QoS. In the next iteration only the reduced workflow is considered. However, the user may annotate each activity with the expected average runtime in order to increase the chance to find a solution with fewer iterations. If the GCs are satisfied, the *WF execution* may start.

In case the execution of certain activities modifies the *meta data*, the dynamic workflow planning is applied (see Fig. 16(b)). For each activity $A(m)$ the negotiation and optimization is performed individually before the execution. After the execution of activity $A(m)$, the next activity $A(m+1)$ is considered. This iterative process of dynamic planning is completed when all activities of the workflow are executed.

5. Case study

In this section we demonstrate the modeling of QoS-aware workflows using a real world application for maxillo facial surgery simulation.

5.1. Maxillo facial surgery simulation

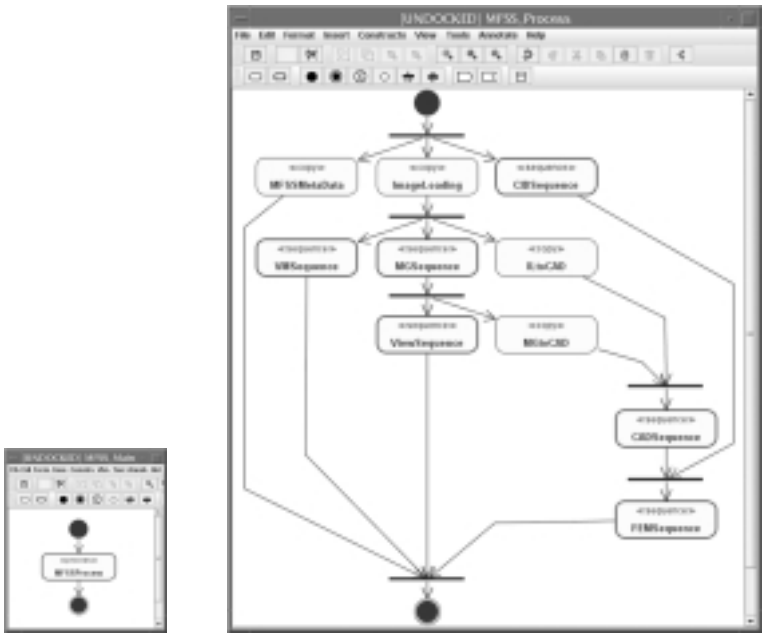
Maxillo facial surgery simulation (MFSS) is one of the six medical applications that we have used within the framework of GEMSS project [22]. The application facilitates the work of medical practitioners and provides the pre-operative virtual planning of maxillo-facial surgery. The application consists of a set of components which can run on a local machine or on different remote machines. These components may be organized as a Grid workflow in order to simplify the work of the end users. Cao et al. [11] describe the specification of MFSS workflow using the Triana tool [13], but QoS requirements are not considered.

MFSS is used for patients with in-born deformations of the mid-face [11]. The MFSS simulation predicts the results of the surgery by putting the bone into a new position using the finite element method. The goal of the surgery is to achieve pleasant medical and cosmetic results. The main steps of the simulation are: (1) *Mesh generation* is used for the generation of meshes necessary for the finite element simulation; (2) *Mesh manipulation* defines the initial and boundary conditions for the simulation; (3) *Finite Element Analysis* is a fully parallel MPI application running on a remote HPC cluster. The application is exposed as a VGE service and can be invoked using standard Web Services technology. We use QoWL in order to automate the execution of the MFSS application.

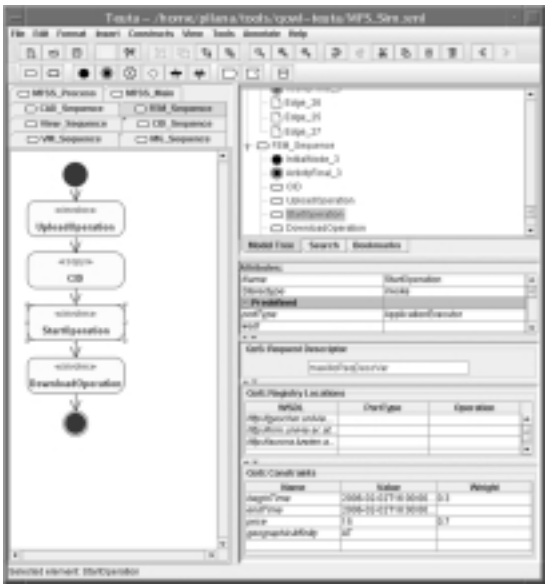
In the following we describe the QoS-aware specification of the MFSS workflow.

5.2. Workflow specification

We have used our UML based workflow editor Teuta (see Section 4.2) for the specification of the MFSS workflow. The workflow specification process involved the definition of the flow of workflow activities and the association of the corresponding properties (such as QoS related properties). Figure 17 illustrates the process of specification of MFSS workflow with Teuta.



(a) (b)



(c)

Fig. 17. Specification of MFSS workflow with Teuta editor. (a) The root element. (b) The content of the root element. (c) Properties of *StartOperation* activity.

The user may combine the predefined UML modeling elements, which are available in the Teuta toolbar, to specify the flow of workflow activities. Figure 17(a) depicts the *MFSSProcess* activity, which is an instance of the *process* element. A *process* element is indicated by the stereotype `<<process>>`. The *MFSSProcess* activity indicates the root of the MFSS workflow. Complex activities, such as *process*, may comprise a group of activities. Teuta supports hierarchical modeling, by representing the body of a complex activity as a subgraph. The body of the *MFSSProcess* activity is depicted in Fig.17(b). The UML element *InitialNode*, which is represented as a filled black circle, defines the starting point of a workflow or of a complex activity.

We have used the *copy* element, which is indicated by the stereotype <<copy>>, to express the data flow. For instance, *MFSSMetaData* activity copies the input data of the workflow to the corresponding activities. The UML elements *Fork* and *Join*, which are represented as bold horizontal bars, express the split and join of multiple flow branches respectively. For instance, after the completion of the *ImageLoading* activity, the flow is split into three branches. The UML elements *Fork* and *Join* are mapped to the BPEL element *Flow*. The UML element *ActivityFinal*, which is represented as a circle surrounding a smaller solid filled circle, indicates the end of the workflow or the end of a complex activity.

The *MFSSProcess* activity comprises several complex activities of type *sequence*, which are indicated by the stereotype <<sequence>>. For instance, the body of the *FEMSequence* activity, placed on the right-down corner of Fig. 17(b)), is represented in Fig. 17(c). An *invoke* element, which is indicated by the stereotype <<invoke>>, specifies the invocation of a remote or local service operation. For instance, the activity *UploadOperation* invokes the *upload* operation of the remote service.

With each workflow element we have associated a set of properties by using the property panel, which is located on the right hand side of Teuta GUI (see Fig. 17(c)). For instance, Fig. 17(c) shows the properties of the invoke element *StartOperation*. The top compartment of the property panel allows the association of attributes such as *name*, *portType* and *operation*. We use the lower three compartments to specify QoS constraints such as *beginTime*, *endTime*, *price*, and *geographicAffinity*.

After the completion of the MFSS workflow specification Teuta is able to generate automatically the corresponding QoWL representation, which is used as input for QWE for workflow execution (see Section 4.1). An excerpt of QoWL representation of MFSS workflow, which specifies the *StartOperation* activity, is depicted in Fig. 18.

6. Related work

Several projects are contributing to the establishment and improvement of the Grid workflow technology, each focusing on a specific research aspect. Recent developments focus on the establishment of the service oriented infrastructure for Grids by augmenting standardized protocols and services [29]. Triana [13], Askalon [2], JOpera [32], eXeGrid [21] are developing tools and languages for graphical workflow composition. The P-GRADE Portal is exploring the collaborative Grid workflows [37]. Pegasus [7,14] and LEAD [34] projects are focused on the development of workflow support for large scale Grid applications (such as galaxy morphology, tomography and mesoscale meteorology). The aspects of semantic grid workflows are investigated within the Taverna [40] and Kepler [6] projects. Gridant [1] presents client-side workflow management system enabling the clients to map the process without considerable help from the service providers. Rygg et al. [36] demonstrate the usage of a commercial workflow engine for the composition of bio-workflows. Work presented in [28] describes a lightweight system for business workflows that are based on Web Services related technologies (such as SOAP, WSDL and BPEL). GridXSLT, an implementation of the XSLT for distributed Web Service orchestration, is described in [23]. Montagnat et al. [26] investigate data composition patterns for service-based workflows. Perera et al. [33] propose Web Service extensions necessary for the proper execution of service oriented Grid workflows. Work described in [31] demonstrates the usage of Grid workflow technologies for modeling of bioclimatic workflows.

There is not much related work focused on the development of a Grid services infrastructure that provides QoS guarantees. Moreover, not enough attention is paid to QoS-aware Grid workflows. Cardoso et al. elaborate theoretical concepts of a QoS-aware workflow defining QoS workflow metrics [12]. Zeng et al. investigate QoS-aware composition of Web Services using integer programming method [43]. The services are scheduled using local planning, global planning and integer programming approaches. Gridbus Project [19] is addressing the QoS-aware Grid workflows. Recent developments are following research problems on budget constraint scheduling of workflow applications on utility Grids based on genetic algorithms [41]. However, within the framework of Gridbus project workflows are specified textually based on XML, which has been proved as a non-intuitive and error-prone approach. While time and cost constraints are considered, there is no support for security and legal QoS constraints. In contrast to existing related work, we are developing a QoS-aware workflow system that supports time, cost, security, and legal constraints. Moreover, our system supports the graphical specification of QoS-aware workflows based on the latest UML standard.

```
...
<invoke inputVar="CID" name="StartOperation" operation="start" portType="ApplicationExecutor">
  <qos-constraints ReqDesc="maxilloReqDescVar">
    <registry wsdl="http://gescher.univie.ac.at:9357/registry/reg?wsdl" />
    <registry wsdl="http://kim.univie.ac.at:9357/registry/reg?wsdl" />
    <registry wsdl="http://aurora.tuwien.ac.at:9357/registry/reg?wsdl" />
    <qos-constraint name="beginTime" value="2006-02-02T16:00:00.000+02:00" weight="0.3" />
    <qos-constraint name="endTime" value="2006-02-02T18:00:00.000+02:00" />
    <qos-constraint name="price" value="15" weight="0.7" />
    <qos-constraint name="geographicAffinity" value="AT" />
  </qos-constraints>
</invoke>
...
```

Fig. 18. QoWL representation of *StartOperation* activity.

7. Conclusions and future work

Currently, most of the workflows are specified in textual form, or are composed based on a self-defined graphical notation. Moreover, there is a lack of adequate tool support for workflow specification, and workflow specification tools are usually not well integrated with Grid environments. Furthermore, there is a large number of application domains for Grid workflows, such as life sciences (for instance medical simulation services) and engineering (for instance vehicle development support), that demand a guarantee that workflow activities are performed within the specified *time*, *cost*, *security*, and *legal* constraints. We consider that for the wide acceptance of Grid technology it is important that specification of tasks to be executed on the Grid is simple, and that the execution of these tasks should meet the user's requirements.

In this paper we have addressed the issue of high-level specification of QoS-aware Grid workflows. In order to streamline the process of workflow specification we have developed a Domain Specification Language (DSL) for QoS-aware Grid workflows based on UML2.0 standard. Furthermore, we have developed a system prototype that supports the whole workflow lifecycle from high-level specification to execution. A set of QoS-aware service-oriented components is provided for workflow planning to support automatic constraint-based service negotiation and workflow optimization. For improving the efficiency of workflow planning, we introduced a QoS-aware workflow reduction technique. Our system allows the specification of a comprehensive set of QoS requirements, that consider performance, economical, legal and security aspects.

In addition, we have described the annotation of workflow activities with QoS information. For each complex activity (such as sequence or flow) we described the implications of the specified QoS for the execution of comprised activities. Furthermore, QoS-related aggregation functions are described for each complex activity. Please note that the overall workflow may be considered as a complex activity as well. Based on this QoS information, a QoS-aware Grid Workflow Engine negotiates with multiple candidate services to find services that fulfill the specified QoS constraints. We implemented two preliminary workflow planning approaches, static and dynamic. We evaluated our approach by modeling a real-world workflow for maxillo facial surgery simulation, and showed the hierarchical modeling capabilities of our approach for modeling complex activities.

In the future we plan to extend our approach with workflow adaptivity mechanisms.

Acknowledgments

The work described in this paper was supported by the Austrian Science Fund as part of Aurora Project under contract SFBF1102 and by the European Union's GEMSS Project under contract IST 2001-37153. We would like to thank Aleksandar Dimitrov, Gerhard Engelbrecht, Rainer Schmidt and Nikolay Terziev for their contributions to the implementation of VGE. We thank anonymous reviewers for helpful suggestions.

References

[1] K. Amin, G. von Laszewski, M. Hategan, N. Zaluzec, Sh. Hampton and A. Rossi, *GridAnt: A Client-Controllable Grid Workflow System*, 37th Hawaii International Conference on System Sciences. Big Island, HI, USA, 2004.

[2] Askalon Project. http://dps.uibk.ac.at/projects/askalon/.

[3] S. Benkner, G. Berti, G. Engelbrecht, J. Fingberg, G. Kohring, S.E. Middleton and R. Schmidt. GEMSS: Grid Infrastructure for Medical Service Provision, *Journal of Methods of Information in Medicine* **44** (2005).

[4] S. Benkner, I. Brandic, G. Engelbrecht and R. Schmidt. *VGE – A Service-Oriented Grid Environment for On-Demand Supercomputing*, Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (Grid 2004), Pittsburgh, PA, USA, November 2004.

[5] S. Benkner and G. Engelbrecht, *Generic QoS Support for Application Web Services*, Proceedings of the International Symposium on Web Services, CSREA Press, Las Vegas, USA, June 2005.

[6] C. Berkley, S. Bowers, M. Jones, B. Ludäscher, M. Schildhauer and J. Tao, *Incorporating Semantics in Scientific Workflow Authoring*, 17th International Conference on Scientific and Statistical Database Management, University of California, Santa Barbara, CA, USA, 2005.

[7] J. Blythe, E. Deelman and Y. Gil, Automatically composed workflows for grid environments, *IEEE Intelligent Systems* **19**(4) (2004), 16–23.

[8] I. Brandic, S. Benkner, G. Engelbrecht and R. Schmidt, *Towards Quality of Service Support for Grid Workflows,* Proceedings of the European Grid Conference 2005 (EGC2005), Amsterdam, The Netherlands, February 2005.

[9] I. Brandic, S. Benkner, G. Engelbrecht and R. Schmidt, *QoS Support for Time-Critical Grid Workflow Applications,* Proceedings 1st IEEE International Conference on eScience and Grid Computing, Melbourne, Australia, December 2005.

[10] I. Brandic, S. Pllana and S. Benkner, *High-level Composition of QoS-aware Grid Workflows: An Approach that Considers Location Affinity,* Workshop on Workflows in Support of Large-Scale Science (works06). In conjunction with HPDC06, Paris, France, 2006.

[11] J. Cao, G. Berti, J. Fingberg and J.G. Schmidt, *Implementation of Grid-enabled Medical Simulation Applications using Workflow Techniques,* The Second International Workshop on Grid and Cooperative Computing, Shanghai, China, 2003.

[12] J. Cardoso, *Quality of Service and Semantic Composition of Workflows*, Ph.D. Dissertation. Department of Computer Science. 2002, University of Georgia: Athens, GA, USA, 215.

[13] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor and I. Wang, *Programming Scientific and Distributed Workflow with Triana Services,* In Grid Workflow 2004 Special Issue of Concurrency and Computation: Practice and Experience, 2005.

[14] E. Deelman, G. Singh, M.-H. Su et al., Pegasus: A framework for mapping complex scientific workflows onto distributed systems, *Scientific Programming* **13**(3) (2005), 219–237.

[15] T. Fahringer, S. Pllana and A. Villazon, *AGWL: Abstract Grid Workflow Language*, International Conference on Computational Science, Programming Paradigms for Grids and Metacomputing Systems. Krakow, Poland, June 2004.

[16] I. Foster, C. Kesselman and S. Tuecke, The anatomy of the grid – enabling scalable virtual organizations, *The International Journal of High Performance Computing Applications* **15**(3) (2001), 200–222.

[17] GEMSS Consortium, Report on COTS Security Technologies and Authorisation Services. Deliverable D2.2c. GEMSS Project, European Commission Framework V Project No. IST-2001-37153. February 2004.

[18] The GEMSS Project: Grid-Enabled Medical Simulation Services, EU IST Project, IST-2001-37153, http://www.gemss.de/.

[19] The Gridbus Project. http://www.gridbus.org/.

[20] J.A.M. Herveg, F. Crazzolara, S.E. Middleton, D. Marvin and Y. Poullet, *GEMSS: Privacy and Security for a Medical Grid*, HealthGRID 2006, Clermont-Ferrand, France, 2004.

[21] A. Hoheisel, *User Tools and Languages for Graph-based Grid Workflows*, In: Special Issue of Concurrency and Computation: Practice and Experience, Wiley, 2004.

[22] D.M. Jones, J.W. Fenner, G. Berti, F. Kruggel, R.A. Mehrem, W. Backfrieder, R. Moore and A. Geltmeier, *The GEMSS Grid: An evolving HPC Environment for Medical Applications*, HealthGrid 2004, Clermont-Ferrand, France, 2004.

[23] P.M. Kelly, P.D. Coddington and A.L. Wendelborn, *Distributed, Parallel Web Service Orchestration Using XSLT*, First International Conference on e-Science and Grid Computing Melbourne, Australia, December 2005.

[24] The lp_solve Project http://lpsolve.sourceforge.net/5.1 2006.

[25] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, 1996.

[26] J. Montagnat, T. Glatard and D. Lingrand, *Data Composition Patterns in Service-Based Workflows*, Workshop on Workflows in Support of Large-Scale Science (works06). In conjunction with HPDC06, Paris, France, 2006.

[27] OASIS, Web Services Business Process Execution Language (WSBPEL) 2.0, 2006.

[28] J. Oberleitner, F. Rosenberg and S. Dustdar, *A Lightweight Model-driven Orchestration Engine for e-Services*, 6th VLDB Workshop on Technologies for E-Services, colocated with VLDB Trondheim, Norway, 2005.

[29] Open Grid Services Architecture. http://www.globus.org/ogsa/.

[30] Object Management Group (OMG), UML 2.0 Superstructure Specification. http://www.omg.org, August 2005.

[31] J.S. Pahwa, R.J. White, A.C. Jones et al., *Accessing Biodiversity Resources in Computational Environments from Workflow Applications*, Workshop on Workflows in Support of Large-Scale Science (works06). In conjunction with HPDC06, Paris, France, 2006.

[32] C. Pautasso, *JOpera: Visual Composition of Grid Services*, In: ERCIM News No. 59, October 2004.

[33] S. Perera and D. Gannon, *Enabling Web Service Extensions for Scientific Workflows*, Workshop on Workflows in Support of Large-Scale Science (works06). In conjunction with HPDC06, Paris, France, 2006.

[34] B. Plale, D. Gannon, D.A. Reed, S.J. Graves, K. Droegemeier, B. Wilhelmson and M. Ramamurthy, *Towards Dynamically Adaptive Weather Analysis and Forecasting in LEAD*, 5th International Conference on Computational Science, Atlanta, GA, USA, 2005.

[35] S. Pllana and T. Fahringer, *Performance Prophet: A Performance Modeling and Prediction Tool for Parallel and Distributed Programs*, In the 2005 International Conference on Parallel Processing (ICPP 2005 Workshops), Oslo, Norway, June 2005. IEEE Computer Society.

[36] A. Rygg, S. Mann, P. Roe and O. Wong, *Bio-Workflows with BizTalk: Using a Commercial Workflow Engine for eScience*, Proceedings 1st IEEE International Conference on eScience and Grid Computing, Melbourne, Australia, December 2005.

[37] G. Sipos, G.J. Lewis, P. Kacsuk and V.N. Alexandrov, *Workflow-Oriented Collaborative Grid Portals*, Proceedings of the European Grid Conference 2005 (EGC2005), Amsterdam, The Netherlands, February 2005.

[38] Web Service Level Agreement. http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf.

[39] Web Services Policy (WS-Policy). http://ifr.sap.com/ws-policy/index.html.

[40] K. Wolstencroft, T. Oinn, C. Goble, J. Ferris, Ch. Wroe, P. Lord, K. Glover and R. Stevens, *Panoply of Utilities in Taverna*, Proceedings 1st IEEE International Conference on

eScience and Grid Computing, Melbourne, Australia, December, 2005.

[41] J. Yu and R. Buyya, *A Budget Constraint Scheduling of Workflow Applications on Utility Grids using Genetic Algorithms*, Workshop on Workflows in Support of Large-Scale Science (works06). In conjunction with HPDC06, Paris, France, 2006.

[42] J. Yu and R. Buyya, *A Taxonomy of Workflow Management Systems for Grid Computing*, Technical Report, GRIDS-TR-2005-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, March 10, 2005. http://www.gridbus.org/reports/GridWorkflowTaxonomy.pdf.

[43] L. Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam and H. Chang, QoS-aware middleware for web services composition, *IEEE Transactions on Software Engineering* **30**(5) (May 2004), 311–327.