© University of Southampton IT Innovation Centre
and Datamat, 2006

P5.3.2
Adaptive Workflow Technology

Workpackage:	5	Grid Dynamics
Author(s):	Nikolaos Matskanis, Mike Surridge, Justin Ferris Barbara Cantalupo	IT Innovation DATAMAT
Authorized by	Mike Surridge	IT Innovation
Doc Ref:	P5.3.2	
Reviewer	Peer Hasselmeyer	NEC
Dissemination Level	Public	

Date	Author	Comments	Version	Status
2006-02-20	B. Cantalupo	Documents structure and contents concerning parser outlined	0.1	Outline
2006-02-27	N. Matskanis	Enactor structure and content	0.1b	Outline
2006-03-03	N. Matskanis	Enactor Content. First Draft	0.2	Draft
2006-03-06	B. Cantalupo	Language and Parser Content Added	0.3	Draft
2006-03-09	M.Surridge	Fixed document format, and other bugs, added JF contribs	0.4	Draft
2006-03-16	N. Matskanis	Corrections as suggested by the internal reviewer	0.5a	Draft
2006-03-17	B. Cantalupo	Corrections as suggested by the internal reviewer	0.5b	Draft
2006-04-03	N. Matskanis	Final adjustments of Approved Document	1.0	Final

Executive Summary

This document describes the result of WP5.3 activity in the months 12-18 concerning development of workflow technology as part of the implementation of the concept of a Grid Virtual Infrastructure Model. The work has been focused in two main areas that are workflow representation and enactment.

From the workflow representation perspective the main output has been the refinement of the workflow ontology OWL-WS defined in months 1-12, and the development of a parser to be used by both for the Workflow Enactment, which is also described in this document, and the Semantic Workflow Editor, which is defined and developed in WP6. The detailed semantics of OWL-WS was refined to remove ambiguities when used in the enactor evaluation model. Some adjustment to the parser were also made to fulfil requirements from the enactment phase, e.g. to maintain a complete history of workflow substitutions to support clear exception reporting. APIs to compose the OWL-WS parser were also developed. Some shortcomings in the underlying OWL-S language were identified and analysed and will be further addressed in future languages updates, most notably concerning the difficulty of scoping process and data names when composing or substituting workflows in a language where all names must be globally unique.

From the enactment perspective an evaluation model based on the workflow representation model has been defined. The evaluation model defines how workflows that contain abstract service descriptions without execution information are bound to concrete ones with the necessary details to invoke them remotely. The relationships and dependencies of this model with its environment are also described. Based on this evaluation model a workflow engine has been designed and implemented. This engine enacts the workflow by evaluating it first if it is necessary. The order of the workflow evaluation is defined by an external service and can be different from the execution one. Additionally an experiment has been set up to test the engine capabilities and the effectiveness of the language and the evaluation model. The results of the experiment highlight the need to address the OWL-S language features discussed above, and show that performance (though better than expected) will need to be optimised in the registry and in the enactor itself.

Table of Contents

1 Introduction 1
2 Objectives 2
2.1 The role of workflow in NextGRID 2
2.2 Workflow enactment requirements 2
2.3 NextGRID architecture questions addressed 3
3 OWL-WS model and language management component 5
3.1 OWL-WS fundamentals 5
3.2 Reference model and language 6
3.3 OWL-WS Parser Analysis and Design 10
3.3.1 OWL-S Related Tools Survey 10
3.4 OWL-WS Parser Design and API 11
3.5 Language issues 12
3.5.1 Multiple Service Models 12
3.5.2 Scope of OWL-S parameters 13
3.5.3 Support of co-allocation and prioritisation 13
4 Workflow enactor 15
4.1 Dynamic workflow enactment 15
4.2 Use of OWL-WS language features 16
4.3 Relationship to Other Components of the Grid-VIM 17
4.3.1 QoS decision services 17
4.3.2 Registry services 18
4.3.3 Brokering services 18
4.4 Enactor implementation 19
4.4.1 Available Technologies Overview 19
4.4.2 The Mindswap execution engine 20
5 NextGRID Enactor Architecture 21
6 Experiment 24
6.1 Experiment Components and Setup 24
6.1.1 GRIA 24
6.1.2 Grimoires Registry 25
6.1.3 OWL-WS Enactor and other Components of Grid VIM 25
6.1.4 Setup 26
6.2 Experiment Walk Through 28
7 Conclusions and Future Work 31
7.1 NextGRID architecture questions addressed 32
References 33
Appendix 35
A1. AbstractProcess with a Query Profile 35

1 Introduction

This report describes NextGRID activity in WP5.3 that is focused on developing workflow technology for dynamic workflow management. More specifically it describes the activities on the workflow representation model and the enactment engine.

First, in Section 2, we introduce the overall context of workflows in NextGRID providing a brief introduction to the Grid Virtual Infrastructure Model, its relationship to the workflow technologies, and summarising the architectural questions we hope to answer by using the parser and enactor.

Section 3 covers the work done at the workflow representation language, starting with a review of results from the first year of the project, describing the refinements to the reference model, and the implementation of the OWL-WS parser and the available API. This section also discusses issues identified with the underlying OWL-S language that make it less well suited to NextGRID requirements than originally thought.

Section 4 starts with the description of the workflow enactment model. The relationship of the engine with the other components of the Grid VIM is then described and a picture of the overall architecture is given. Finally the enactment engine architecture is analysed by describing the extensions that mainly implement the workflow evaluation mechanism.

In order to verify the effectiveness of the developments, an experiment has been carried out to demonstrate the dynamic adaptation of workflow enactment, although not yet to answer the architectural questions posed about the full Grid VIM. The experiment setup, step-by-step description and results are described in section 6.

The concluding Section summarises the refinement of the representation model (up to date and planned), the development of the architecture and the result of the experiment, and the issues highlighted by it. Finally, some comments are given on the architectural questions, indicating what this initial implementation exercise suggests about possible answers or investigation methods.

2 Objectives

2.1 The role of workflow in NextGRID

The aim of NextGRID is to design and develop components that will define the "Next generation grid architecture" [1]. The target is to broaden the use of grids from the research-academic domain to include applications from the business world. So grid architecture should be such that will extend the support of application domains and adapt to different organisations in a secure and economically viable way.

Workflow has a major role in grid dynamics and is adopted as the core technology for applying grid dynamics in NextGRID. It provides the ability to compose and dynamically adapt grid services available in distributed systems and orchestrate their execution.

The workflow components, the services and their environment compose an infrastructure that is described in NextGRID as the concept of Grid Virtual Infrastructure Model or Grid VIM [2]. This infrastructure is designed to allow Grid applications and Grid business models and processes to be combined at run-time. This is an essential architectural feature without which it would be impossible to design applications independently of the business models for provision and procurement of services from which they are composed. Workflow is one of the key technologies that can provide this adaptability to distributed Grid environments at run-time.

In parallel with the development of a dynamic workflow representation, we describe the design and development of an enactor prototype, to enable experiments on combined applications and business policies.

2.2 Workflow enactment requirements

The analysis of the Grid VIM enactment procedure shows that the following aspects must be supported to enable run-time adaptable infrastructure:

- Run-time bindings: workflows need not specify a binding of every task to a specific service, so that the bindings can be chosen at run-time.
- Selective enactment: a single service may provide multiple functions, and it must be possible to choose which is bound to an abstract task, supported by the service.
- Workflow substitution: some abstract tasks may be bound at run-time to more detailed workflows that can be inserted into the enactment at run-time. A common example is substitutions with template business operations such as account and billing workflows.
- Workflow prioritisation. Critical processes, which are either expensive in resources or define the result or the performance of the workflow, must have high priority in the evaluation order.

This means any adaptive workflow representation used to specify behaviour of the Grid VIM must have the following characteristics:

- A workflow description must allow the inclusion of abstract tasks, which contain a description of their function and data/control dependencies, but do not specify which service should be used to implement them.
- A service description must include a description of its functionality, which can be published in a service registry and matched against an abstract task description,

enabling discovery of services that could be used to implement each abstract task in a concrete way.

- When a service has been discovered, it must be possible to obtain from it a process (workflow) over its operations, which provides the functionality needed for the original abstract task.
- It must be possible to discover services that match a functionality description, even if the corresponding process specified by the service has extra (e.g. business) dependencies that the user must be able to satisfy.
- If a service supports multiple functions, the discovery process must provide the correct process matching the desired function.
- A quality of service (QoS) service will define the evaluation priority of the workflow by taking into account parameters provided by the user or the service provider. This mechanism can spot potential problems on workflow execution and prevent wasting of resources by prompting users to take necessary actions.

The binding of an Abstract Task during enactment is summarised in Figure 1:

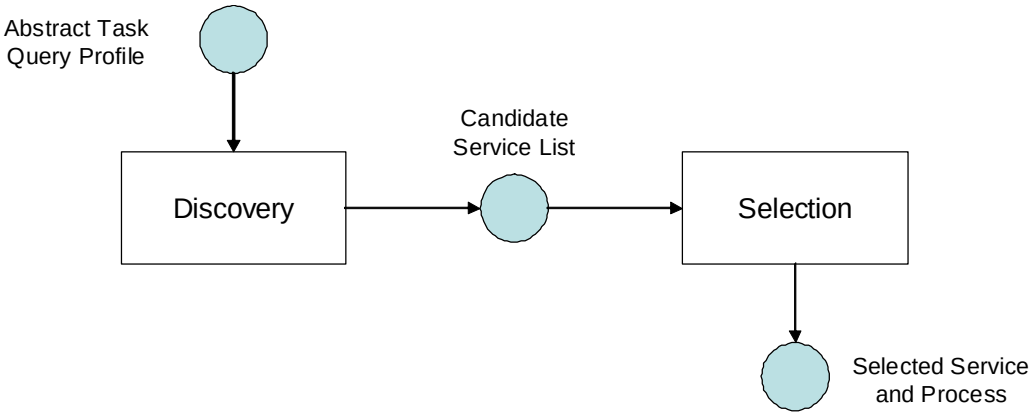

Figure 1: Abstract Task Evaluation Requirements

The abstract task "query profile" (containing the task functionality description) is taken from the workflow to be enacted and fed into the service discovery and selection process. From this we obtain a service and a workflow made up from its operations, which (when supplied with the same inputs and outputs) replaces the original abstract task in the original workflow. This procedure is known as task evaluation.

In practice, it is possible that the selected service is not concrete, in which case the substituted workflow will contain further abstract sub-tasks or unbound inputs. In that case, further evaluation is needed to discover services and processes that provide these. However, eventually, one should reach a point where the workflow evaluation leads to a set of concrete tasks (i.e. tasks that can be executed using a specific service).

At this point, the enactment procedure executes the tasks, and removes them from the description of workflow still to be enacted. This procedure is known as task application.

2.3 NextGRID architecture questions addressed

The role of the workflow language and enactor is to provide the "glue" in the Grid VIM. Its main purpose at this stage in NextGRID is to support experiments centred on how other components such as registries and QoS services interact.

There are three architectural questions that relate specifically to the Grid VIM:

- 80) What is the relative importance of overlapping workflow standards: BPEL and ebXML from OASIS, and OWL-S from the SWS Consortium?
- 84.1) What kind of VM should be an architectural basis for the use and enactment of workflow?
- 84.2) How does this relate to workflow languages and API?

These questions will be answered in the future, when further developments, integration tests and experiments will be carried out. The experiment described here (see Section 6) was intended only to show that the enactor could be used to couple other components in a flexible way. Nevertheless, the work did go some way towards answering these questions, or at least showing how they could be answered.

3 OWL-WS model and language management component

3.1 OWL-WS fundamentals

A Semantic Workflow Language and Model for representing workflows in the context of the Grid Virtual Infrastructure was defined in WP5.3 activity during months 1-12. Several languages in both semantic and workflow context were analysed and OWL-WS was defined as a "compositional" extension of OWL-S, an OWL ontology for describing (Web) Services. Since then, the OWL-WS workflow model was refined to provide an unambiguous semantic and APIs to manage the language were defined and developed.

A detailed description of the OWL-WS foundations and the reason of the choice of extending OWL-S, both from the NextGRID requirements and state-of-the-art perspective are described in [3]. Here we just provide basic information to help understanding of the workflow model from a practical point of view and the design of the model management components that will be described in next sections.

Semantic Workflow and Service model we take as a reference is based on the a basic Service concept and a couple of fundamental statements:

- ☑ *A **Semantic NextGrid Service** (**Service**) is a Grid Service designed to operate in a NextGRID environment also using some kind of semantic information.*
- *Services and Workflows can be viewed as the same functional entity addressed from a different perspective and therefore can be managed in the same way.*
- *Services (and Workflows) can be described as Abstract or Concrete*

We adopt an extension of OWL-S [4] to provide a representation of this model. OWL-S is an ontology based on OWL. It can be viewed as a *language* for describing services, reflecting the fact that it provides a standard vocabulary to create service descriptions. OWL-S description is composed by three main components:

- *a Service Profile, which tells "what the service does" in terms of information about the provider, functional description of the service and additional properties to describe features of the service;*
- *a Service Process, formally a subclass of Service Model, which describes "how to use the service" detailing the semantic content of requests, the conditions under which particular outcomes will occur, and, where necessary, the step by step processes leading to those outcomes.*
- *a Service Grounding which specifies the details of "how the service can be accessed" typically specifying a communication protocol, message formats, and other service-specific details such as port address to be used in contacting the service.*

A Service Process can either be Atomic (which means it can be executed by sending a request message as described in the Service Grounding), or Composite (describing a sequence, conditional or alternative branches, or loop over one or more sub-Processes)[1]. Every Process thus describes (possibly via several nested Composite structures) a concrete workflow made up of Atomic processes that can be executed by the service.

[1] OWL-S also provides the SimpleProcess sub-class of Process, but this is used only as a super-class of other Process types, and will not be considered further here.

The Service Grounding describes the interface(s) and end-point(s) for each Atomic Process supported by the Service. This allows a user to construct and send request messages to the service in order to execute any Atomic Process from the Service Model.

OWL-WS extension uses the concept of composite process for workflow modelling but while OWL-S focuses on modelling a workflow that is internal to a single service, i.e. a sequence of calls to the service operations, we extend this notion to comprise also inter services processes. In practice, while an OWL-S Process specifies the steps needed to interact with a service implementation, an OWL-WS Process specifies the steps needed to interact also with different services in order to perform the functionality described in the service Profile.

Moreover, OWL-WS extends OWL-S by adding a new type of Process, the Abstract Process. This corresponds to an abstract task that has not yet been evaluated and so is not yet associated with a concrete service. An Abstract Process is thus not related to any grounding or concrete service implementation, but describes its functionality via a profile, linked by a new property "definedBy", that can be used as a Query Profile to discover a concrete service and workflow.

In [3] we already provided high-level specification of the relationship between the Semantic Workflow and Service model and its OWL-WS representation and suggested main operation to be used in the context of a service-provider scenario, that is Profile Matching, Concrete Workflows Merging, Abstract Workflow Composition. In the following sections we will provide a more detailed specification with some adjustment derived from initial experiment in developing and using workflow enactor.

3.2 Reference model and language

Experience in parser development and integration with NextGRID dynamics components raised some new requirements and issues concerning the OWL-WS model and language:

- The need of a more detailed and unambiguous specification of OWL-WS became evident to avoid misunderstanding in modelling, and therefore in the language and related parser usage. This led us to provide a better specification of the model, independent of the language itself, to provide common understanding and agreement without involving complex details of the OWL-based language.
- Adjustments in the model were also provided to fulfil new requirements, the main one being the need of maintaining history of workflow substitution until the completion of the execution of the overall workflow.

To provide a better understanding of the Semantic Workflow and Service Model (SWSM), avoiding that the detail of the OWL-WS representation can generate confusion and misunderstanding, we provide simplified graphical modelling of fundamental components and related mapping of OWL-WS language using UML notation.

Graphical notation is based on representing the usual basic concepts (**Figure 2**):

- *Profile* containing properties, both functional (e.g. functionality, input, output) and non functional (e.g. cost, maximum execution time), describing what the service/workflow provides or should provide.
- *Process*, describing the execution flow of the service/workflows (e.g. data flow, control flow).
- *Grounding* describing implementation mechanism to activate the service/workflow.

It is worth noticing that, even if we are using OWL-S terminology to simplify understanding of the overall picture, these concepts are not strictly dependent on the specific semantic language and therefore they must not be confused with the language classes. We also represent OWL-WS language mapping with a class diagram notation.

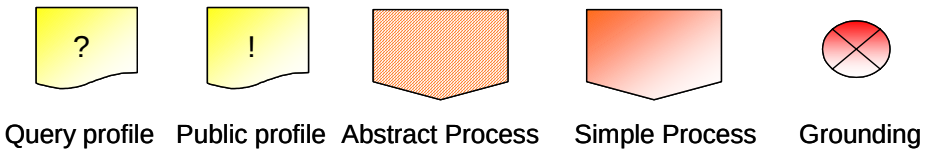

Figure 2: Simple Graphical Notation Legend [2]

In the working experience with the development of the OWL-WS parser and the integration with the evaluation model we understood the need to better highlight distinction between profile that is used to query abstract service and the service that is published to describe Concrete Service (or Workflow) properties. In [3] we presented distinction between constraints and capabilities and also informally introduced Query Profile. In order to avoid any ambiguity, we now state that:

- A Profile modelling an Abstract Service/Workflow, and therefore referred by an Abstract Process, contains *constraints* information and is called ***Query Profile*** (*?Profile*). Constraints specified in the ?Profile are the requirements that a Services matching the Profile must satisfy or hints that must be applied in the evaluation of the related sub workflow.
- A Profile modelling a Concrete Service/Workflow, and therefore referred by a Service contains *capabilities* information and is called ***Public Profile*** (*!Profile*). Capabilities contained in the !Profile are the properties that the Service or the related sub workflows can guarantee.

In **Figure 3** UML class diagram of OWL-WS model is represented. We note that an Abstract Process is a Process with a new property “definedBy” that points to a (Query) Profile [3]. Query Profile is just a Profile with a “defines” property (inverse of “definedBy”) pointing to an Abstract Process. Attention must be also paid to the “has_process” property, linking a Service Profile to the Process that is associated with the service. As we will see, it is fundamental to model abstract workflow substitution without any loss of information.

[2] Composite Process is represented by composition of Simple and Abstract Processes.

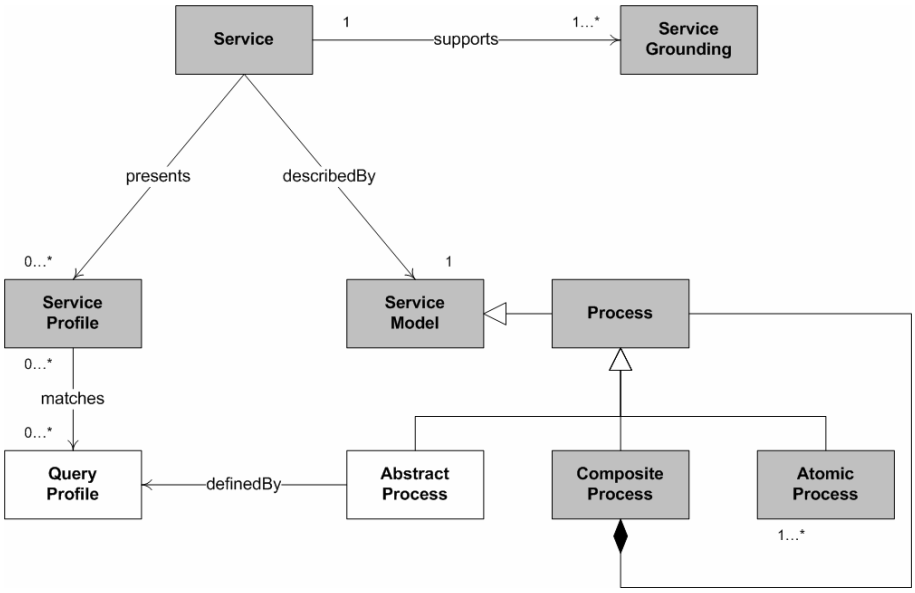

Figure 3: OWL-WS Class Diagram Model

In **Figure 4** a Concrete (composite) Service is represented with its OWL-WS mapping. From a model perspective a Concrete Service is composed by a Public Profile and a composition of Atomic Processes each one with its own Grounding. This kind of service is easily modelled in OWL-S without using any of the OWL-WS extensions.

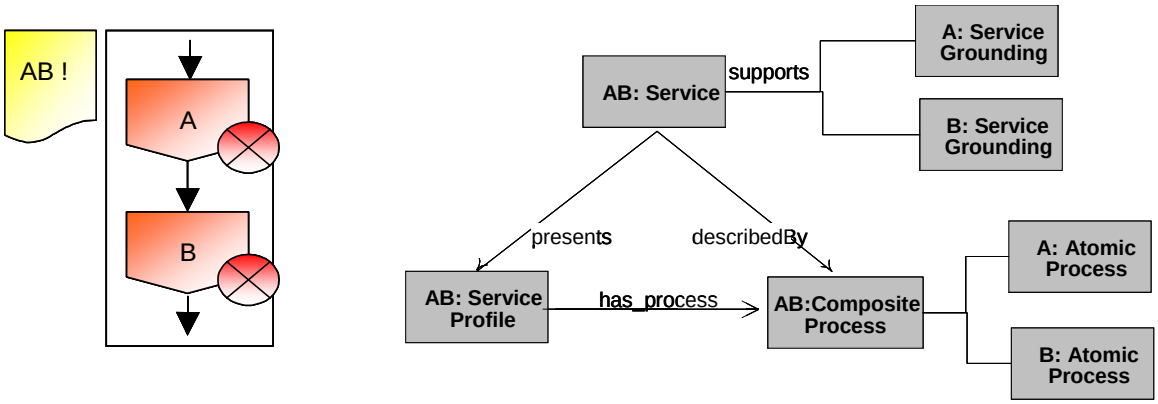

Figure 4: Concrete (Composite) Service Instance – Model and Language Mapping

More interesting is the Abstract Service modelling that is sketched in **Figure 5**. From a model perspective an Abstract Service is only represented by a Profile component but in order to make the OWL-WS model consistent an Abstract Service is defined in relation to an Abstract Process and a Query Profile. The Service class is only used as a container while the Abstract Process is the bridge to the Query Profile describing requirements for the Abstract Process. In this case, Service property "presents" does not point to any Profile because this would have to be a Public Profile, and the Profile property "has_process" does not point to any Process. As we will see, "has_process" is used to build Abstract Workflows composing Abstract Services.

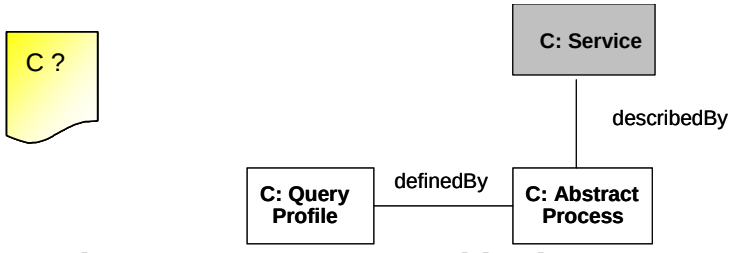

Figure 5: Abstract Service Instance – Model and Language Mapping

An Abstract Workflow is represented in **Figure 6**. It is worth noticing that Abstract Process is used to model Abstract Service A and B but also the overall workflow that is also "not concrete". This allows using the AB Profile to specify requirements that have to be applied to the entire sub workflow, while A and B Profile are used to specify single service functionality and requirements.

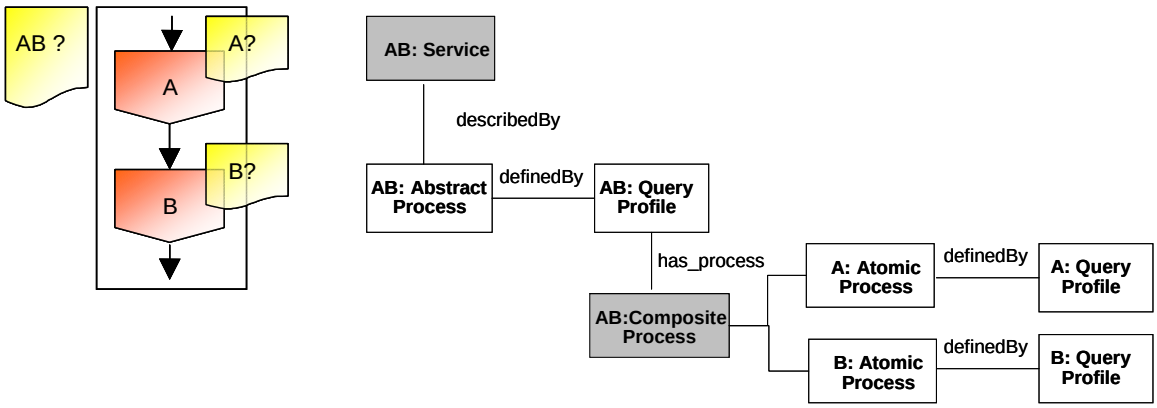

Figure 6: Abstract Workflow Instance – Model and Language Mapping

When a matching Concrete Service is found, for instance for Abstract Process B, the Query Profile is used as the bridge towards the new concrete components of the workflow (see **Figure 7**). In this way, the complete history of workflow evaluation (e.g. the Query Profile) is completely saved.

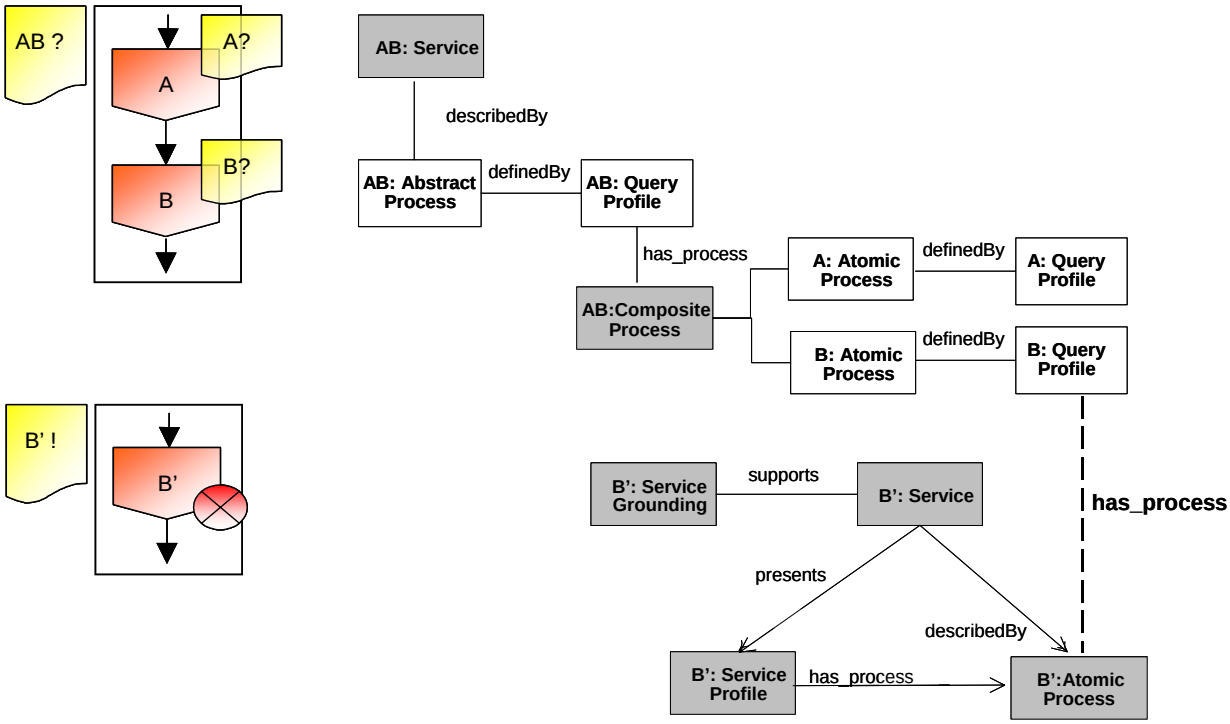

Figure 7: Concrete Workflow Substitution Sample

It is evident from these simple examples that the OWL-S structure underlying OWL-WS specification lead to a quite elaborated mapping of the Semantic and Workflow and Service Model onto the language. The result is a kind of representation that is quite complex even when information can be provided in more compact way. It must be taken into account that only an agile representation can guarantee an effective workflow management and evaluation

process. In next few months, we will address this issue mainly considering our latest experience in the workflow component development.

3.3 OWL-WS Parser Analysis and Design

3.3.1 OWL-S Related Tools Survey

In line with the original approach of not defining a completely new language but keeping in touch with existing and evolving technologies, we decided to take advantage of ongoing efforts in the semantic area. Therefore, we performed a survey of available OWL-S related tools and APIs in order to select one to be used as a starting point for OWL-WS parser development.

Directly reflecting the status of OWL-S language that is built on established W3C standard but it is not a standard itself, we found that while there are several well known instruments for managing RDF, RDFs, and OWL, tools for OWL-S management are not so widely adopted and mature yet. The most of the tools available is focused on managing ontology, likely using RDF and OWL. Among the others, the following are very popular:

- Protégé [22] is a free, open source ontology editor and knowledge-base framework developed at the Stanford University School of Medicine. Protégé supports modelling ontologies via two different editors that are Protégé-Frames and Protégé-OWL and allows exporting ontologies into a variety of formats including RDF(S), OWL, and XML Schema. An OWL-S editor has been also developed as a widget plug-in for Protégé.
- Jena [13] is a Java framework for building Semantic Web applications developed by HP Labs Semantic Web Programme. Jena provides a programmatic environment for RDF, RDFS and OWL, including an OWL API and a rule-based inference engine.

Both of these projects are very active and well supported. However, they do not provide directly OWL-S ontology API even if OWL API is surely the base to build a Service ontology.

We preferred to look for OWL-S API and we found that the two major open source packages are both based on Jena framework. They are respectively provided by:

- The Mindswap Group operating in the MIND LAB of University of Maryland Institute for Advanced Computer Studies [12]. Mindswap OWL-S API provides a Java API for programmatic access to read, execute and write different versions of OWL-S service descriptions. The API provides an Execution Engine that can invoke Atomic Processes that has WSDL groundings, and Composite Processes that uses control constructs Sequence, Unordered, and Split. Executing processes that relies on conditionals such as If-Then-Else and RepeatUntil is not supported in the default implementation but it can be extended.
- Carnegie Mellon University (CMU) [23]. CMU OWL-S API converts the OWL-S descriptions to java objects providing four different readers and writers. The resulting integrated tool is part of a Semantic Web Service IDE that is an end-to-end development environment in which a user can develop, host and use semantic web services.

We evaluated both of these APIs and we eventually chose to adopt the first one, mainly because it seems to be supported, at least for its usage, by an active community. The Mindswap APIs package also provides an integrated execution engine that has been useful for

initial experiments on the language and the enactment phase and it can also be useful for the development of the NextGRID VIM.

3.4 OWL-WS Parser Design and API

As we already said, the Mindswap OWL-S API is right now built on top of Jena. This means that the underlying data model is not part of the API and therefore it maybe changed to another library in future (OWL API being the most probable candidate). Anyway, data structures in the API has been designed closely to match the definition in the OWL-S ontology deriving all the objects from an OWLIndividual class which has accessory functions to get the value of any OWL property

In order to adapt existing Mindswap APIs to OWL-WS language, the first step is adding extensions defined for the language. Anyway, as it should be clear from the detailed presentation in previous section, the OWL-WS model is built mainly taking advantages on the OWL-S feature. Therefore, the main part of the OWL-WS API development has been, and will be more and more in future, defining data structures and functions that allow navigating and managing the model according to the semantic workflow model specification. From the OWL-WS API perspective, this has meant that:

- ***Abstract Process***, that is the only real addition provided in OWL-WS to allow defining Abstract services and composing them to specify workflows, had to be added as a native concept.
- A ***Workflow Model*** structure implementing the semantic of the OWL-WS model was defined. Basic operations to manage workflow were also developed even if several high-level functions must be further developed.

In practice, main changes in OWL-S API concerned the addition of the ***Abstract Process*** as a new type of Process class. This involved additions and changes in OWL-S vocabulary, model and management functions. In **Figure 8** modified files are listed

```
Extension and modifications

Version 1.1.1 (Mindswap OWL-S API)---------
* ADDED:    support for using OWL-WS "AbstractProcess" as a
*            native owl-s concept
* MODIFIED: file "src/impl/jena/OWLModelImpl.java"
* MODIFIED: file "src/impl/owls/OWLSConverters.java"
* MODIFIED: file "src/impl/owls/process/AbstractProcessImpl.java"
* MODIFIED: file "src/org/mindswap/owl/OWLModel.java"
* MODIFIED: file "src/org/mindswap/owls/process/AbstractProcess.java"
* MODIFIED: file "src/org/mindswap/owls/process/Process.java.java"
* MODIFIED: file "src/org/mindswap/owls/vocabulary/OWLS_1_1.java"
```

Figure 8: Files involved in the addition of Abstract Process class

Next step was building on top of the OWL-WS API, a ***Workflow Model*** class used to represent and manage workflows and services by means of a tree structure (***Workflow Tree*** class):

- The *Workflow Model* provides functions to load and save, respectively, an OWL-WS workflow description into/from the Workflow Tree structure using the OWL ontology structure and functionality provided by the OWL-S extended API.
- The *Workflow Tree* is a kind of abstract syntax tree reflecting workflow control flow structure. Tree nodes are modelled as NextGRID Service (NGService).

- An *NGService* is a structure used to effectively store, and manage, all the information related to each single type of basic OWL-WS element, that is Abstract/Concrete services and workflows. In addition, it also models auxiliary components that are Abstract and Concrete Processes to allow navigation of the OWL-WS workflow structure. An NGService contains at most information related to OWL-WS Service, Profile, Process and Grounding components but the single components are instantiated only dependent on the specific type of service that is modelled.

In this first release only basic (load/save) functionality has been developed but in order to provide and maintain a consistent and effective Workflow Model, higher-level functionality should be further added, like Workflow Substitution, Concrete Workflow Merging and Abstract Workflow Composition [3]. Design and development of top-level functionality was discussed and planned together with the Enactor development (also in WP5.3) and the Semantic Workflow Programming Tool activity performed in WP6. In fact, OWL-WS parser and workflow model are fundamental for both these components.

3.5 Language issues

This section describes the problems we encountered while using the representation language during the design and development of the enactor. Although OWL-WS extensions have provided some key mechanisms to support the needs of the Grid VIM there are some OWL-S characteristics that appeared to be problematic and needed to be resolved.

As we have seen in section 3.2, mapping the Semantic Workflow and Service Model onto OWL-WS is quite elaborated. For sake of simplicity, the first release of the enactor component is based on a simplified OWL-WS model that is basically an OWL-S model with the addition of Abstract Process. This experience has been extremely useful to highlight OWL-S shortcomings, some of them can also be encountered in OWL-WS and will have to be therefore solved in next release

3.5.1 Multiple Service Models

In principle, it is likely that each service will need to support multiple possible workflows, i.e. have multiple service models, each of which is associated with one or more profiles. This is not permitted by the cardinality constraints of OWL-S, so we had to relax these constraints in OWL-WS. Also according to the OWL-S cardinalities (see [4]) there must be only one Service per Service Grounding. So if a single endpoint provides several functions, they have to be expressed using a single Service.

These issues appear when discovering for a service implementation of an abstract process and matching its query profile to a service profile (which of the service models is used?) and also binding the abstract process to an atomic grounding (what if it is of a different service?). In both cases we may end up using multiple service models in the same workflow structure (a service).

There are several possible solutions:

- associate each profile of a service with a sub-process of the Service Model.
- break the OWL-S cardinality: a Service can have multiple Service Models.
- break the OWL-S cardinality: a Service Grounding can have multiple Services.

The decision was to associate each profile and grounding with a sub-process of the service model. Using this approach helps discovery and query mechanisms of registries [11] and simplifies the enactment process (see also Section 4).

3.5.2 Scope of OWL-S parameters

In OWL-S and consequently in OWL-WS identifiers of components and their parameters including processes, inputs and outputs must be globally unique. At the same time these components do not have a name attribute. This means that every process description in the workflow must be unique since there is a single global scope of process and parameter definitions. There are two questions:

- Can we embed a workflow in another one?
- How do we relate the names between the two levels of workflow composition?

For example when an abstract process is substituted with a workflow how do we deal with scope of names? A solution can be to implement a hierarchical environment at the enactor where each process is defined inside its own frame. This approach has been used at the developments of the enactment engine. Thus the enactor implementation has been used to deal with the lack of naming scope at the language. A simple enactor environment was developed at the prototype implementation and is described in Section 4.

3.5.3 Support of co-allocation and prioritisation

One aspect that certainly needs to be resolved is the case where multiple tasks should be co-located. This is likely to arise when a workflow-level QoS analysis indicates that there is a large data dependency between two tasks, so that it makes sense to find a single provider for both tasks, so reducing the need for large data transfers between service providers. We need to show how the OWL-WS language would support this, and determine how registry and broker functions could support selection of a single provider for sets of co-located services. This has not yet been addressed.

Prioritisation is concerned with the question of when each AbstractProcess in a workflow should be substituted by a more concrete workflow through discovery and selection. This can make a big difference to the cost-effectiveness of a workflow, e.g. by allowing the enactor to decide where the most expensive processes will run first, so it can optimise the location of less critical processes to take advantage of the best deals. An additional parameter must be added to AbstractProcesses to indicate the prioritisation.

The new OWL-WS model that results from applying all the above changes is given in the following diagram (see Figure 9). Note that the groundings can be extended to cater for different invocation models (not just Web Services), or to handle specialisations of Web Services (e.g. involving specific security models or process constraints). In the current implementation, a specialised grounding was used for GRIA 5 services, to avoid the need at this stage of defining in a general WSDL grounding the security policy and token exchanges used with those services.

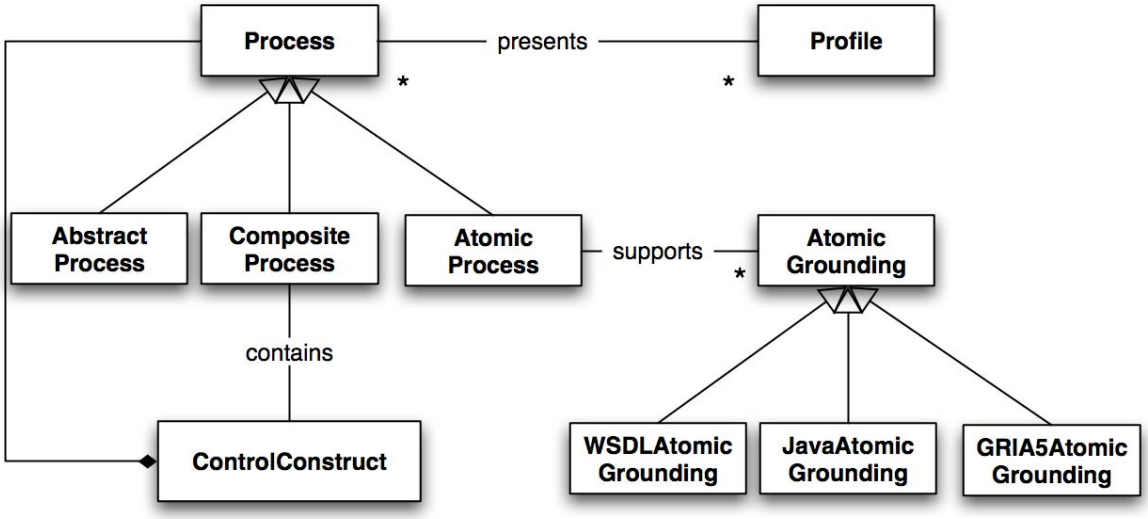

Figure 9. Relationships of the new OWL-WS Model

4 Workflow enactor

4.1 Dynamic workflow enactment

As already mentioned dynamic workflow enactment is achieved by run-time evaluation of the workflow, which aims to adapt it according to the business processes, virtual organisation policy and the environment.

The workflow enactment is based on "evaluate – apply" cycles, borrowed from functional programming. The aim of the evaluation is to replace abstract processes with concrete ones at run-time using components of the Grid VIM. This procedure may bind an abstract service description to a web service implementation or with other abstract workflows. The apply phase that follows, executes the concrete processes.

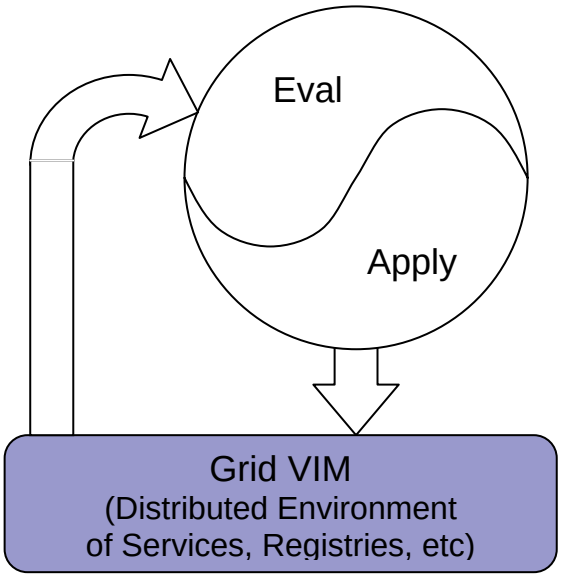

Figure 10: Abstract Task Evaluation and Application

It is clear from a given workflow what ordering (or partial ordering) must be used when executing its concrete processes. These are given by the composite processes that define explicitly the sequential, concurrent or alternate execution ordering of sub-processes, and by the data dependencies that provide implicit ordering constraints on sub-processes.
It is also clear that an abstract process must be evaluated before it can be executed. However, there are no clear constraints on when an abstract process should be evaluated in relation to the evaluation or execution of other processes. There are many options:

- abstract processes could be evaluated just before they are executed (so-called lazy evaluation);
- abstract processes could be evaluated before anything is executed (eager evaluation), so all the concrete services are known before execution begins;
- abstract process evaluation could follow the same order as execution, even if lazy evaluation is not used;
- abstract process evaluation could follow a different order from execution, if eager evaluation is used;
- different parts of a single workflow could use different strategies for deciding the evaluation order.

The main advantage of lazy evaluation is that no time would be spent evaluating abstract processes that don't need to be executed (e.g. those in unselected alternate branches in a workflow). It also means all the inputs to an abstract process are available when finding a concrete service provider, which may allow more accurate estimates of QoS requirements. However, lazy evaluation also means that one cannot be sure that a workflow is executable until it has finished executing – it may turn out that the last abstract process has no available provider, for example.

In a commercial Grid environment, the execution of processes is likely to cost money, yet the results may be of no use if that final step could not be completed. It is also possible that the choice of services for early sub-processes may constrain (due to large data dependencies or stateful control dependencies) the choices for later sub-processes. This could cause problems if the later sub-processes are the most expensive, as service providers could easily discount the smaller processes to "capture" a workflow near the beginning, so forcing the user to accept high prices for later steps in the same workflow.

The purpose of the enactor at this stage is to investigate and clarify these issues, and provide some flexibility so that different options can be tried in later experiments to manage the overall cost and quality of service for workflow enactment.

4.2 Use of OWL-WS language features

Detailed analysis of the evaluation mechanism lead to the conclusion of implementing it using the relationships from Figure 11, which are built on the OWL-S ontology and OWL-WS extensions:

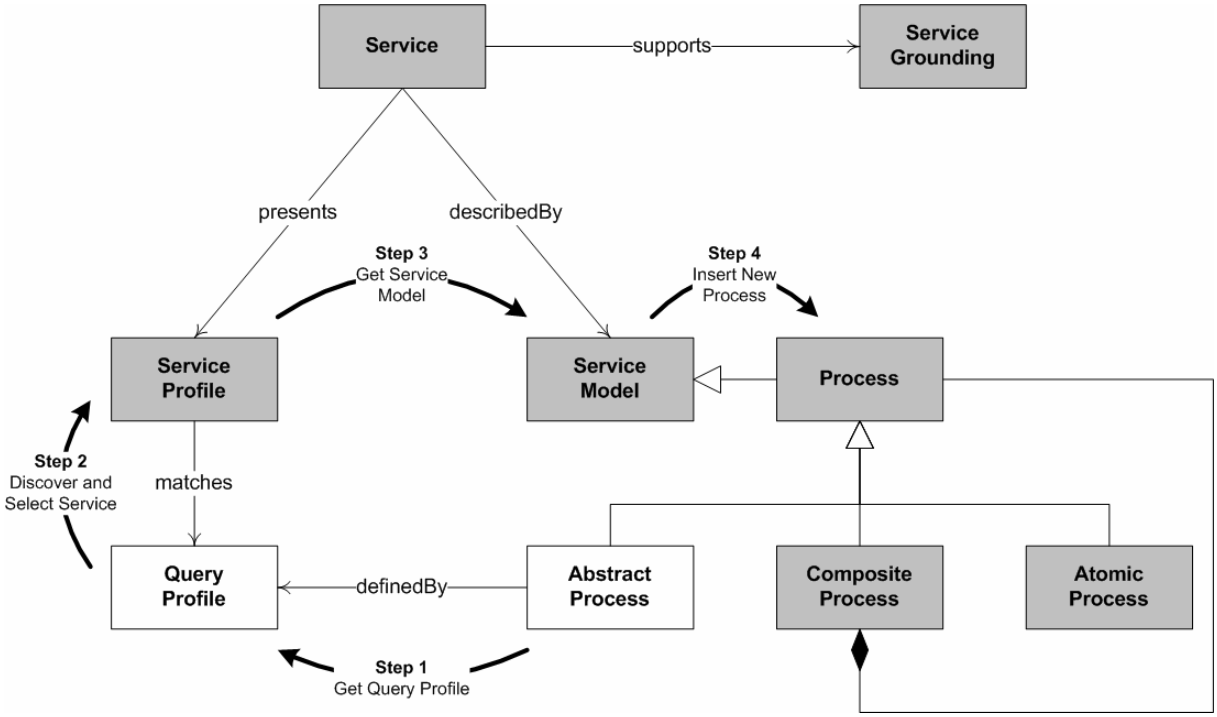

Figure 11. OWL-WS Relationships and Evaluation

The enactor starts with a Service, whose Service Model describes a workflow to be enacted. If the Service is concrete, it can traverse the Service Model, recursively evaluating Composite processes, until it finds Atomic processes that can be executed using the Service Grounding.

If the Service is Abstract then at least one of the processes is abstract and has no Grounding. Each abstract process is associated with a "Query" Profile. The "Query" profile is used by the discovery mechanism to construct queries on processes and find matching service profiles.

The procedure from Figure 10 is applied, starting from this Query Profile to discover a matching Service Profile, and inserting the Service Model corresponding to the matched Profile of this new service. The sequence of inferences to get from an Abstract Process to a Service that supports it is shown by the thick arrows in Figure 11.

Note that the use of a Query Profile is not an OWL-WS extension, but part of OWL-S, though the form of a Query Profile and the mechanisms used to find matching Service Profiles is left up to the registry providing the matching procedure. The only aspect of Figure 11 that is specifically an OWL-WS feature is the use of this procedure to substitute an Abstract Process appearing in a workflow.

Note also that if the newly discovered service may support multiple functions, it must be possible to choose between them when seeking a binding for an abstract task. This means that in OWL-WS, the Service Profile, Processes (Service Model) and Service Grounding should be able to refer to and represent each of the functions available from the service.

4.3 Relationship to Other Components of the Grid-VIM

The relationships between the components of the Grid-VIM are summarised at the following diagram (see Figure 12), which captures the VIM architecture:

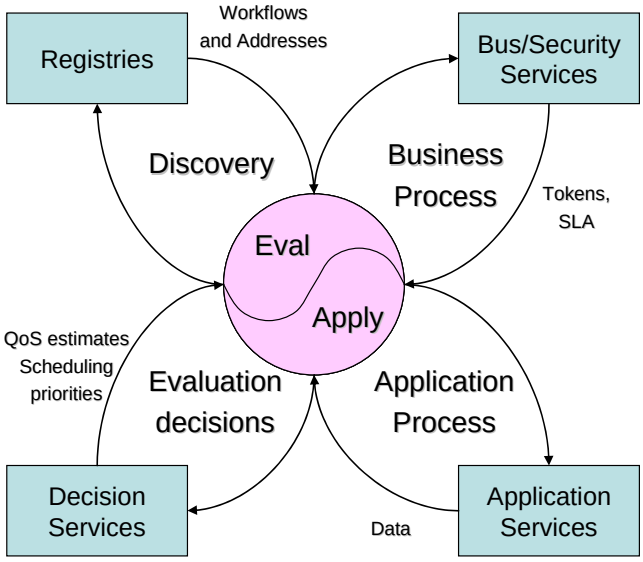

Figure 12. The enactment engine and other components of the VIM

4.3.1 QoS decision services

The relationship of the enactor to QoS decision services is fairly clear. There are two types of QoS service that can readily be incorporated:

- a workflow-level QoS service that takes an abstract workflow and assigns binding priorities or constraints to the abstract sub-processes in the workflow;
- task-level QoS services that take abstract processes plus information about their expected inputs and produce estimates of their computational and data requirements, plus information about their expected outputs.

The workflow-level QoS service provides crucial input – the workflow enactor can still work with no information about priorities, but it is obvious that intelligent prioritisation could make a lot of difference to the quality of service provider selection.

4.3.2 Registry services

The workflow enactor maintains an environment for each workflow in which bindings are stored between named entities in the workflow and their values. When the enactor finds a named data item or task, it first looks in its environment to see if the entity has already been assigned a value. The role of registry services is to extend this environment, storing several kinds of bindings:

- bound data items or workflows that are available to the enactor, as a result of previous steps in the workflow;
- pre-defined workflows that should be used for common enactment stages;
- services (or extended workflows) that could fulfil abstract tasks yet to be made concrete.

The last of these corresponds to the "traditional" service registry role, but note that the registry may offer candidate task implementations that consist of abstract or concrete workflows as well as services.
It is clear that when searching for services to fulfil abstract tasks, one should take account of the business resources already available, i.e. one should apply some of the selection criteria in the discovery step. Indeed, it may be sensible to adopt a multi-pass discovery and selection procedure, such as:

- discover free to use (e.g. internal) services, and select the best QoS if any are good enough; else
- discover services with which an SLA (i.e. resource allocation) is already in place, and select the best QoS if any are good enough; else
- discover services that can bill to an existing account, negotiate SLA with each, and select the best QoS if any are good enough; else
- discover services anywhere, negotiate SLA with each (including establishing a new billing account if that is a pre-requisite), and select the best QoS if any are good enough; else
- present the user with the best available services and ask them to decide which is best and whether to use it.

This implies that we may need a hierarchy of registries: one for internal services, one for approved suppliers (with whom accounts and possibly SLA are already in place), and one for other services.

4.3.3 Brokering services

Brokering services come in when a user doesn't have a business resource (e.g. an SLA or possibly an account) needed to access a service. In this situation it is necessary to negotiate for the required resource, and possibly to compare terms offered for access to such a resource by different possible providers of the service. A brokering service can perform the required negotiation with one or more potential providers.

In some situation, the user may not be authorised to operate the necessary business processes directly – e.g. if they are not authorised to open a new billing account, or even sign off a new SLA with an existing service provider. The most likely role for a broker service is therefore:

- to handle the negotiation of terms for new SLA (and possibly also billing accounts) with one or more existing or new service providers; *and*
- to represent an authorised agent (not the application end user) that can negotiate such things on behalf of the user's organisation.

In the first case, encapsulating the negotiations by using a broker just means the workflow enactor doesn't need to load an extra workflow for negotiation of new business resources. The full value of this encapsulation comes when the broker service (but not the user) has the power to make new business agreements. In that situation, encapsulation of the negotiating authority is essential, as it provides a point of control for that authority to regulate the actions of users, yet also allows the users to proceed with their workflows, subject only to the controls in place. The broker can also be used to apply organisational policies over how service providers are chosen (e.g. from an approved supplier list), and to ensure that business resources (e.g. SLAs) are shared through an organisational registry, so it isn't necessary to have a new SLA per employee.

The broker service should appear, along with registries (plural) and QoS decision services in the full Grid VIM binding procedure. If we do use brokers to encapsulate "authorised negotiators" to extend the user's capabilities, then our brokers will also need to support some management functions allowing the authorised negotiator to set policies on how to negotiate and select providers, and for which users.

4.4 Enactor implementation

4.4.1 Available Technologies Overview

The first attempts to produce the enactor architecture were based on Freefluo [6,7], which is the workflow enactor used in conjunction with the Taverna [8] user interface from myGrid [9]. Freefluo was chosen as the starting point because it was originally developed for myGrid by IT Innovation as an open source project, so there are no IPR barriers to its use and exploitation in NextGRID. Furthermore, although used with Taverna to process the XSCUFL [10] language, Freefluo is a separate input-language independent enactor core that has been used with other workflow languages (including a version of WSFL with semantic annotation of data flows). So it provides a candidate platform for an experimental implementation of OWL-WS.

The Mindswap OWL-S API [12] is a java library that has a parser and an execution engine. It is open-source software with a permissive MIT licence, so also in this case there were no barriers using it in NextGRID. The Mindswap library is based on Jena library [13] and is designed to parse and execute OWL-S services.

After the analysis on the language and the objectives of the NextGRID VIM we decided that it is more appropriate to use the Mindswap technology. The reason was that the OWL-S API supplies a parser and an execution engine capable of enacting OWL-S workflows, so there was no need to write or adjust to the needs of OWL-S another one. Additionally we decided that the existing functionality of Freefluo – other than that found in Mindswap - mostly focuses on providing ways to monitor and steer the workflow, which are not directly in the scope of this experiment. So our task was to extend the OWL-S API and supply features to support grid dynamics, which is the objective of this project.

4.4.2 The Mindswap execution engine

The Mindswap OWL-S API, as mentioned before, is built on top of Jena, so every resource in the API (Service, Profile, Process, etc.) has a corresponding implementation in the Jena resource model. Every resource extends the OWLIndividual class, which provides very basic functionalities to get and set object and data properties for any OWL resource.

Each service description is loaded into a separate Jena model (OWLModel class) where the model contains the imports closure of the service description, i.e. each ontology service description imports is also in that model. Individual OWLModel's can be accessed by the OWLKnowledgeBase they are a part of. An OWLKnowledgeBase class contains all the loaded OWL-S ontologies and provides the mechanisms to create a new or add one by reading its description from a specified URI. Using these mechanisms the service described by the ontology can be retrieved.

Executing a service means executing the process it has. The process should have a valid grounding specification in order to invoke the service successfully. The WSDL [14] and UPnP [15] groundings are supported by the API. A process is executed by a ProcessExecutionEngine.execute(Process, ValueMap) function where second parameter supplies the environment for the process in which input parameters are bound. This function returns another ValueMap, which contains the output value bindings.

If the service has a CompositeProcess then the structure (ContoConstruct) that this process encloses is resolved and executed. So for example, if the ControlConstruct is a Sequence, i.e. a sequence of processes connected with a dataflow, then every component (Perform class that contains a Process) in this sequence is executed and its output is fed to the input of the next component.

The API contains a package org.mindswap.wsdl that provides support for reading and executing WSDL services. Execution of OWL-S services is achieved through this package. The WSDL functionality is based on the Axis [21] package. The SOAP [16] messages are created from the string representation of XML Schema complex type.

5 NextGRID Enactor Architecture

As mentioned before the execution engine will use the evaluate-apply procedure to enact the workflows. The evaluation cycle will take place only if the processes or constructs that are to be executed are abstract, i.e. contain at least one AbstractProcess instance.

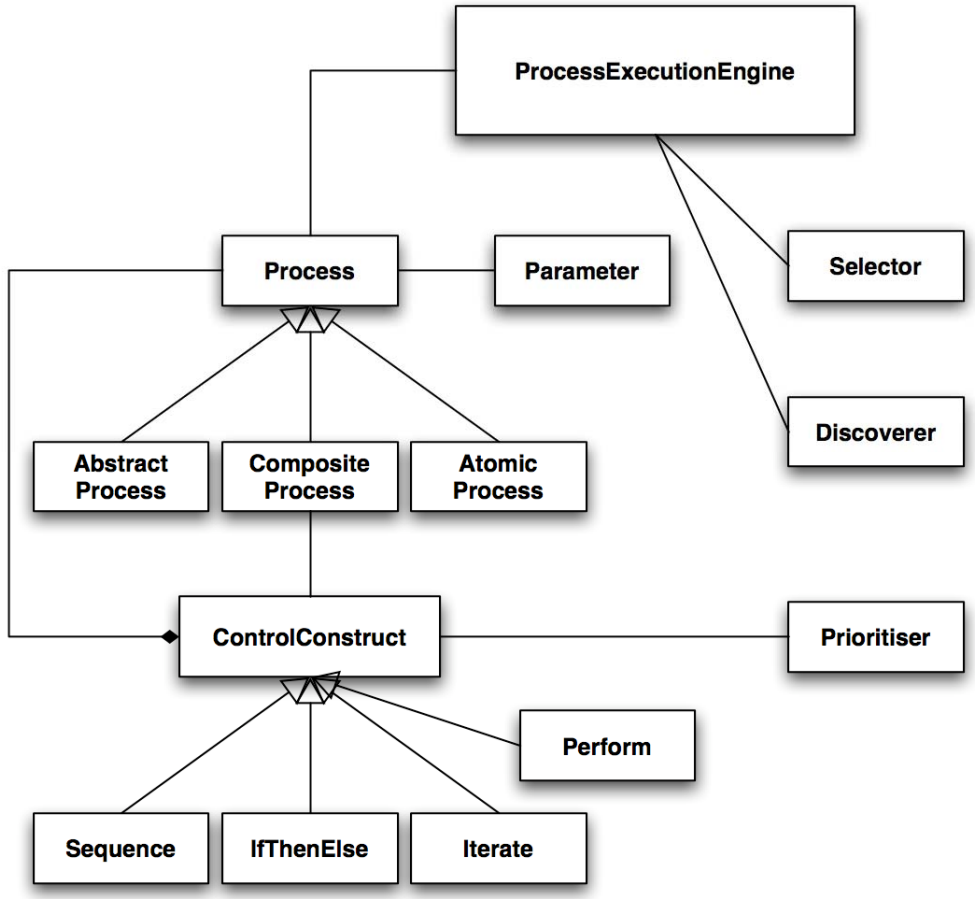

Figure 13. Enactment Engine Architecture

Before the evaluation of the workflow begins the evaluation order has to be defined. To handle this, we introduced a new subclass of the Parameter class that is called Priority. This new parameter (see Figure 14) has been added to the OWL-WS model, and support implemented in the parser (also to be used by workflow authoring tools from WP6) and in the process execution engine. The new parameter has a constant value of integer type: a zero value indicates the absence of evaluation priority, while larger values indicate successively lower evaluation priorities.

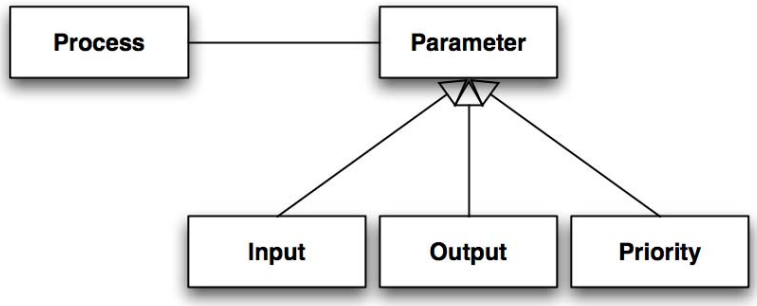

Figure 14. The parameter class diagram

Priorities may be set in advance by the author of the workflow, but the enactor also supports automatic prioritisation using the Prioritiser associated with each ControlConstruct class (see Figure 13). At present, priorities are used only in the OWL-S ControlConstruct sub-class Sequence, which represents a sequential execution order over its sub-processes. Priorities are useful here because we know all sub-processes of a sequence should be executed (assuming there is no error), so all the possible variations and tradeoffs between enactor performance, workflow QoS and cost are present. At this stage, the implementation prioritises sub-processes in the opposite order to execution, on the basis that data volume and hence costs tend to increase with each step of a workflow. In future, this method will pass information about the ControlConstruct and sub-processes to a workflow QoS service, which will provide the priorities.

Evaluation is implemented by the ProcessExecutionEngine class, and is used whenever an AbstractProcess has to be evaluated. The evaluation process consists of the discovery operation that discovers candidate bindings for the abstract process, and the selection operation, which binds it to the selected implementation. In the current implementation, the Discoverer sends the Query Profile of the AbstractProcess to a NextGRID semantic registry [11], which is implemented using Grimoires. At this stage the Selector is implemented via a SimpleProcessSelector class, which always selects the first process of the candidate list of services returned by the Discoverer. In future, the Selector will make use of QoS and where necessary Brokering services to determine which of the discovered candidate services should be used. Finally, the Process model for the selected candidate is parsed to create a (possible composite) sub-process, which is stored as the isRealisedByProcess property of the AbstractProcess. The original AbstractProcess is retained as a container for this workflow, allowing data to be conveyed easily from the outer composite process to the inner one, ensuring that there can be no conflict between names used at the two levels. This way the enactor deals with the lack of scoped naming in the OWL-S language, by effectively treating the realisation of an AbstractProcess as a separate workflow.

The evaluation procedure also depends on the type of ControlConststruct within which abstract processes are found. The “default” mechanism described above is used to evaluate Sequence constructs. For the IfThenElse construct a lazy approach is used, in which evaluation is delayed until it is known which branch should be executed. Finally in the Iterate construct, the body to be iterated is evaluated eagerly once, and not re-evaluated each time the body is executed.

The execution mechanism of the workflow has also been extended. The executeAbstract method tries to execute the “binding” process that the isRealisedByProcess property points. If this property is empty evaluation takes place otherwise the pointed process is executed. Execution passes the input of the abstract process to its realisation, and retrieves the output, writing it back into the environment of the containing process.

An Execution exception is thrown in case of an execution failure and the containing composite catches it and:

- If an atomic or a composite process has failed, it throws the exception upwards.
- If an abstract process has failed it can also throw the exception upwards or try to re-evaluate it but avoiding the same binding and execute again. This may be something we’ll have to set in the workflow execution policy (see also section 6) or choose depending on the type of composite process.

A discovery exception is thrown upwards in case of a discovery failure. There is no way to recover from this, as it indicates that there are no services capable of providing the required functionality.

A selection exception has been added and is thrown when select fails. A selection exception will also be thrown in case of an execution failure of the selected process. In both cases a reEvaluate method is called, causing the ProcessSelector to choose another candidate service, and try to complete the evaluation process using a different realisation.

A prioritisation exception is thrown in case of failure of the prioritisation service and the built-in prioritiser will be used.

6 Experiment

6.1 Experiment Components and Setup

The purpose of the experiment is to demonstrate the dynamic adaptation of workflow enactment. This means that the experiment should successfully carry out a run-time evaluation of an abstract workflow using services inside the Grid VIM and execution of the resulting concrete workflow.

In this section the software components used for the experiment are described and the experiment set-up is analysed. The components used for the experiment are:

- GRIA. A grid middleware that is the host of the concrete services, but requires a business process to be followed before concrete application services can be used.
- Grimoires Registry. It is the registry of the VIM that provides discovery services.
- QoS services for selection and prioritisation of workflow evaluation.
- The workflow enactor itself.

At this stage, no attempt was made to answer the deep architectural questions discussed in Section 2.3 – the purpose was simply to verify that the enactor and parser implemented the current OWL-WS specification effectively, so it can be used in other experiments later in the project.

6.1.1 GRIA

GRIA is a Web Service grid middleware created by the University of Southampton and NTUA in the GRIA project, based on components developed by them in GRIA and also in the EC GEMSS and UK e-Science Comb-e-Chem projects [17,18].

GRIA uses secure, "off the shelf", web services technology and is designed for business users supporting B2B functions and easy-to-use APIs. The security infrastructure of GRIA is designed to support and enforce these processes. The procedure for using GRIA services is summarised in Figure 15:

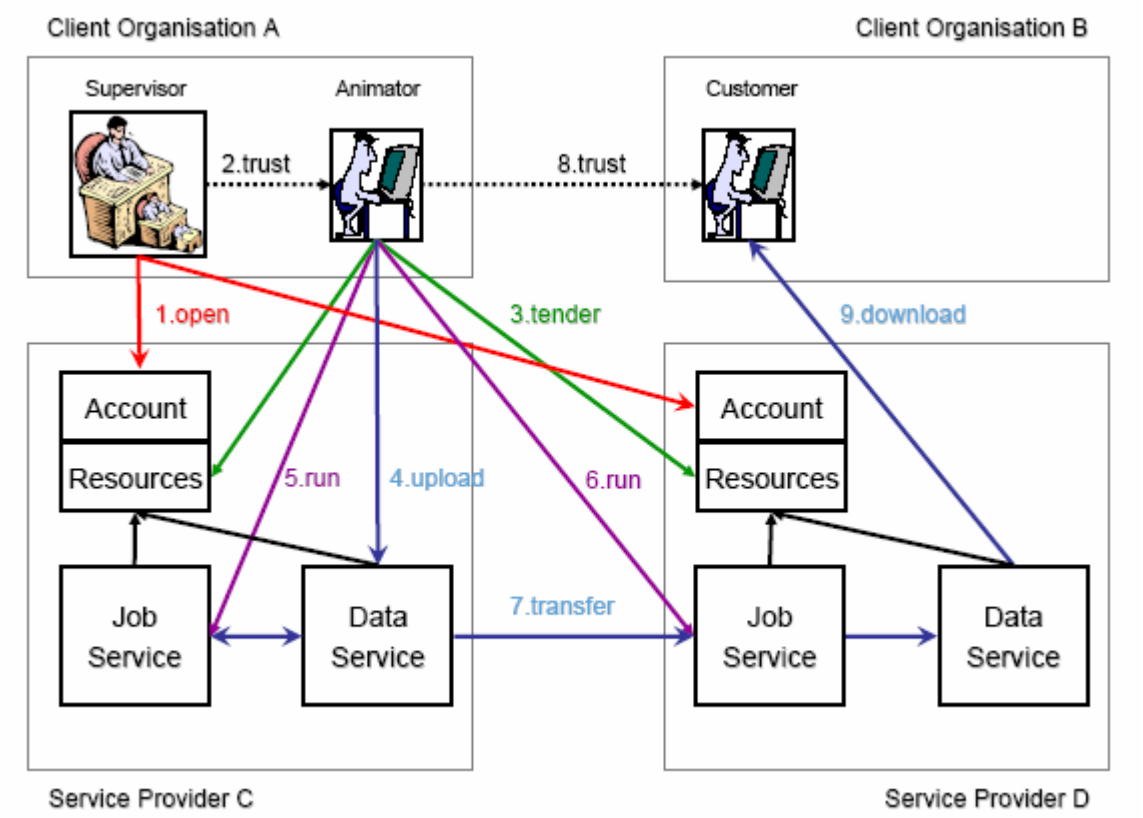

Figure 15. GRIA usage procedures

According to the diagram, first the supervisor must open an account with the service provider. Then using details provided by the supervisor, the client (in the diagram is an animator) must allocate resources using the service provider's resource allocation service. The data transfer is performed using the Data Service. The application that the user wants to use is already deployed as a GRIA job service. So the client has to set up a job using the job service. When the job is finished, the results can be retrieved from the output data store(s).

At present, GRIA application developers have to encode these business steps along with their application. The main goal of the experiment was to verify that the OWL-WS language and enactor can be used to insert the business steps into an application at run-time, by allowing the business steps to be included in the registered OWL-WS description of each service.

6.1.2 Grimoires Registry

For the purposes of this experiment an OWL-WS workflow registry has been developed [11]. It is based on the Grimoires registry that is being developed in the UK OMII Managed Programme. It implements UDDI v2 [19] and extends the UDDI data model and feature set with facility for publication and semantic query of metadata annotations of UDDI and web service entities. The Grimoires registry supports semantic query using RDQL [20].

6.1.3 OWL-WS Enactor and other Components of Grid VIM

As described in section 4 the Mindswap OWL-S API library has been extended to support OWL-WS workflow enactment and the prototype enactor was used in this experiment.

All the other components that were required for the experiment, including the QoS services (selection, prioritisation), at the time of writing this document, were under development. So for the purposes of the experiment simple implementations have been used that were built-in the enactor (see also section 4).

6.1.4 Setup

The GRIA architecture described above corresponds to the current version of GRIA [18], which at the time of writing of this document was version 4.3.1. The complexity of the full GRIA 4 business process was considered too complex for an initial test of the enactor, so we decided to use GRIA 5 Alpha2, a pre-release of GRIA 5.0 that doesn't have QoS management yet (and hence doesn't involve QoS negotiation steps), but does require a billing account (so it does still have a business process that must be followed). In future experiments we will check that the enactor can cope with more complex scenarios involving full GRIA 4.3 or GRIA 5.0 services, and potentially other Grid middleware (when these implement business processes).

Three GRIA (single-host) service provider installations have been used in the experiment, each supporting a subset of ImageMagick applications, each corresponding to a different image transformation option. On the service providers we deployed three application using different ImageMagic options and one Grimoires registry. The three applications were deployed so that two are on two service providers each, and one is available from all three (see Table 1). This allowed us to create 3 step application workflows that provide choices (or not) for the application steps, and to support different (even possibly multiple) co-location scenarios, although in this first experiment no co-allocation is demonstrated, as this is not yet supported from the decision services:

Applications deployed on GRIA5 SP1	Applications deployed on GRIA5 SP2	Applications deployed on GRIA5 SP3
PAINT	PAINT	PAINT
SWIRL	SWIRL	
	SCALE	SCALE

Table 1. Application deployment table on the three GRIA5 service providers

The basic GRIA5 business workflow is shown at the following diagram (see Figure 16). It is a composite workflow that includes an account EPR and the job service EPR. This workflow was implemented with our own custom RunJobGrounding that contains an EPR for the GRIA5 job service and was developed for this experiment. This was decided because otherwise the grounding would have been a very large and complicated OWL-S description that it would be difficult to test.

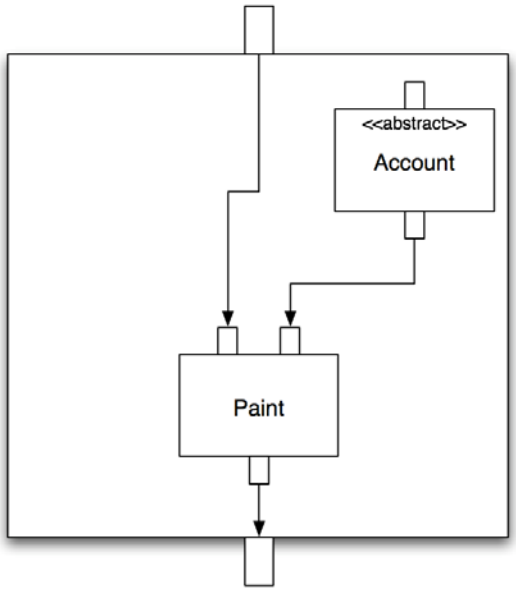

Figure 16. GRIA5 Business workflow

OWL-WS has no obvious way to represent data. There was a need to represent the user account information for the GRIA5 business workflow (see Figure 16). The approach of this experiment was to use an atomic process that is grounded with an atomic Java grounding[3] that uses the java.lang.String class to output a string.

The following diagram is a synopsis of the architecture and components used at the experiment:

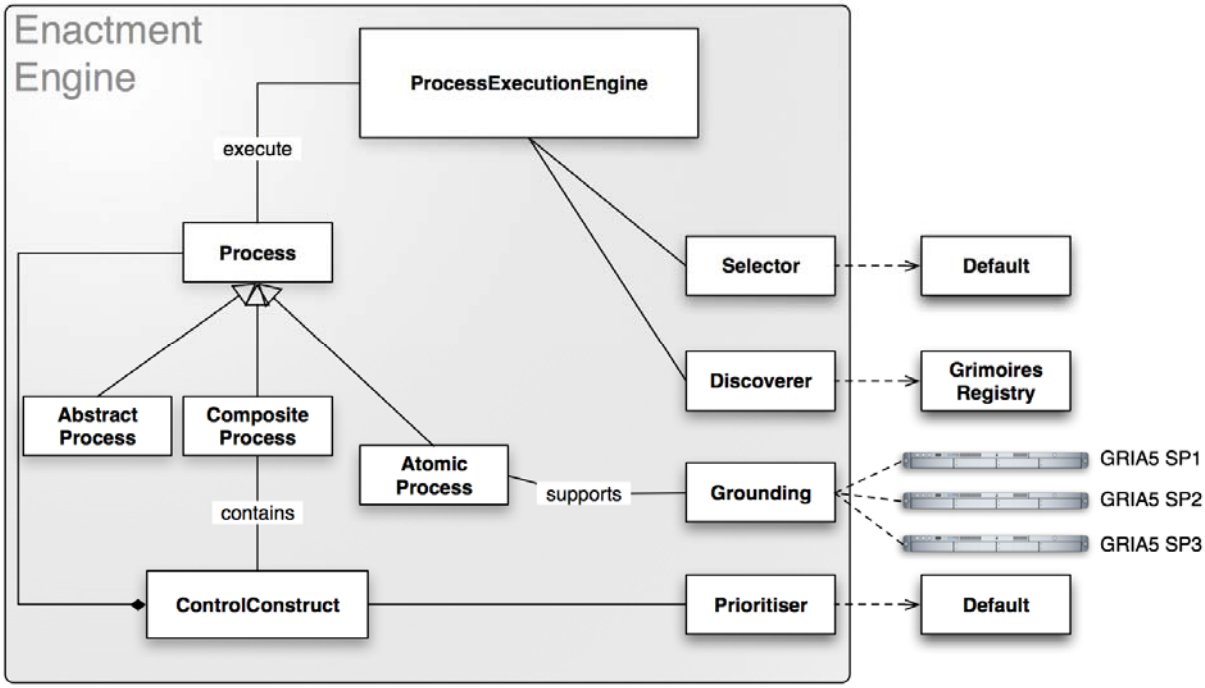

Figure 17. Experiment architecture

The application workflow consists of three processes one paint process, a swirl and a scale (see Figure 18). The registry was populated with three concrete account processes, one for each of the service providers used for this experiment.

[3] A Java Grounding is an extension to the OWL-S Grounding. It is used to "ground" or bind an OWL-S process to a Java class and operation.

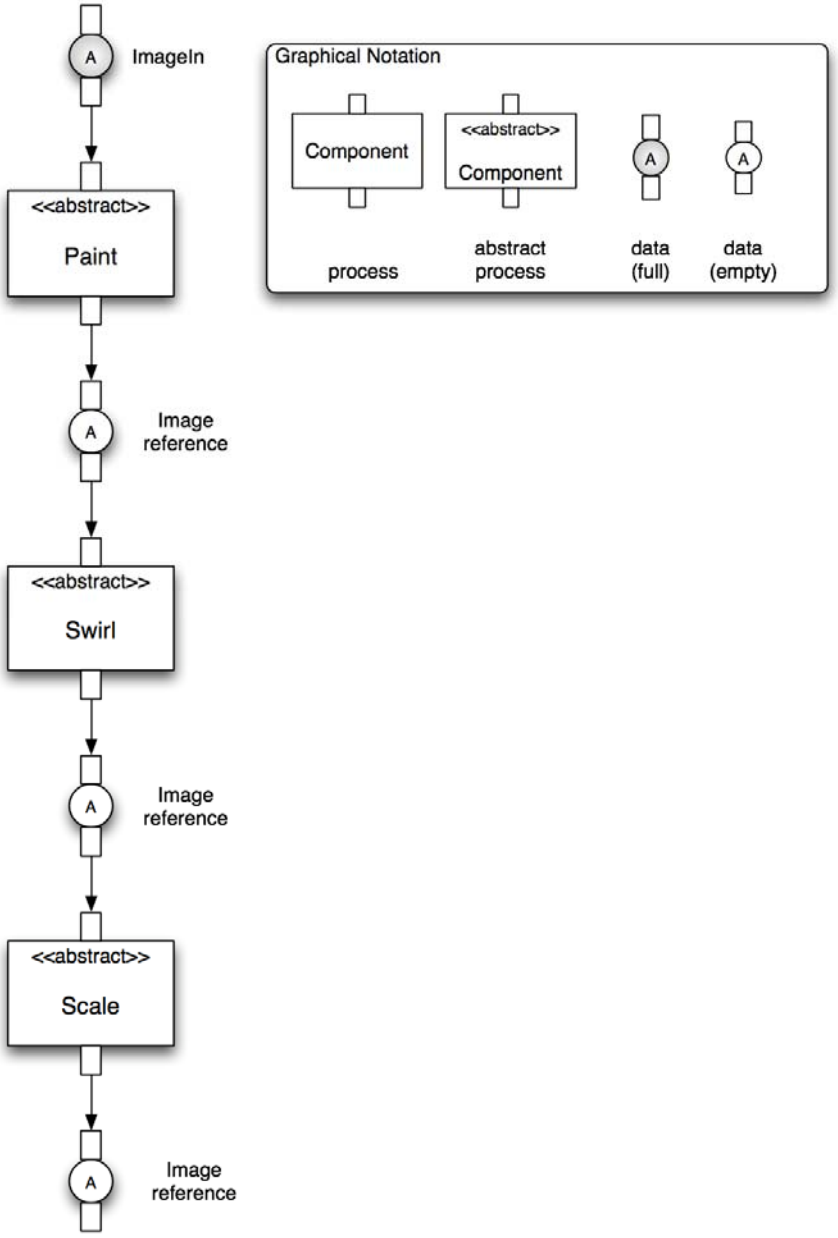

Figure 18. Application workflow

6.2 Experiment Walk Through

Enactment starts with the abstract workflow of Figure 18 being parsed and evaluated by the enactment engine. The default ProcessPrioritiser gave priorities with the inverse order from the execution one, so abstract process “scale” is first evaluated. The Grimoires registry returns the business workflow described in Figure 16. The default ProcessSelector selects the first candidate that the discovery step returned and the engine binds the selected process to the abstract one. The workflow at this stage is shown in the following figure:

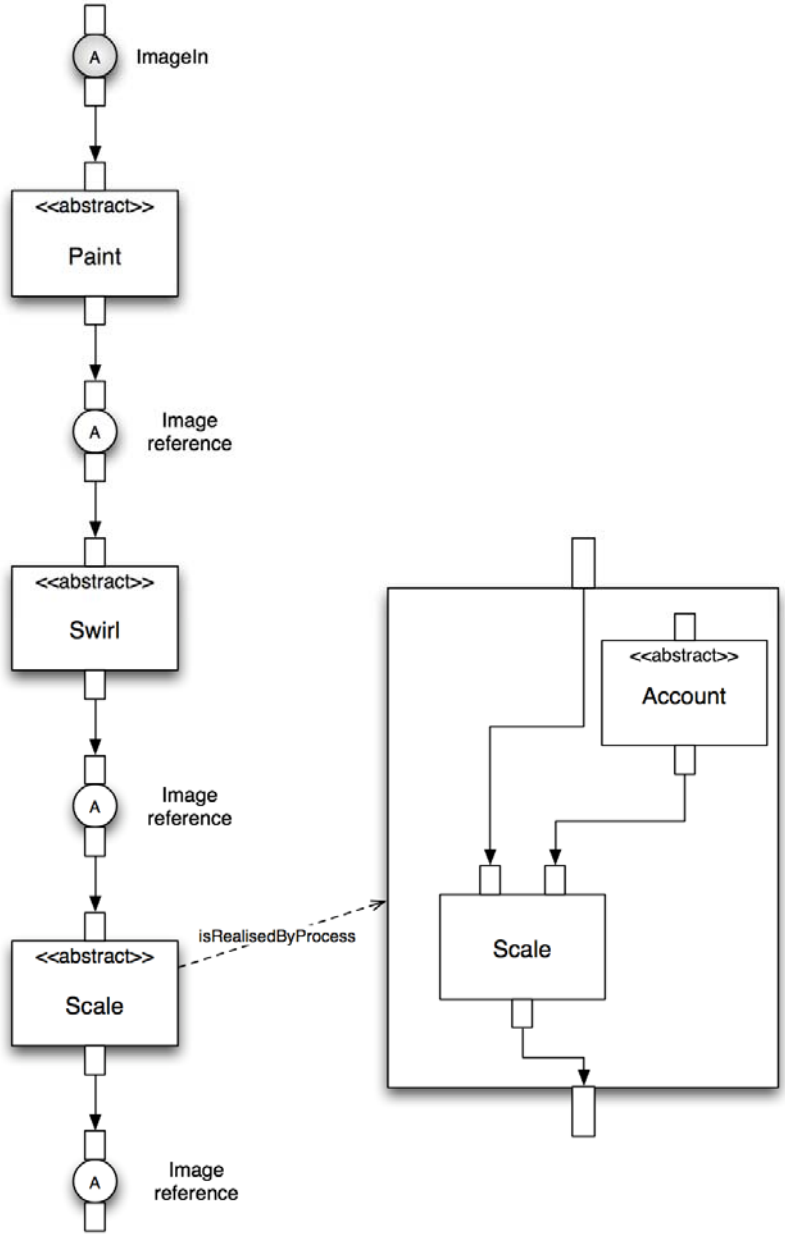

Figure 19. Evaluation of abstract process "scale"

The same evaluation procedure is applied for the abstract swirl and paint. The next step is to execute the workflow by applying "paint", i.e. execute the process that the *isRealisedByProcess* attribute points to. First the value of the input parameter of the abstract "paint" is copied to the input of the selected binding. The "Account" process is an instance of *AbstractProcess* class so it has to be evaluated and so discovery and selection steps follow. Grimoires registry will return the concrete implementation of the "account" which is a process that returns a string grounded by an atomic java grounding (see Figure 20). The string contains the account reference. The query profile for the abstract account specifies a service provider. Thus, the query of the registry to find a suitable concrete account is constrained by the service provider.

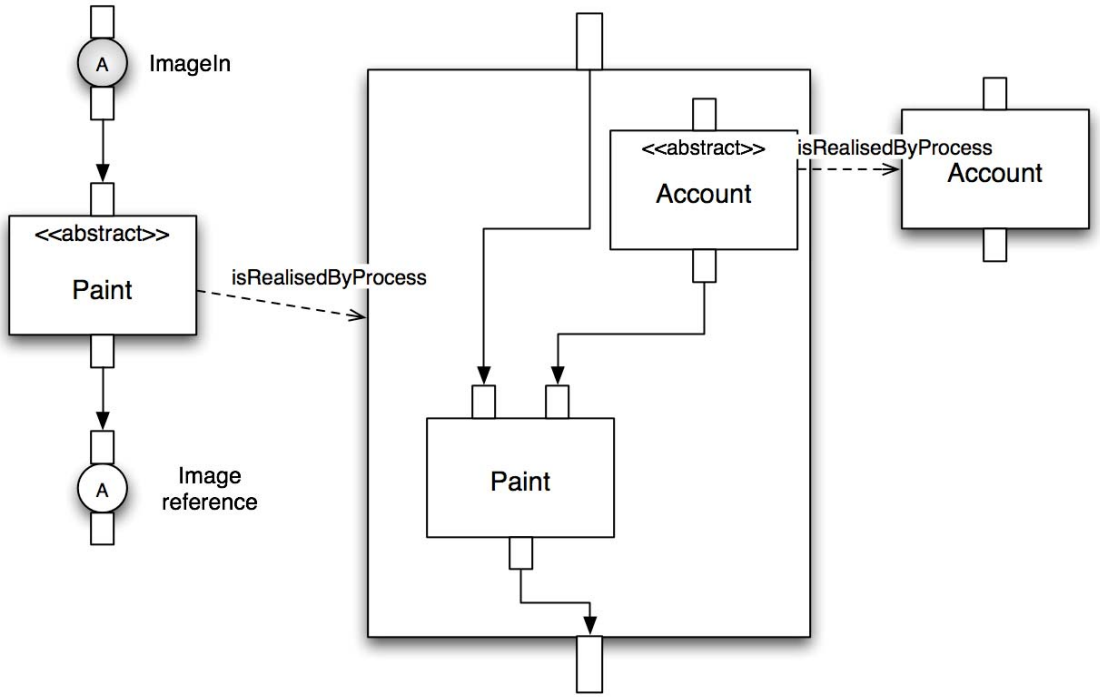

Figure 20. Evaluation of Account

The engine copies the output value of the account to the output of abstract "Account" so input of "Paint" is ready and it is executed. The output of "Paint" is then copied to the output of abstract paint process. The enactment of the workflow continues in a similar manner for the rest of the processes. The experiment finished successfully and returned the image file reference. This reference can be used to download the resulting image using the GRIA 5 Data Service [18].

The above experiment was executed in roughly 2.5 minutes. This compare with roughly 1.5 minutes for the same application workflow run by Taverna, using hard-coded GRIA 4.3 service providers. (Taverna does not yet support the GRIA 5 business processes, so a direct comparison using GRIA 5 services and a "traditional" workflow system could not be performed).

It was expected that the execution time would be worse than this, because of the overhead of parsing and evaluating the workflow and of course the time needed for the applications to run. Enactor's abilities to evaluate lazily conditional branches and upfront iterations were not tested in this first, simple, experiment, so this represents a "worst case" comparison, over very lightweight services in which the evaluation overhead is quite significant. Performance of other services used in the Grid VIM is important, should be improved for the semantic registry, and will have to be carefully controlled in the QoS and Broker services (once in use). However, the experiment suggests that the overall approach will be useful.

7 Conclusions and Future Work

In this document we presented a dynamically adaptive workflow enactment model that is based on the OWL-WS representation model. We also described the architecture of the workflow enactment engine that uses the enactment model to evaluate and execute OWL-WS workflows. Finally this document describes the experiment that we carried out to verify and demonstrate the ability of the enactment engine to dynamically evaluate and enact workflows. The experiment resulted to a successful enactment of an abstract workflow that was bound to concrete implementations on GRIA5 service providers and successfully resolved the business processes by fetching GRIA5 user account information. Apart from the adaptability to business processes this experiment was a good test for the representation language since it revealed some weaknesses.

From the language perspective, the semantic workflow model and language based on OWL-WS has been refined to provide an unambiguous semantic and to fulfil requirements from the evaluation model (e.g. to maintain complete history of workflow substitution). Shortcomings in the language (such as name scoping management,) have been identified and will be further addressed and solved in next language updates. A parser for the current OWL-WS version has been developed extending available APIs for OWL-S and has been used for a first step integration with the workflow enactor and the workflow editor, developed in the context of WP6 Workflow Programming Tool.

From the Workflow Enactment Engine perspective we plan to provide refinement of the evaluation and enactment procedures and more specifically:

- Add evaluation support for more types of control constructs, including the refinement of iterations.
- Refinement of the enactment environment of the engine.
- Refinement of the exception mechanism and the ability of the engine to recover from execution failures by selecting different bindings.
- Support for the enactment policy document where parameters like the discovery, QoS, brokering and other services will be defined and also provide some preferences and property settings on the behaviour of the engine. An example would be if to re-evaluate on execution failure or not.
- Create an engine command line user interface with monitoring and status report capabilities.

From the Workflow Representation Language perspective we plan to provide some additional refinement and possibly specification. We will focus on defining updates of the OWL-WS language, and therefore of the related parser, that can provide effective solutions for the shortcomings emerged using the representation language within the Grid Virtual Infrastructure components. The approach will be developing the workflow model underlying OWL-WS, also taking into account new proposals from standard bodies (OASIS, W3C, GGF) and existing Grid projects. Just to mention the most interesting proposals, we will take into account Member Submissions to W3C that was published after OWL-S, namely WSDL-S [24], Semantic Web Service Framework [25] and Web Service Modelling Ontology (WSMO) [26]. Any convergence between these submissions that should be reached by the renewed Semantic Web Services Interest Group will surely provide clear direction to follow in the Semantic Web Service Modelling framework. From the OASIS perspective, we will consider both results from the Semantic Execution Environment Technical Committee [27] that was established in November 2005, if any, and further evolution in the Web Services Business

Process Execution Language (WS-BPEL) specification [28]. Finally, in a more specific Grid context, even if no Semantic Grid standard is emerged till now, we will take into account any suggestion from discussion in the Semantic Grid Research Group (SEM-RG) [29] and from experiences in other Grid projects, mainly K-WFGrid [30] and OntoGrid [31].

The workflow model we are developing, primarily expressed using OWL-WS, will be as much as possible language-independent, integrating any useful technology with the aim of specifying a portable and effective representation for the management of semantic workflow components. In practice activity will be focused on:

- Discussing the issues that have been raised from the engine design and development and mainly the query profiles, the lack of variable naming scopes and data representation in the workflow.
- Investigating the possibility of finding a new representation language that better matches the model.

In collaboration with the partners inside the work-package and possibly partners from other work-packages, we intend to setup an integration plan and a second experiment that will include all the necessary components and services for all the phases of the evaluation and enactment of the workflow lifecycle.

7.1 NextGRID architecture questions addressed

As expected, none of the architecture questions posed in Section 2.3 were answered fully at this stage. The parser and enactor were used mainly to elucidate the requirements and function of the service dynamics registry component, allowing questions on that to be addressed, as described in [11].

Partial answers (or suggestions on how to get answers) were obtained as follows:

- 80) What is the relative importance of overlapping workflow standards: BPEL and ebXML from OASIS, and OWL-S from the SWS Consortium?
    - OWL-S is important because it supports semantic workflow description and service discovery. The others seem less relevant because they don't provide these features, but OWL-S is also not a perfect match to NextGRID needs, so further investigation of other specifications is needed.
- 84.1) What kind of VM should be an architectural basis for the use and enactment of workflow?
    - An evaluate-execute enactment model appears to be a good choice, though the evaluation order and prioritisation, and performance aspects need further investigation.
- 84.2) How does this relate to workflow languages and API?
    - The OWL-S workflow language is not such as good starting point as originally thought, but the OWL-WS extensions appear to work well. The MindSwap OWL-S API (plus NextGRID extensions) should be analysed to see what lessons can be learned for a more performant, directly programmable API.

The full answers to these questions lies in future work using and developing the adaptive workflow language and enactor, in combination with QoS, brokering and registry components.

References

1. Dave Snelling, Malcolm Atkinson, Sven van den Berghe, Konstantinos Dolkas, Mark Gilbert, Alastair Hume, Greg Kohring, Guy Lonsdale, Thomas Sandholm, Mike Surridge, Kostas Tserpes, NextGRID Architecture, January 17, 2005, http://www.nextgrid.org/download/publications/NextGRID_Conceptual_Architecture_V_1_5.pdf
2. Mike Surridge (ed), NextGRID Project Deliverable D5.1 Grid Dynamics Report, 08 Oct 2005.
3. Barbara Cantalupo, Ludovico Giammarino, Nikolaos Matskanis, Mike Surridge, Fabrizio Silvestri: Semantic Workflow Representation and Samples, http://www.nextgrid.org/publications.htm
4. David Martin (ed), OWL-S: Semantic Markup for Web Services, W3C Member submission, November 2004, http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/
5. Harold Abelson and Gerald Jay Sussman with Julie Sussman, Structure And Interpretation of Computer Programs,1996, The Massachusetts Institute of Technology
6. Matthew Addis, Justin Ferris, Mark Greenwood, Darren Marvin, Peter Li, Tom Oinn and Anil Wipat Experiences with eScience workflow specification and enactment in bioinformatics, http://www.nesc.ac.uk/events/ahm2003/AHMCD/pdf/108.pdf
7. Freefluo Enactor: See http://freefluo.sourceforge.net/
8. Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R. Pocock, Anil Wipat, Peter Li: “Taverna: A tool for the composition and enactment of bioinformatics workflows”, accepted for Bioinformatics Journal, 16 June 2004, doi:10.1093/bioinformatics/bth361.
9. Carole Goble, Chris Wroe, Robert Stevens and the myGrid consortium: The myGrid project: services, architecture and demonstrator. EPSRC e-Science Pilot Project myGrid. Available at http://www.mygrid.org.uk/
10. Extended Simple Conceptual Unified Flow Language (XSCUFL), see http://www.ebi.ac.uk/~tmo/mygrid/XScuflSpecification.html
11. J. Ferris and M. Surridge, NextGRID Project Output P5.2.2: Service Binding and Access Model Registration.
12. Mindswap OWL-S API. See http://www.mindswap.org/2004/owl-s/api/
13. Jena Semantic Web Framework for Java. See http://jena.sourceforge.net/
14. Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana: “Web Services Description Language (WSDL) 1.1”, http://www.w3.org/TR/wsdl
15. Universal Plug and Play, see http://www.upnp.org/
16. Simple Object Access Protocol (SOAP), see http://www.w3.org/TR/soap/
17. Shahbaz Hafeez, Ananth Krishna, Thomas Leonard, Mike Surridge, Sofia Tsasakou and Mehran Ahsant Dynamic Trust Federation and Access Control, NextGRID project deliverable P5.4.1, 04 April 2005
18. GRIA project. See http://www.gria.org

19. UDDI version 2. See http://www.oasis-open.org/specs/index.php#uddiv2

20. RDQL - A Query Language for RDF see http://www.w3.org/Submission/RDQL/

21. Axis a SOAP implementation. See http://ws.apache.org/axis/

22. Protégé. See http://protege.stanford.edu/index.html

23. CMU OWL-S API. See http://www.daml.ri.cmu.edu/owlsapi/

24. Web Service Semantics – WSDL-S, W3C Member Submission, 7 November 2005, Version 1.0, http://www.w3.org/Submission/WSDL-S/

25. Semantic Web Services Framework (SWSF) Overview, W3C Member Submission, 9 September 2005, http://www.w3.org/Submission/SWSF/

26. Web Service Modeling Ontology (WSMO), W3C Member Submission, 3 June 2005, http://www.w3.org/Submission/WSMO/

27. OASIS Semantic Execution Environment TC. See http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=semantic-ex

28. OASIS Web Services Business Process Execution Language (WSBPEL) TC. See http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel#technical

29. Semantic Grid Research Group. See http://www.semanticgrid.org/index.html.

30. K-Wf Grid FP6 Project. See http://www.kwfgrid.net.

31. OntoGrid FP6 Project. See http://www.ontogrid.net/.

Appendix

A1. AbstractProcess with a Query Profile

Here we provide an example of an abstract process with a query profile that was used at the experiment.

```
<process:AbstractProcess
    rdf:about="http://applicationWorkflow/processes/scale/process">
  <rdf:type rdf:resource="http://www.daml.org/services/owl-
    s/1.1/Profile.owl#Profile" />
  <profile:serviceName>http://it-
    innovation.soton.ac.uk/grid/imagemagick/scale</profile:serviceName>
  <profile:serviceParameter>
  <profile:ServiceParameter>
  <profile:serviceParameterName>http://profile/profile-
    type</profile:serviceParameterName>
  <profile:spDataValue>http://profile/profile-type/query</profile:spDataValue>
  </profile:ServiceParameter>
  </profile:serviceParameter>
  <profile:serviceParameter>
  <profile:ServiceParameter>
  <profile:serviceParameterName>http://profile/query/rdql</profile:serviceParamete
rName>
  <profile:spDataValue>SELECT ?processURI WHERE (?processURI <rdf:type>
    ?processType) (?processURI <process:hasProfile> ?profile) (?profile
    <profile:serviceName> "http://it-
    innovation.soton.ac.uk/grid/imagemagick/scale") (?profile
    <profile:serviceParameter> ?serviceParameter) (?serviceParameter
    <profile:serviceParameterName>
    "http://www.gria.org/gria5/process/composition/level")
    (?serviceParameter <profile:spDataValue> "0") AND !((?processType NE
    <process:AtomicProcess>) && (?processType NE
    <process:CompositeProcess>)) USING rdf FOR
    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>, process FOR
    <http://www.daml.org/services/owl-s/1.1/Process.owl#>, profile FOR
    <http://www.daml.org/services/owl-
    s/1.1/Profile.owl#></profile:spDataValue>
  </profile:ServiceParameter>
  </profile:serviceParameter>
  <process:hasInput>
  <process:Input
    rdf:about="http://applicationWorkflow/processes/scale/inputs/imagein">
  <process:parameterType
    rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://ww
    w.w3.org/2001/XMLSchema#string</process:parameterType>
  </process:Input>
  </process:hasInput>
  <process:hasOutput>
  <process:Output
    rdf:about="http://applicationWorkflow/processes/scale/outputs/imageout"
    >
  <process:parameterType
    rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://ww
    w.w3.org/2001/XMLSchema#string</process:parameterType>
```

```
</process:Output>
</process:hasOutput>
<process:hasProfile
   rdf:resource="http://applicationWorkflow/processes/scale/process" />
<profile:has_process
   rdf:resource="http://applicationWorkflow/processes/scale/process" />
</process:AbstractProcess>
```