# Semantic Support for Meta-Scheduling in Grids

*Paolo Missier*

`pmissier@cs.man.ac.uk`

*The Univeristy of Manchester, United Kingdom*


*Wolfgang Ziegler*

`wolfgang.ziegler@scai.fraunhofer.de`

*Fraunhofer Institute SCAI, Germany*


*Philipp Wieder*

`ph.wieder@fz-juelich.de`

*Research Centre Jülich, Germany*

# Semantic Support for Meta-Scheduling in Grids

Paolo Missier
`pmissier@cs.man.ac.uk`
The Univeristy of Manchester, United Kingdom


Wolfgang Ziegler
`wolfgang.ziegler@scai.fraunhofer.de`
Fraunhofer Institute SCAI, Germany


Philipp Wieder
`ph.wieder@fz-juelich.de`
Research Centre Jülich, Germany

*CoreGRID TR-0030*

April 19, 2006

**Abstract**

Co-ordinated usage of resources in a Grid environment is a challenging task impeded by the nature of resource usage and provision: Resources reside in different geographic locations, are managed by different organisations, and are by no means accessible via standardised interfaces, protocols or commands. These prerequisites have to be taken into account in order to provide solutions in the area of Grid scheduling and resource management.

In this document we propose the employment of a semantic model for Grid scheduling. The Grid Scheduling Ontology describes the capabilities and state of a scheduler, providing a machine-processable and interoperable model for the integration of local schedulers into Grid resource management. Along with the model we present a meta-scheduling architecture based on the VIOLA Meta-Scheduling Service that use ontology modelling and reasoning capabilities of OWL to provide semantic support for meta-scheduling in Grids.

## 1 Introduction

The resources needed to execute workflows in a Grid environment are commonly highly distributed, heterogeneous, and managed by different organisations. One of the main challenges in the development of Grid infrastructure services is the effective management of those resources in such a way that much of the heterogeneity is hidden from the end-user. This requires the ability to orchestrate the use of various resources of different types. In this work we focus on the co-allocation of resources to assemble a virtual machine that enables the execution of distributed workflows consisting of many parallel tasks.

### 1.1 Related Work

Recent research [1] has shown how a meta-scheduler can be employed to schedule workflows by co-allocating resources on multiple Grid nodes. A meta-scheduler is a Grid service that interfaces with multiple local schedulers which control the use of individual nodes by negotiating with them advance reservation of resources based on user

1

requirements such as time or QoS constraints. The goal of this negotiation is to determine feasible time slots in which all required resources are available for the requested start time to execute the distributed workflow.

In order to be able to participate in negotiation, local schedulers should be capable and willing to accommodate specific meta-scheduler requests (see Section 2 for details):

- Advance reservation of resources by offering job execution start and stop times.

- At least partial access to local schedules, e.g. by providing information about available time slots.

- Some control over existing reservations, e.g. to cancel or extend a reservation.

Currently only a few local scheduling systems such as CCS [2], PBS Professional [3], or LSF [4] offer these capabilities; however, more are expected to appear as there is interest of resource owners who intend to advertise their resources with guarantees for QoS to the Grid.

The main functions of a meta-scheduler include (i) allocation of a single resource for a single application for a fixed period of time, (ii) co-allocation of multiple resources for the same fixed period of time for single or multiple applications, (iii) allocation of multiple resources for multiple applications for different fixed periods of time, and (iv) allocation of dedicated resources for either of the cases above.

The prototype *Meta-Scheduling Service* described in [1] currently realizes functions (i) and (ii). The scheduling algorithm is based on a multi-step negotiation process, involving the pre-selection of suitable local schedulers, the acquisition of feasible start times from them, selecting resources, and a confirmation of the available start times from each of the schedulers involved.

The Meta-Scheduling Service interfaces with local schedulers through dedicated adapters (as depicted in Fig. 1) that hide the heterogeneity of the schedulers' native interfaces. These adapters offer a uniform set of abstract operations to the meta-scheduler which include requesting available start time slots for jobs, submitting scheduling requests for a specific time slot, and requesting the state of the current reservation. Meta-scheduling requests are communicated by client applications using WS-Agreement [5] while the adapters forward local requests using proprietary commands. This architecture allows an easy integration of meta-scheduling capabilities into existing Grid environments.
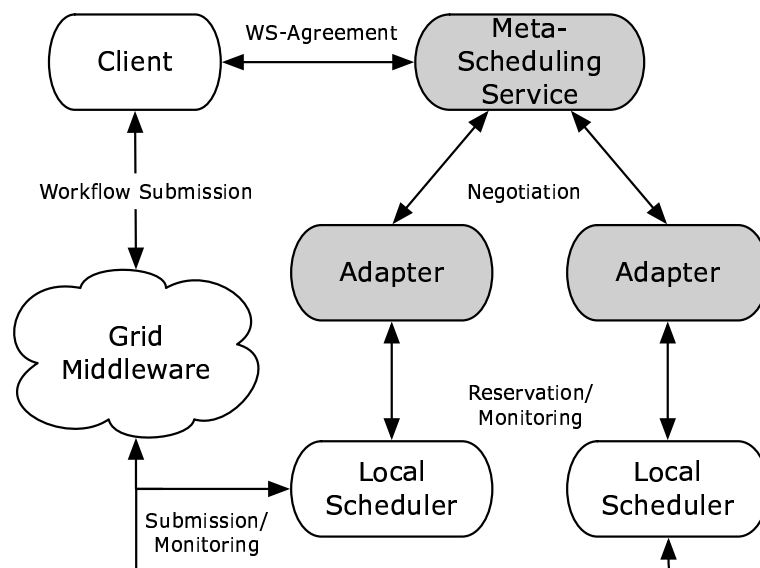


Figure 1: High-level meta-scheduling architecture

## 1.2 Motivation to Employ Semantic Models

One shortcoming of the current architecture is that, while the adapters provide uniform operations, no shared data model is available to describe a scheduler's set of capabilities and current schedule state. For example, there is no explicit and shared definition of scheduling concepts like *time slot* or *schedule queue*, or of capabilities like *time slot*

*reservation change*. Instead, these concepts are left implicit in the implementation of the adapters, which only expose a simple set of scheduling operations.

This makes the adapters not only heavy-weight, but also rather inflexible: all possible interactions with a local scheduler have to be defined at the time of implementation. Assumed that a modified version of the scheduling algorithm needed to query the schedulers' ability to modify the time slot allocation after the initial reservation (a feature that was previously not needed). At present, the only option available would be to accommodate the new feature to upgrade the adapters (all of them) in order to support the new query.

In this paper we introduce a novel approach to extensible meta-scheduling, based on the definition of a shared, explicit and extensible vocabulary to describe a scheduler state as well as its capabilities. We show how such a shared information model for scheduling concepts supports the negotiation process in a more flexible and adaptable way than it is currently possible.

Our approach is inspired by a number of recent initiatives towards the design of *semantic models* for describing Grid resources [6], mainly for interoperability purposes [7, 8, 9]. Common to all these efforts is the explicit representation of knowledge regarding available resources, encoded in such a way as to make it machine-processable.

Following the same principles, but on a more limited scope, we have developed a lightweight semantic model, called the Grid Scheduling Ontology (GSO), to describe the capabilities and state of Grid schedulers. The ontology definition process has recently been described in [10] and is based partially on the Grid Scheduling Dictionary of Terms and Keywords [11].

Along with the model we also present an enhanced meta-scheduling architecture and show how it can improve support for meta-scheduling algorithms and negotiation processes. The ontology modelling and reasoning framework offered by the OWL semantic modelling language [12] provides the necessary functions. Specifically the GSO includes a collection of scheduler classes, where each class is defined in terms of a set of underlying capabilities, for instance the ability to expose the current schedule, to accommodate changes in a reservation, and so forth.

Using a *OWL DL reasoner* [13], individual schedulers whose capabilities can be expressed using the terms in the ontology can then be automatically classified as belonging to one or more of the scheduler classes. This classification is then exploited by the meta-scheduler, as described in detail in Section 3.

## 1.3 Reminder of the Paper

The reminder of the paper is organised as follows. In Section 2 we present the requirements and use cases that provide basis for the knowledge modelling activity. The semantic model itself is described in Section 3, followed by the proposed implementation in Section 4. An overview of further developments for this work concludes the paper.

## 2 Requirements for the Scheduling Domain Knowledge Model

The introductory section already listed three general requirements that define the meta-scheduling environment our work is based on. The fulfilment of these requirements is currently realised through adapters which provide an abstraction level between local schedulers and the Meta-Scheduling Service. This approach has obvious restrictions (as reported in Section 1.2) and we therefore suggest the definition of a scheduling domain vocabulary. Since this approach makes a thorough knowledge acquisition process necessary, we re-visit and examine the original meta-scheduling use cases in this section.

### 2.1 Resource Pre-selection Use Case

Many Grid resource management and scheduling scenarios include a resource pre-selection phase where resources are selected as candidates for the actual scheduling process based on static properties [14]. "Static" in this case refers to properties which do not change from the time the resource request is submitted until the work is finished. Such properties are e.g. the operating system of a compute resource or the maximum bandwidth of a network connection, but also the capability of a resource management system to support meta-scheduling.

According to [1] a local scheduler/resource manager has to provide the following two functions to support meta-scheduling:

1. To schedule a reservation at a fixed date and time for a well-defined period of time (*Advance reservation*).

2. To provide an aggregated overview of the usage of the managed resource between now and a well defined date and time in future (*Usage preview*).

The first requirement is a binary one: If the local scheduler fulfils it the managed resource will be able to participate in a meta-scheduling process, if not, the resource will not be pre-selected. The fulfilment of the second requirement has implications on the scheduling algorithm: ideally the meta-scheduler receives all information from the local schedulers it needs to execute a co-ordinated schedule for all resources involved. But even if a local scheduler does not provide any information about the future usage of the managed resource, the meta-scheduler (or the entity that pre-selects resources) may decide to include it into the scheduling process given that requirement 1 is fulfilled.

The vocabulary should therefore provide answers to the following questions:

- Does the local scheduler provide a reservation function (to be answered yes or no according to requirement 1 above)?

- Does the information provided by the local scheduler with respect to the future usage of the resource fulfil the requirement of the meta-scheduler (according to the requirement 2 above)?

## 2.2 Schedule Query Use Case

This use case extends the second requirement of the previous use case.

Once the resource pre-selection is finished the local schedulers involved in the meta-scheduling process are queried for a resource usage preview (provided that they deliver this kind of information). With the current adapter in place it is no problem for the meta-scheduler to retrieve the usage preview in the required format since the adapter converts the preview information into the format needed by the meta-scheduler. Assuming that we want to design the adapter as generic as possible, which ideally means independent of the local scheduler, it is necessary to provide meta-data to convert the local scheduler's parameters to the format consumed by the meta-scheduler. One of the local schedulers integrated in the previously described meta-scheduling environment is the EASY scheduler [15]. It provides commands to, inter alia, return the current queue (`pq`), reserve e.g. 10 nodes for 5 minutes for an interactive job (`psubmit -n 10 -i -t 5`), show an estimation when the jobs in the current queue will be executed (`pwhen`), or give a preview of the free nodes (`prevlist`).

The vocabulary representing the scheduling domain should therefore help to answer questions like:

- What is the total estimated run time of all jobs under the control of the local scheduler?

- What is the status of a certain queue?

- What is the first possible date and time a certain job can be scheduled on the managed resource?

## 2.3 Alter Reservation Use Case

Although the current implementation of the adapter allows the cancellation of a workflow we did not examine the different states of a reservation (and their potential alteration). Nevertheless we added a capability to the semantic model which captures the notion of reservation state change. Before we closely examine and model this capability in detail in a next version of the vocabulary we want to implement an environment as described in Section 4 to validate the model based on the first two use cases. This will provide us with the foundation to meet the requirements of this use case.

# 3 A Semantic Model for Grid Scheduling

As outlined in the introduction, a meta-scheduler is able to negotiate resource allocation with local schedulers that are capable of providing advance reservation of resources (requirement 1), and that optionally allow at least partial access to the local schedules (requirement 2), and allow some control over existing reservations (requirement 3). In this section we describe the semantic model called Grid Scheduling Ontology which we use in our new meta-scheduling architecture to support these functionalities.

The negotiation process relies on a registry of available local schedulers, in which each scheduler is described according to the common semantic model. The meta-scheduler uses the registry to pre-select schedulers, as well as to inquire their state throughout the reservation and resource allocation process.

The model is defined using the OWL-DL ontology language [12], the standard semantic web language. OWL-DL allows the definition of classes, relationships among the classes (called *object properties*) as well as individuals.[1] Classes can be organised into hierarchies (using the *is-a* built-in relationship), and a class can have any number of parents. Individuals may be instances of multiple classes. We write $x \in C$ to indicate that individual $x$ is a member of class $C$.
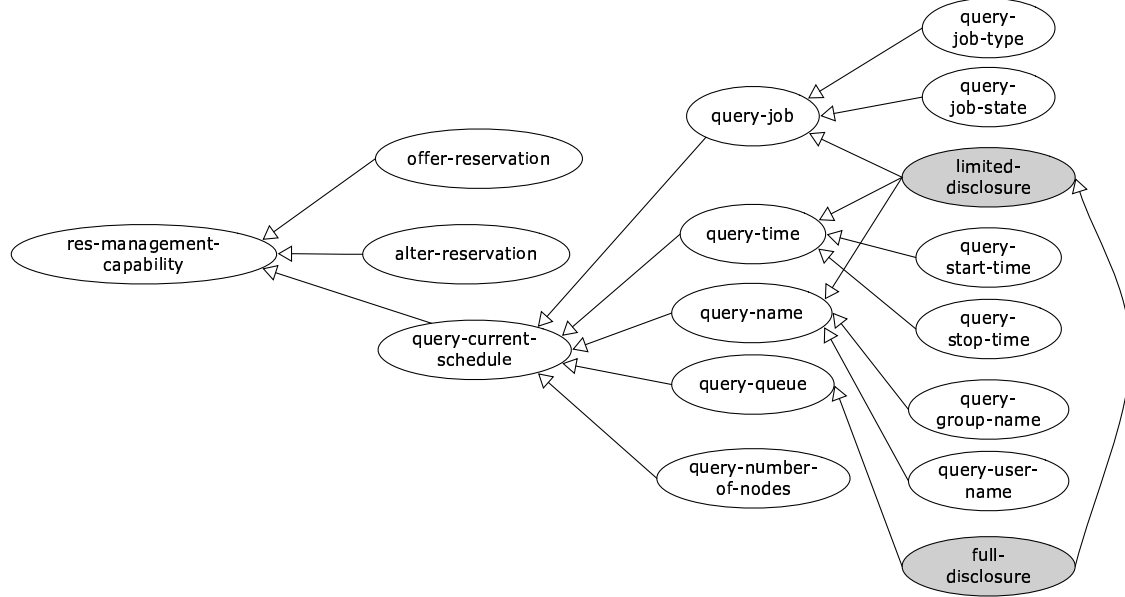


Figure 2: Fragment of the capability hierarchy (some nodes are not expanded)

At first we introduce a hierarchy of classes to model *resource management capabilities*, as shown in Fig. 2. The capability `offerReservation` corresponds to the first of our requirements: a scheduler not providing it cannot take part in the advance reservation negotiation. Additionally, we model the ability to query the current scheduler as a tree with multiple levels of precision, and the ability to alter reservations that have already been made (this node is not expanded in Fig. 2). Other branches of the hierarchy, which deal for instance with the access control mechanisms enforced by schedulers, are not shown. Note that some of the classes are defined in terms of other classes using OWL class construction operators. For example, `limited-disclosure` is defined by composition, as the capability to request the job names, the job types and the submission times from a schedule.

As a next step we introduce a classification for local schedulers, rooted at the top-level `scheduler` class. The property `scheduler-has-capability` allows us to define various subclasses of schedulers in terms of capability sets.

Consider the schedulers classification hierarchy shown in Fig. 3. Each of the classes in this hierarchy is defined in terms of other classes and properties in the ontology, using OWL-DL's class definition operators. For instance, `schedule-disclosing-scheduler` is the class of all schedulers whose set of capabilities includes at least the ability to query the current schedule. Note that the type of query that is allowed (i.e., queryJob, queryTime) is not specified. Therefore, any scheduler whose capability includes at least the generic `query-current-scheduler` class, is a `schedule-disclosing-scheduler`. Using these operators, it is easy to define classes that correspond exactly to the set of schedulers which are eligible for negotiation in the context of meta-scheduling.

In general, OWL-DL allows classes to be defined as a set of necessary and sufficient conditions, as in the example above.[2] An `AR-capable-scheduler`, for example, is any scheduler that, among other capabilities, offers advance reservation, and for this reason satisfies our requirement 1 property.

---

[1] Additional features of the language will be introduced when needed as part of this description.
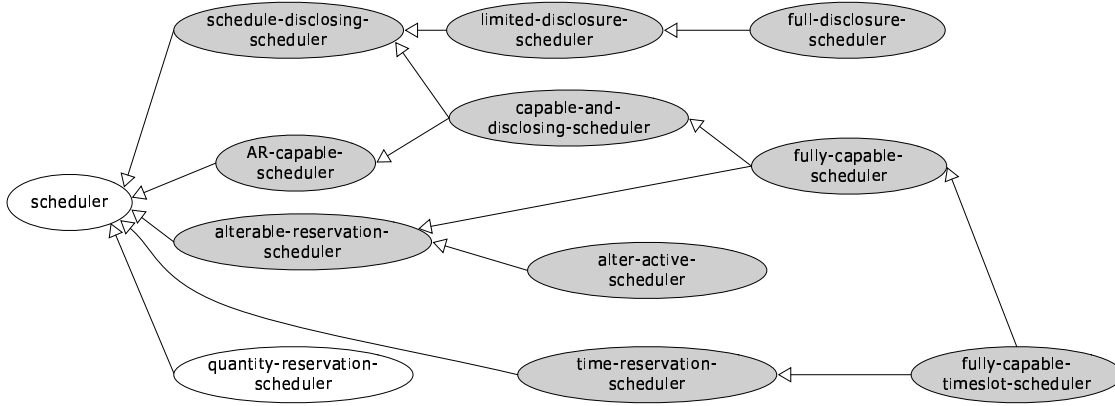[2] Formal OWL terminology is avoided in this paper for the sake of readability.

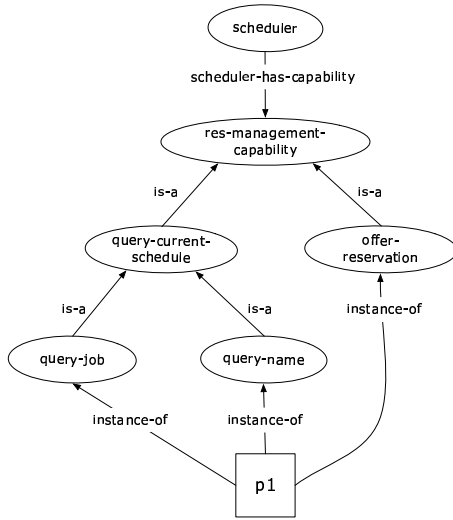Figure 3: A classification of schedulers



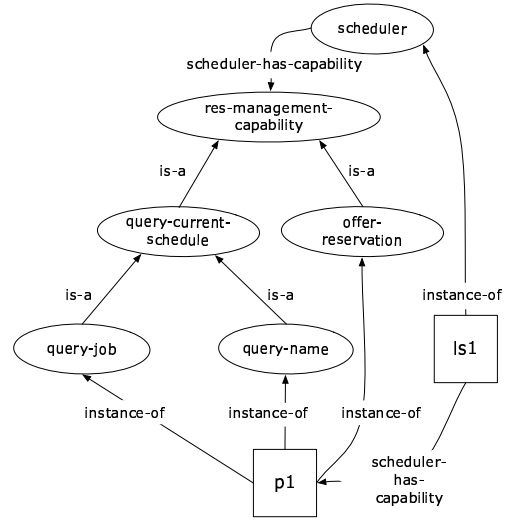Figure 4: A scheduler profile



Figure 5: A scheduler with a profile

The next example of a scheduler class highlights an important feature offered by the OWL-DL language. The `capable-and-disclosing-scheduler` class represents all schedulers whose capabilities include *both* `offerReservation` and `query-current-schedule`. In Fig. 3 this class appears as a subclass of both `AR-capable-scheduler` and `schedule-disclosing-scheduler`, although these *is-a* relationships are not part of the definition. The OWL-DL operators are defined in such a way that it is possible to perform some specific type of reasoning on the definitions of classes and individuals. In particular, an OWL-DL reasoner [13] computes the set of most specific *is-a* relationships for a collection of classes defined using necessary and sufficient conditions as shown above. Thus, the hierarchy shown in Fig. 3 is an example of *inferred* classification that has been computed from the class definitions just given.

Let us now return to the definition of scheduler classes for meta-scheduling. A *scheduler profile* is any individual $p_1$, which is an instance of one or more capability classes, for instance `offerReservation`, `queryJob`, or `queryName` (see Fig. 4). If we introduce an instance $ls_1$ of `scheduler`, and assert that $ls_1$ has capability profile $p1$ (Fig. 5), we can leverage the OWL-DL reasoning capabilities again, this time to infer the *most specific classes* of which the individuals of the ontology are instances. In other words, can we say that $ls_1$ must be a member of some specific subclass of `scheduler`, given that it has capabilities $p1$, and that `scheduler`'s subclasses are indeed defined in terms of sets of capabilities? In this case, the reasoner is indeed able to infer that $ls_1 \in$ `capable-and-disclosing-scheduler`, because $p_1$ includes both `offerReservation` (necessary for any `AR-capable-scheduler`) as well as two schedule querying capabilities that are more specific than the required generic resource management capability `query-current-scheduler` (Fig. 6).
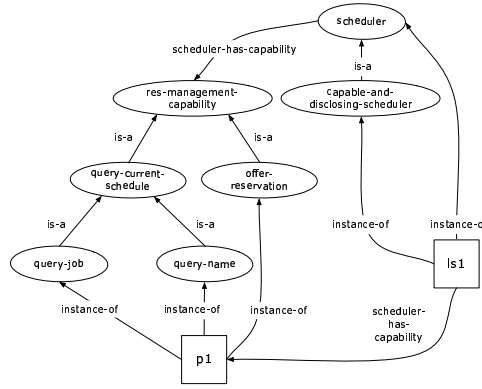
Figure 6: Classified scheduler

In practice, using this model and the associated reasoning features, we are able to obtain a classification of schedulers on the grounds of capability profiles. This automatic classification enables a meta-scheduler to retrieve all the schedulers of interest with minimal effort, simply by querying the model for all instances of one or more specific classes. As a more complete example, consider the following assertions for a set of capabilities and schedulers:

- $p_1 \in \texttt{offerReservation} \sqcap \texttt{queryJob} \sqcap \texttt{queryName}$

- $p_2 \in \texttt{queryTime} \sqcap \texttt{alter-from-booked} \sqsubseteq \texttt{alterReservation}$

- $p_3 \in \texttt{offerReservation}$

- $ls_1$ has capabilities $p_1$;

- $ls_2$ has capabilities $p_2$;

- $ls_3$ has capabilities $p_3$;

- $ls_4$ has capabilities $p_2$ and $p_3$.

With these definitions, the reasoner computes the schedulers' classification shown in Fig. 7. A query for all AR-capable schedulers would now return $\{ls_3, ls_1, ls_4\}$, while the alterable-reservation schedulers are $\{ls_2, ls_4\}$.
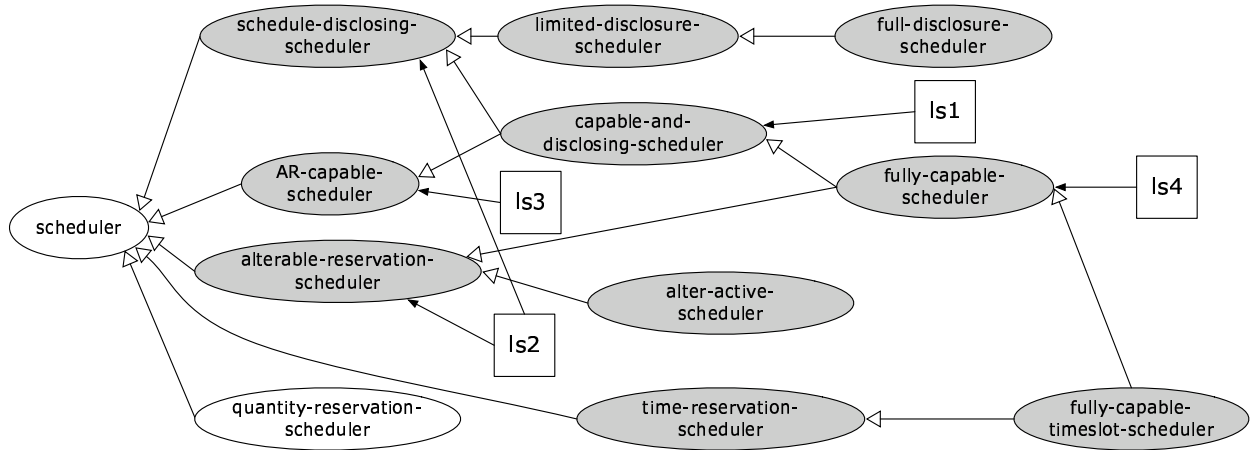


Figure 7: An automatic classification of schedulers

# 4 Environment for Semantic Exploitation

The idea of providing Web and Grid services with a semantic description in order to facilitate their semantic discovery is not new. In this section, we outline an architecture to realise our model, based on the recently proposed S-OGSA architecture [16]. S-OGSA identifies key Grid services that enable the collection and exploitation of semantics within a Grid architecture, and prescribes patterns of interaction among these services.

## 4.1 Architecture

An instantiation of an S-OGSA architecture for the meta-scheduling usage scenario is presented, at a high level, in Fig. 8.

Two types of knowledge characterise local schedulers:

- Their capabilities with respect to reservation management, as discussed earlier. This knowledge is expected to be fairly stable in time, and independent of the current scheduling activity.

- A representation of the current scheduling activity which includes the current advance reservations that have been accepted and their states. The meta-scheduler must consider that such dynamic information may not be available, as not all schedulers will be configured to provide it. If available, however, it allows a meta-scheduler to pre-select resources based for example on the current scheduler workload.
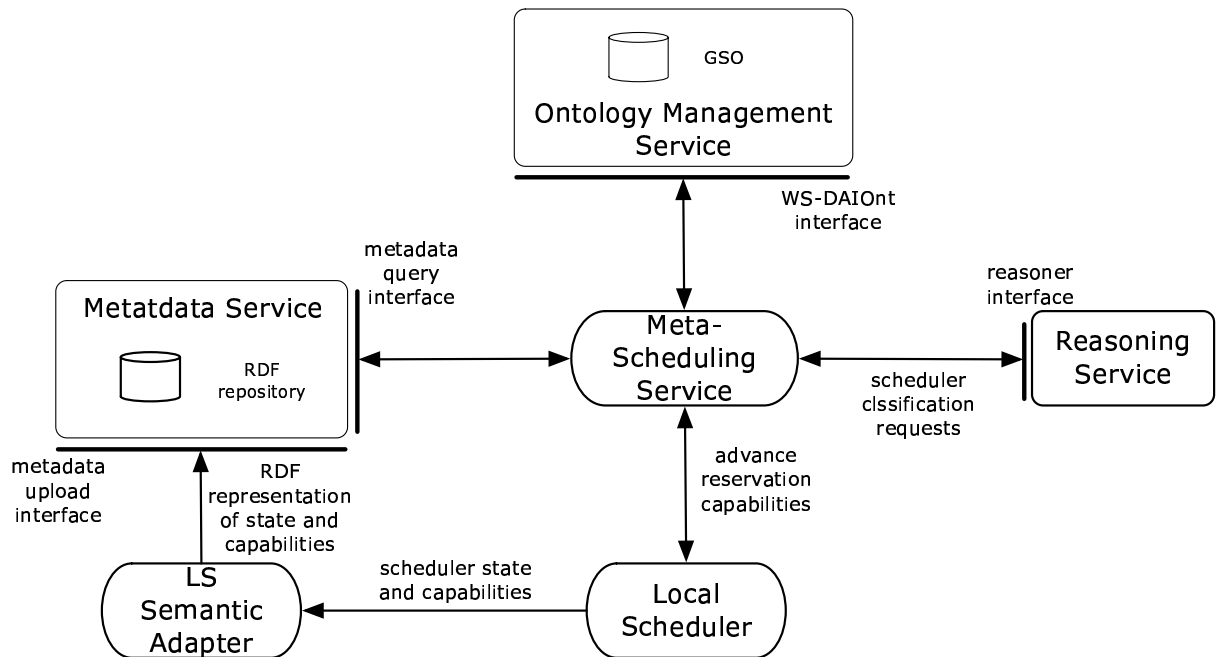


Figure 8: Architecture for exploitation of semantics by the Meta-Scheduling Service

All this knowledge is encoded using the RDF format [17], a W3C standard for describing semantic annotations, and stored in a *Metadata Service*. The meta-scheduler may query the metadata service using an S-OGSA-compliant Web Service interface, in order to obtain the capability profiles and current state of registered schedulers. An *Ontology Management Service* can be used to obtain the latest version of the Grid Scheduling Ontology in a location-transparent way. At this point, the meta-scheduler holds both the individuals (in RDF format) and the class definitions (in OWL) required to carry out the inferencing process described in Section 3, using a separate *Reasoning Service* as shown in Fig. 8.

## 4.2 Implementation

The transition from the current VIOLA meta-scheduling architecture, presented in section 1.1 (shown in Fig. 1), to an architecture which integrates the semantic model (as shown in Fig. 8) can be done smoothly in several steps without breaking the overall architecture. The following steps have to be performed:

1. Define and implement the Metadata Service (MS) with a query interface for the Meta-Scheduling Service and a metadata upload interface for the Semantic Adapter of the Local Scheduler (SALS). The MS also implements the RDF repository that contains the RDF representation of state and capabilities of the different local schedulers.

2. Define and implement the SALS which converts the scheduler state and capabilities, transforms them into an RDF representation and uploads then to the RDF repository. The SALS is needed for those local schedulers - currently all - which are not able to provide the metadata. Future schedulers may have this capability and will then be able to upload their metadata directly to the RDF repository.

3. Implement the Ontology Management Service (OMS) based on WS-DAIOnt [18] as defined by the OntoGrid project [19].

4. Add new interfaces to the Meta-Scheduling Service in order to query the metadata interface to obtain scheduler metadata of the local schedulers and to access the Grid Scheduling Ontology via the OMS. In addition interfaces are needed to submit scheduler classification requests to and receive the classifications needed for the pre-selection from the Reasoning Service.

5. Add logic to the meta-scheduler that allows to pre-select appropriate local schedulers based on their capabilities and actual state, and to use the metadata to negotiate the advance reservation with the pre-selected local schedulers.

6. Remove the adapters as they became redundant with the previous step.

# 5 Future Perspectives

Once implemented we will evaluate the GSO-based architecture and compare it to the solution presented in Section 1.1. Since the current implementation of the VIOLA Meta-Scheduling Service is already used to co-allocate MPI workflows it seems feasible to set up a testbed combining "semantically-enriched" and "classical" adapters. It will then be possible to compare the functional range of both solutions. It is envisaged that this evaluation will lead to another iteration of the ontology building process.

In this context it will also be necessary to review the model by means of usage scenarios that include arbitrary resource types. Although the use cases described in Section 2 reflect mostly general Grid scheduling requirements, the `query-current-schedule` capability is modelled according to the requirements of queue-based local resource managers. The integration of network resource managers into the VIOLA meta-scheduling environment will allow us to do this review.

In addition we will examine the `alter-reservation` resource management capability and negotiation-related issues linked to the negotiation protocol activities of the GRAAP Working Group at GGF [20].

# References

[1] O. Wäldrich, Ph.Wieder, and W. Ziegler. A Meta-scheduling Service for Co-allocating Arbitrary Types of Resources. In *Proc. of the Second Grid Resource Management Workshop (GRMWS'05) in conjunction with the Sixth International Conference on Parallel Processing and Applied Mathematics (PPAM 2005)*, Poznan, Poland, September 11–14, 2006. To appear.

[2] A. Keller and A. Reinefeld. Anatomy of a Resource Management System for HPC Clusters. In Y. C. Kwong, editor, *Annual Review of Scalable Computing*, volume 3 of *Series on Scalable Computing*, pages 1–31. Singapore University Press and World Scientific Publishing, 2001.

[3] PBS Professional, 2006. 12 Mar. 2006 <http://www.altair.com/software/pbspro.htm>.

[4] Platform LSF, 2006. 12 Mar. 2006 <http://www.platform.com/Products/Platform.LSF.Family/Platform.LSF/>.

[5] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Specification (WS-Agreement) Version 2005/09, September 2005. 12 Mar. 2006 <https://forge.gridforum.org/projects/graap-wg/document/WS-AgreementSpecificationDraft.doc/en/24>.

[6] S. Quirolgico, P. Assis, A. Westerinen, M. Baskey, and E. Stokes. Toward a Formal Common Information Model Ontology. In *International Workshop on Intelligent Networked and Mobile Systems, Web Information Systems Engineering (WISE 2004)*, volume 3307 of *Lecture Notes in Computer Science*, pages 11–21. Springer, 2004.

[7] J. Brooke, D. Fellows, and J. MacLaren. Interoperability of Resource Description Across Grid Domain Boundaries. In *Proc. of the European Congress on Computational Methods in Applied Science and Engineering (ECCOMAS 2004)*, Jyväskylä, Finland, July 24–28, 2004.

[8] J. Brooke, D. Fellows, K. Garwood, and C. Goble. Semantic matching of Grid Resource Descriptions. In *Proc. of the 2nd European Across Grids Conference*, Nicosia, Cyprus, January 28–30, 2004.

[9] J. Brooke, K. Garwood, and C. Goble. Interoperability of Grid Resource Descriptions: A Semantic Approach. In *Proc. of the 1st GGF Semantic Grid Workshop in conjunction with GGF 9*, Chicago, USA, October 5, 2003.

[10] Ph. Wieder and W. Ziegler. Bringing Knowledge to Middleware – Grid Scheduling Ontology. In V. Getov, D. Laforenza, and A. Reinefeld, editors, *Future Generation Grids, Proceedings of the Workshop on Future Generation Grids*, pages 47–59, Dagstuhl, Germany, November 1–5, 2004. Springer. ISBN: 0-387-27935-0.

[11] M. Roehrig, W. Ziegler, and Ph. Wieder. Grid Scheduling Dictionary of Terms and Keywords. Grid Forum Document GFD.11, Global Grid Forum, 2003.

[12] D. L. McGuinness and F. v. Harmelen. OWL Web Ontology Language Overview. W3C Recommendation, World Wide Web Consortium (W3C), February 10, 2004. 12 Mar. 2006 <http://www.w3.org/TR/owl-features/>.

[13] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[14] J. Schopf. Ten Actions When Grid Scheduling – The User as a Grid Scheduler. In J. Nabrzyski, J. Schopf, and J. Weglarz, editors, *Grid Resource Management – State of the Art and Future Trends*, pages 15–23. Kluwer Academic Publishers, 2004.

[15] J. Skovira, W. Chan, H. Zhou, and D. Lifka. The EASY — LoadLeveler API Project. In D. G. Feitelson and L. Rudolph, editors, *Proc. of 2nd Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1162 of *Lecture Notes in Computer Science*, pages 41–47. Springer, 1996.

[16] O. Corcho, P. Alper, I. Kotsiopoulos, P. Missier, S. Bechhofer, D. Kuo, and C. Goble. An Overview of S-OGSA: A Reference Semantic Grid Architecture. *Journal of Web Semantics*, n.d. To appear.

[17] G. Klyne and J. J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, World Wide Web Consortium (W3C), February 10, 2004. 12 Mar. 2006 <http://www.w3.org/TR/rdf-concepts/>.

[18] M. E. Gutirrez, A. Gmez-Prez, O. M. Garca, and B. V. Terrazas. Ontology Access in Grids with WS-DAIOnt and the RDF(S) Realization. In *Proc. of the 3rd GGF Semantic Grid Workshop in conjunction with GGF 16*, Athens, Greece, February 15, 2006. 12 Mar. 2006 <http://www.semanticgrid.org/GGF/ggf16/papers/OntoGrid-GGF16-SemGrid-Wrkshp.pdf>.

[19] OntoGrid Project, 2006. 12 Mar. 2006 <http://www.ontogrid.net/ontogrid/home.jsp>.

[20] Grid Resource Allocation Agreement Protocol (GRAAP-WG), 2006. 13 Mar. 2006 <https://forge.gridforum.org/projects/graap-wg/>.