



ADMIRE D4.2 – Development and Deployment Report for USMT V2: Capabilities of USMT V2

Project Title	ADMIRE
Document Title	ADMIRE D4.2 – Development and Deployment Report for USMT V2: Capabilities of USMT V2
Deliverable Number	D4.2
Authorship	Vivian Lee and work package partners
Document Filename	ADMIRE-D4.2-DevelopmentandDeploymentReport_1.1.doc
Document Version	1.2
Distribution Classification	Internal
Distribution List	<i>ADMIRE Project Team</i>
Approval List	<i>Alexander Wöhrer, Project Manager, Executive Board</i>

Document History

<i>Personnel</i>	<i>Date</i>	<i>Comment</i>	<i>Version</i>
V.Lee	30/01/2009	First draft	0.1
V.Lee	19/20/2009	Input from AH, AW, CA, VL	0.2
D. Snelling	20/02/2009	Last initial review	0.3
R.Baxter	23/02/2009	Project manager's review	0.4
A. Wöhrer	23/02/2009	First review	0.5
V.Lee	24/02/2009	Tidy up after the first review	0.6
V.Lee	02/03/2009	Tidy up after CA's modified registry	0.7
D. Snelling	02/03/2009	Final Proof Read	0.8
R.Baxter	04/03/2009	Final edit and signoff	1.0
V.Lee	24/08/2009	Revised by AH, VL	1.1
R,Baxter	27/08/2009	Signoff for resubmission	1.2

Contents

Contents.....	1
Executive Summary	3
1 WP 4: Summary: Project Month 12.....	5
1.1 Products delivered to other work packages	5
2 Progress against Planned Activities.....	6
2.1 Activity 4.1: OGSA-DAI Integration (FLE, UEDIN).....	6
2.1.1 Planned activity.....	6
2.1.2 Actual activity	6
2.1.3 Deviations from plan	7
2.2 Activity 4.3: Semantic Registry Integration (FLE, UPM)	7
2.2.1 Planned activity.....	7
2.2.2 Actual activity	8
2.2.3 Deviations from plan	9
2.3 Activity 4.4: Semantic Provisioning Services Integration (FLE, UPM)	10
2.3.1 Planned activity.....	10
2.3.2 Actual activity	11
2.3.3 Deviations from plan	11
2.4 Activity 4.7: General Enhancement to USMT (FLE, UVIE, UEDIN, UPM).....	12
2.4.1 Planned activity.....	12
2.4.2 Actual activity	12
2.4.3 Deviations from plan	15
2.5 Other Activities.....	15
3 Risks and Issues.....	16
3.1 Project issues.....	16
3.2 Significant problem reports.....	16
3.3 New risks	16
4 Plans for next Period.....	16
4.1 Activity 4.1: OGSA-DAI Integration (FLE, UEDIN).....	16
4.2 Activity 4.2: Enhanced Monitoring (FLE, UVIE)	16
4.3 Activity 4.3: Semantic Registry Integration (FLE, UPM)	17
4.4 Activity 4.4: Semantic Provisioning Services Integration (FLE, UPM)	17
4.5 Activity 4.5: WEEP Integration (FLE, UVIE).....	18
4.6 Activity 4.6: OGSA-DQP Integration (FLE, UEDIN).....	18
4.7 Activity 4.7: General Enhancement to USMT	18

- 4.8 Activity 4.8: Standards Alignment (FLE, UVIE, UEDIN, UPM) 19**
- 5 APPENDIX Error! Bookmark not defined.**
- 5.1 OGSA-DAI Services 20**
- 5.2 USMT Design and Implementation 20**
 - 5.2.1 Resource Property 20**
 - 5.2.2 Resource Lifetime 21**
 - 5.2.3 Notification 22**
 - 5.2.4 Subscriptions 22**

Executive Summary

This document is ADMIRE Deliverable D4.2 and describes progress on Work Package 4 during months 7 to 12 of the ADMIRE project.

At project month 12 ADMIRE WP4 is slightly behind the original plans, but has successfully deployed the ADMIRE-enhanced version v1.5¹ and v1.6 of USMT. The integration of OGSA-DAI with USMT is slightly delayed, partly for technical reasons and partly because of staff departures from both the USMT and OGSA-DAI teams. It was also expected that monitoring and registry activity would have reached an initial implementation phase; however, more focus has been placed on architecture, design, and reviewing existing semantic technologies for registries and provisioning. Implementation will be straightforward, as we have decided to use existing technology from the University of Vienna for monitoring as a standalone capability rather than integrating it within OGSA-DAI. Likewise, the use of an existing open source, OGSA-DAI compatible registry (OGSA-DAI-RDF²) implementation will simplify this development.

Related documents

- [1] The ADMIRE Consortium. ADMIRE: Description of Work, Feb 2008. <http://www.admire-project.eu/trac/browser/projman/plans/ADMIRE-Annex1-DoW.doc>
- [2] ADMIRE Risks and Issues Log, July 2008, <http://www.admire-project.eu/trac/browser/projman/plans/ADMIRE-risksIssues.doc>
- [3] The OGSA-DAI Project <http://www.ogsadai.org.uk/>
- [4] OGSA-DAI Component Registry: <http://www.ogsadai.org.uk/contribs/registry.php>
- [5] JAX-WS reference Implementation: <https://jax-ws.dev.java.net/>
- [6] Apache Ant: <http://ant.apache.org/>
- [7] WS-Naming: <http://schemas.ogf.org/naming/2006/08/naming/>
- [8] O. Corcho, P. Alper, I. Kotsiopoulos, P. Missier, S. Bechhofer, C. Goble, An overview of S-OGSA: a Reference Semantic Grid Architecture, J. of Web Semantics 4(2), 2006.
- [9] Amy Krause, Carlos Buil, Rafał Gąsiorowski, Branislav Simo, Michal Laclavik, Ivan Janciak, and Rob Baxter. ADMIRE – Tools Development Report and Requirements Analysis. Deliverable report D5.2, the ADMIRE Project, Feb 2009.
- [10] Sven van den Berghe et al. ADMIRE – Development and deployment report for USMT V0 and V1: Capabilities of USMT V1. Deliverable report D4.1, the ADMIRE Project, Aug 2008.
- [11] R. García-Castro and M.C. Suárez-Figueroa (eds), “D 1.2.4 Architecture of the Semantic Web Framework”, Knowledge Web Deliverable, Feb 2007.
- [12] W. Fang, S. C. Wong, V. Tan, S. Miles, L. Moreau, (2005) Grimoires: Grid Registry with Metadata Oriented Interface: Robustness, Efficiency, Security--- Work-in-Progress<<http://eprints.ecs.soton.ac.uk/10862/>>. In Proceedings of Work in Progress Session held in ClusterComputing and Grid(CCGrid)/, Cardiff, UK.

¹ The USMT release numbers in this document, e.g. V0, v1.5, v1.6, v1.7 follow the software engineering convention. The matching up between these numbers and the number in the DoW follows the pattern below.

USMT v0.x matches to USMT V1 in the DoW

USMT v1.5 and v1.6.x matches to USMT V2 in the DoW

USMT v1.7.x matches to USMT V3 in the DoW

² <http://dbgrid.org/OGSA-DAI-RDF>

- [13] W. Waterfeld, M. Weiten, P. Haase, H. Cunningham, M. Dzubor, R. Palma, (2007)
Specification of NeOn Reference Architecture and NeOn APIs. Deliverable D 6.2.1, NeOn
Project, March 2007

1 WP 4: Summary: Project Month 12

The objective of Work Package 4 is to develop and enhance the core service infrastructure that will form the foundations of the DMI platform which will be deployed to this infrastructure and delivered to the consortium and beyond by WP3. The architecture effort of WP2 determines the components to be developed and included as part of the DMI platform and the modeling and language research undertaken in WP1 will form the basis of the DMI model for the platform. Where requirements are identified for enhancing the capabilities of the infrastructure for wider applicability across a number of tools from WP5 and services from this work package, respectively, the infrastructure will be enhanced accordingly.

The core service infrastructure is known as USMT (Unified Systems Management Technologies). USMT will provide a set of common service management capabilities that are required for the monitoring and management of services hosted by the DMI platform. USMT, provided by Fujitsu Labs of Europe at the start of the project (as V0 in task WP4.1), will be enhanced and expanded as required by the work of the project to support the needs of Data Mining and Integration.

Data Mining and Integration processes are implemented as a set of individual processing steps using Grid services, which are combined to form the end-to-end process. These processes can be complex workflows involving several sub-processes such as data selection, data filtering, data transformation, data integration, data modeling (applying a data mining algorithm), and the post-processing of mining results (e.g. visualization). The Data Mining and Integration processes are often large in terms of the number of sub-processes in the workflow, in terms of the total execution time of the process and in terms of the data volume involved.

In this work package, USMT will be modified to facilitate the management and monitoring of complex DMI oriented workflows, by integrating a number of technologies.

Current and future activities include the OGSA-DAI platform, an enhanced monitoring framework for DMI oriented processes, a DMI workflow enactment engine, the OGSA-DQP platform, a semantic registry and semantic provisioning services for providing advanced service and resource publishing and discovery mechanisms.

USMT will give rise to a layered Data Mining and Integration environment that will allow DMI services to make use of encapsulated DMI service with USMT as the foundation layer of each.

In months 7 to 12 of ADMIRE WP4's main focus has been to deploy enhanced USMT to the project and to port OGSA-DAI to the enhanced USMT. Two versions of USMT have been distributed to the project, with enhancements that address the requirements that arose from the first stage of the integration activity. Due to staff changes within both teams, the integration of OGSA-DAI to USMT is slightly delayed; however, it is hoped that this will be overcome in a relatively short period of time by employing a new integration strategy. It was expected that monitoring and registry activity would have reached an initial implementation phase; however, more focus has been placed on architecture, design, and reviewing existing semantic technologies for registries and provisioning. Implementation will be straightforward, as we have decided to use existing technology from the University of Vienna for monitoring as a standalone capability rather than integrating it within OGSA-DAI. Likewise, the use of an existing open source, OGSA-DAI compatible registry (OGSA-DAI-RDF) implementation will simplify this development.

1.1 Products delivered to other work packages

FLE released two versions of USMT to the ADMIRE project, primarily to address the requirements from UEDIN during the integration of OGSA-DAI with USMT:

- USMT for Admire v1.5 was released to ADMIRE on 8 July 2008.
- USMT for Admire v1.6 was released to ADMIRE on 31 January 2008

- OGSA-DAI has not yet been released as an integrated platform with USMT.

2 Progress against Planned Activities

2.1 Activity 4.1: OGSA-DAI Integration (FLE, UEDIN)

2.1.1 Planned activity

The OGSA-DAI platform technology provides extensive data access, integration and management functions for a large variety of database operations and data handling scenarios. An objective of this work package is to complete the deployment the OGSA-DAI platform to the USMT, leveraging all of the generic capabilities offered by the USMT, in particular those to monitor workflows and report faults, and extending the infrastructure with any data oriented management functionalities. The University of Edinburgh will work with FLE in this activity.

Appendix B gives details of the six OGSA-DAI services. The aim of this activity is to develop a USMT presentation layer for each of these services that is a layer on top of the core OGSA-DAI APIs released by the OGSA-DAI project.

2.1.2 Actual activity

The last report on this work [10] noted that four of the six OGSA-DAI services had been ported to USMT. The two remaining services (DataSink Service and DataSource Service) proved harder to port than expected. This was due to differences in the threading model expected by JAX-WS, and hence USMT, and those of OGSA-DAI. OGSA-DAI executes workflows on several threads sometimes even after the user's Web service request that sent the workflow to OGSA-DAI has returned. These threads had difficulty in creating USMT resources because JAX-WS only supports resource creation for request threads. A workaround was adopted to bypass this problem and allow development to continue in advance of a solution included in USMT release v1.6.

Much progress has been made in unifying OGSA-DAI's resource ID based approach with USMT's EPR based approach. Support was added to USMT for abstract names (based on the WS-Naming specification [7]) and this has been used within OGSA-DAI's USMT layer to provide better integration between the two approaches. To fully complete the EPR-based view for OGSA-DAI new versions of OGSA-DAI activities that work with remote data source and data sink services must be written. Work on this area has led to a new approach to designing these activities that will be adopted in the next public release OGSA-DAI in addition to internal ADMIRE releases.

USMT release v1.5 did not support notifiable runtime resource properties. The only runtime support for resource properties was dynamic resource properties that do not support notification. The port of OGSA-DAI to USMT v1.5 used dynamic resource properties and hence did not support notification. To fix this problem runtime static resource properties that do support notification were added to USMT release v1.6.

The port of OGSA-DAI to USMT v1.5 has bugs relating to the destruction of dynamically created resources that still appear in the USMT nucleus's list of resources even then they have been destroyed. USMT v1.6 will provide OGSA-DAI with a more flexible approach to creating resources that is expected to fix this problem.

The task of porting the USMT v1.5 version of OGSA-DAI to USMT v1.6 to overcome the problems noted above is currently ongoing. This has proved harder than expected. Many things have changed and these are currently breaking previously working code. Development is slowed by the lack of a comprehensive development guide for the new USMT architecture. While the documentation is a huge improvement on that available with v1.5 it still lacks detailed working examples of how to use much of the provided functionality. This is being addressed as part of the v1.6 development.

2.1.3 Deviations from plan

The task of porting OGSA-DAI to USMT has taken longer than expected. We believe we are close to having all six OGSA-DAI services running on USMT v1.6. UEDIN and FLE are working closely on this task. The ideal plan would be to develop simple basic services in USMT that exhibit all the functionality that OGSA-DAI requires of USMT. While sounding simple, OGSA-DAI is extremely complex and isolating functions into simple examples may obscure subtle interactions. Nonetheless, this approach will provide those simple examples needed by other parts of the project and form part of the USMT documentation.

2.2 Activity 4.3: Semantic Registry Integration (FLE, UPM)

2.2.1 Planned activity

The purpose of a Registry is to define certain aspects of the computational context of some stage of a DMI process. For example, in the context of a DMI Workbench that houses a set of tools, including at least one DMI Process Designer (PD) tool, it will keep track of the set of Processing Elements (PE) that may be used via the PD tool. As the tool may be used to generate a new PE it will also track these as they are iteratively developed, even before they are ready for execution.

Additionally, a Registry supporting a DMI Gateway will keep track of the entire set of PEs that can be used via that Gateway; these must all be in some sense complete and valid in that context so that they are capable of being executed. The process design inevitably occurs at a different (earlier) time from process enactment. Consequently the state of the computational context as represented by a Registry during process design may be different from the state during enactment.

There is a many-to-many mapping between DMI Workbenches being used by design teams and DMI Gateways being used by design teams or application-domain user communities to enact the DMI processes. Therefore the computational context will differ depending on location. Consequently, separate Registries are needed for each location. For the moment, we assume the following locations:

1. one for each community of DMI Workbenches in use for related DMI purposes, i.e. a Registry may support many DMI Workbenches but each Workbench uses only one Registry, and
2. one for each DMI Gateway.

Of course, as far as possible, the functions and representation used by ADMIRE Registries should be the same in every location, so that the same semantics, code and service interface can be used throughout. We say “as far as possible” because Registries themselves will evolve and this evolution will propagate incrementally through the locations.

The initial version of USMT includes a basic Registry that supports service discovery and management of property-based information about the target services. Enhanced support is also available to use this registry mechanism to provide collective management of collections of services, e.g. a workflow of processes or a network of clusters. Within this activity we will assess the ability to either extend this functionality to provide semantic content, i.e. by providing a SPARQL based query capability, or alternatively by integrating a third party semantic registry, e.g. Grimoires [12], the semantic registries of the NeOn Toolkit [13] or the Metadata Registry of the Semantic Web Framework [11]

In this period an analysis of several semantic registries has been done. jUDDI+, Grimoires, Fusion semantic registry, S-MDS and OGSA-DAI-RDF(S) were analysed. We finally selected OGSA-DAI-RDF(S) due to the integration of OGSA-DAI with USMT. In the next two months the implementation of a first version of the Registry will be completed.

2.2.2 Actual activity

The Registry will be composed by an OGSA-DAI activity, which will be queryable by means of SPARQL. The queries will be high level enough to retrieve all the general data mining methods and their different implementations. The Registry will be built taking into account this restriction. For fulfilling it, OGSA-DAI-RDF will be used. OGSA-DAI-RDF is a set of OGSA-DAI activities designed for RDF processing, including a query processing activity using W3C SPARQL. OGSA-DAI-RDF provides an internal product-independent access layer, which wraps product dependency. In OGSA-DAI-RDF, an RDF data resource in the context of OGSA-DAI is equivalent to a RDF Dataset of W3C SPARQL. An RDF Dataset is a set of RDF Graphs. An RDF Graph is a set of RDF triples, which include Subject, Predicate and Object. Each graph is uniquely identified with a URI. Thus, an RDF data resource holds a set of Graphs with a URI.

When users are designing a new data mining process they will query for functions, named types or libraries satisfying certain conditions it supports (or can support indirectly).

The ADMIRE Registry will provide access to the Processing Elements that are designed by the ADMIRE end users. It will be a semantic registry in nature based on the WS-DAI-RDF specification. Currently an implementation of this specification is OGSA-DAI-RDF. This implementation provides more operations than the ones existing in the specification like update methods. The queries to the resource are done by using SPARQL. Users of the ADMIRE Registry will be the Process Designer or the SKSA. The working process is described as follows:

- a user requests for an already-designed Processing Element (PE);
- the PD, knowing what element is requesting by the user, activates the SKSA;
- the SKSA has access to the CRISP-DMI and Data Mining ontologies (developed in WP1) and is able to build a SPARQL query to the Registry. This is possible because all PEs that are created by the PD are annotated in the CRISP-DMI ontology by the SKSA;
- if there is no PE matching the query in the local Registry, the local Registry will query the ADMIRE Gateway for possible matches in other Registries. If there is any match, the location of these matches will be propagated to the local Registry;
- the Registry will return the locations of the desired PEs to the PD;
- the PD will access to the PE stored in the database containing the data mining models.

It is important to note that there will be two Registries with the same functionality but in different locations. The first Registry will be located locally for a PD to access directly. The other one will be available for external queries from the ADMIRE Gateway. The OGSA-DAI-RDF project is developed as an OGSA-DAI 3.0 resource for the Globus toolkit and Apache Tomcat. One of the tasks to be done in this activity is to port the code from these systems to OGSA-DAI 3.1 and USMT.

Figure 1 shows the interactions described (cf. ADMIRE Deliverable D5.2 [9]).

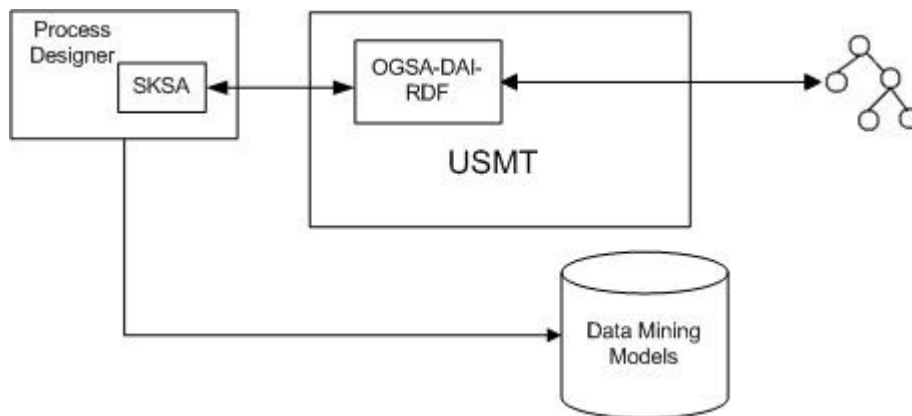


Figure 1: Registry interactions

The functionality provided by the resource OGSA-DAI-RDF corresponds to the WS-DAI-RDF specification but for update activities. OGSA-DAI-RDF provides six activities:

- **DataSetManagementActivity:** The interface for RDF Dataset (= Set of RDF Graphs). This function allows one to create, remove and list RDF Graphs within a DataSet (= RDF data resource).
- **GraphManagementActivity:** The interface for RDF statements. This activity allows one to insert and delete RDF statements (triples).
- **sparqlQueryStatementActivity:** The interface for the SPARQL query interface. Named graphs and default graphs are supported.
- **OntologyReasonerActivity:** This interface invokes a reasoner and creates an inferred graph. The output can be an RDF data stream or graph URI, which identifies the created graph.
- **rdfsSchema Activity.**
- **rdfsInstance Activity.**

Currently the WS-DAI-RDF specification only includes the **DataSetManagementActivity** and the **sparqlQueryStatementActivity**. The activity **DataSetManagementActivity** provides operations for creating, deleting and listing existing RDF graphs. These operations allow users to insert new RDF(S) ontologies into the storage system. The activity **sparqlQueryStatementActivity** contains operations for querying the RDF(S) ontologies and allows users to query for elements in the Registry (which is an ontology). The other activities are not defined in the specification. These undefined activities contain the RDF graph operations on RDF data resources, which are insert statements and delete statements. These activities are part of the core of a Registry (along with the query activities).

2.2.3 Deviations from plan

The main deviations concern the operations *insert* and *delete*. These operations are included in the release of the OGSA-DAI-RDF resource and activities but not in the last version of the WS-DAI-RDF specification³. Since these two operations are important and already implemented in a concrete implementation of the specification they most probably will be added to the specification. There are two problems which might arise from this: these operations might not be included in the WS-DAI-RDF specification; or these operations are included but with a different signature from that in the OGSA-DAI-RDF implementation. The actions for these deviations would be:

- If the specifications for insert/delete statements were included with different signatures the implementation of access to the Registry would be adapted. The new signature most probably

³ <http://forge.gridforum.org/sf/go/doc14074?nav=1>

will not differ significantly and access would be similar. By adding a new layer between the end user application and the Registry the only implementation to be changed would be the one from the Registry side. A new version of the Registry would be released using the correct WS-DAI-RDF specification.

- If the specification does not include the insert/delete statements operations the Registry will remain unchanged. These operations are necessary in a Registry and they will be considered as an extension of the final specification.

Another deviation would arise if the specification changes completely. The end users would need to modify their implementation to access the Registry. A layer between the Registry and the end user applications would then be needed. If the new specification differs from the current implementation a new implementation would be created and a new version of the Registry released.

For month 12 the goal was to complete an analysis of several existing registry systems, which has been accomplished. The next goal is to finish the implementation for month 14 which will be accomplished. No planned deviations are contemplated here.

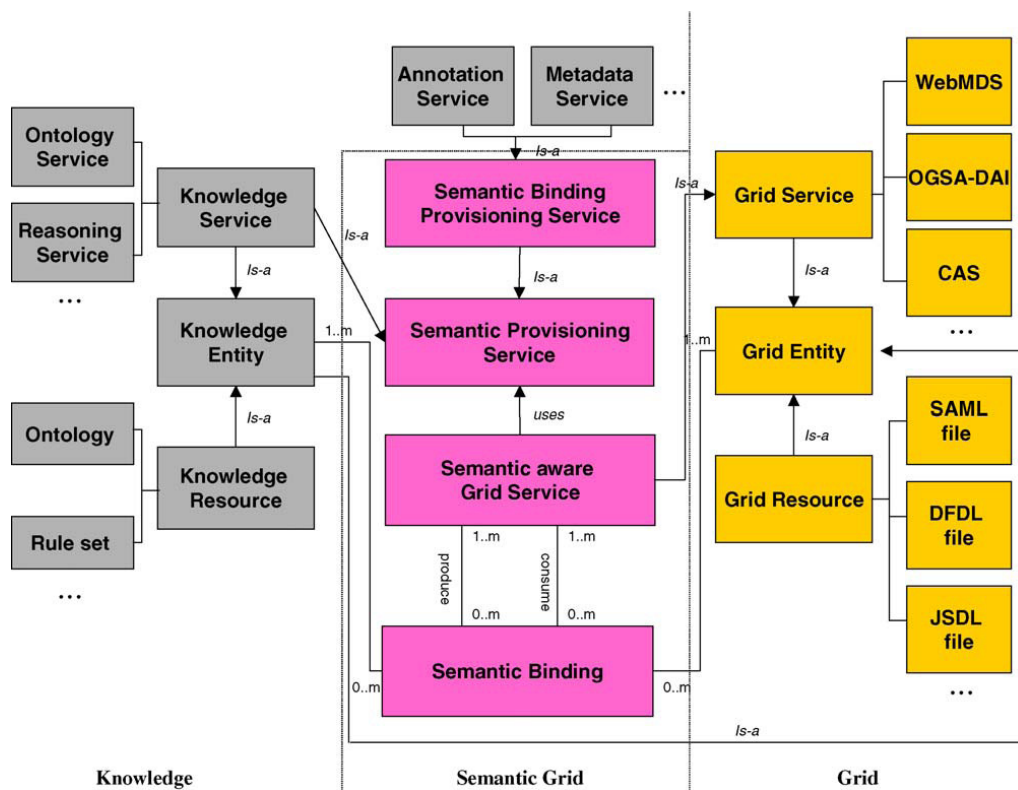
2.3 Activity 4.4: Semantic Provisioning Services Integration (FLE, UPM)

2.3.1 Planned activity

Semantic Provisioning Services are the services that give support to the provision of semantics, by allowing the creation, storage, update, removal and access of different forms of knowledge and metadata (cf. S-OGSA [8]). Figure 2 shows the S-OGSA architecture. The figure is coloured to differentiate the type of services. Gray elements in the figure are knowledge entities using for representing concepts, reasoning, etc. Yellow elements are grid elements such as OGSA-DAI, JSDL files, etc. Finally pink elements represent semantic grid elements in the S-OGSA architecture.

Semantic provisioning services are classified into two different groups: knowledge provisioning services (includes services for storage and reasoning) and semantic binding provisioning services (includes services for managing metadata, mainly annotation services). Activity 4.4 mainly focus on integrating semantic provisioning services into the ADMIRE platform. This activity therefore includes the integration of services that create annotations of the Processing Elements (PEs) (the Service Descriptor) with the Workbench used to create the PE (the Process Designer), and with the services that will use the annotations (the SKSA and the ADMIRE Registry). In ADMIRE the planned semantic provisioning services are:

- **Annotation services:** the annotation service is the Service and Data Description Assistant (SDDA) [9], which is a tool for annotating the interactions of the users with the Process Designer (PD). These interactions refer to data mining elements created by a user in the PD Workbench, e.g. when a user creates a new CRISP-DMI element in the PD, a corresponding new instance that represents this CRISP-DMI element will be created in the ontology. In the meantime, the SDDA will also create all the relationships with previously annotated elements. This instance allows the SKSA to manage the design process for the user.
- **Metadata services:** the metadata services in the ADMIRE context is the ontology repository in which the ontologies will be stored. There is no repository selected at this stage.

Figure 2 : S-OGSA Architecture⁴

2.3.2 Actual activity

This activity has ensured that the semantic provisioning services interact perfectly with the services that use them. This means that:

- the SKSA is able to access the annotations and the elements of the CRISP-DMI ontology through a semantic repository (in which this ontology will be stored);
- the ADMIRE Registry is able to access the elements that are created in the ontology for describing the PE;
- the Service Descriptor interacts properly with the Process Designer for correctly annotating all the created PEs;
- a reasoning service is available if necessary for the ADMIRE services.

All the previous interactions of the semantic provisioning services are described in deliverables D4.2 (this document) and D5.2, Report on Tools [9]. This activity relies heavily on other ADMIRE components and therefore cannot be completed until these other components are finished, but the activity can monitor all interactions. Since the developer organization of the semantic provisioning services (UPM) is the same than the organization in charge of the integration of them with other components active tracking of this activity will be straightforward.

2.3.3 Deviations from plan

The deviations from plan were caused either the status of development of other components, or the delay of the semantic provisioning services. To minimize possible future deviations from the integration with other components, unitary tests will be done to the semantic provisioning services

⁴ From [8].

accessed by other ADMIRE components. If the deviation comes from the development of the semantic provisioning services, more effort will be distributed to these development activities. Another deviation would be the need for a reasoning service. These services provide inference functions to the users. One possibility in ADMIRE is that the SKSA would require more reasoning capabilities than the ones offered by an ontology repository. Effort would be put to the development of a reasoning service.

2.4 Activity 4.7: General Enhancement to USMT (FLE, UVIE, UEDIN, UPM)

2.4.1 Planned activity

As a service-consolidating platform, USMT has already provided a set of core, common management capabilities, including:

- service monitoring, including monitoring of both generic and application-specific properties;
- lifetime management, including creation and destruction of services;
- lifecycle management, including provisioning, configuration, and activation of resources;
- subscriptions, requests and notification, including lifecycle and property change events;
- service naming and addressing;
- basic registry provision;
- security; and,
- fault propagation and handling.

Appendix **Error! Reference source not found.** provides a brief overview of the USMT service architecture.

Over the course of the project, these functions may need further development and enhancements as dictated by the requirements of the DMI platform. Any such work will be the responsibility of this work package and will be carried out by FLE.

To this end, over the lifetime of the project, as new tools and services are deployed to USMT, infrastructure requirements and enhanced capabilities will be identified. Where a capability is identified as a new infrastructure requirement, and can be generalized for provision to a number of tools and services, this capability will be integrated into USMT for wider applicability.

On the other hand, if an infrastructure functionality is identified that is specific to a given tool or service, that functionality will be provided specifically to support the tool or service for which it was identified. The overall design goal is to create as much reuse of functionality as possible, where that functionality is identified as being of a general and broadly applicable nature, while also supporting the specific functional requirements of every tool and service deployed to USMT.

2.4.2 Actual activity

In the first period of the project, USMT provided a set of core, common management capabilities, including: generic and application-specific properties access, support and management; service lifetime management; lifecycle management; subscriptions and event notifications; basic registry provision; security; and fault handling and propagation. Section 5.2 describes details of the design and implementation of these core capabilities. At the second stage, the development of the USMT platform is driven mainly by the requirements arose during the OGSA-DAI/USMT integration activity. These requirements are identified as follows:

- support for service instance creation from non-request threads;
- the need for WS-Naming support;

- runtime resource property creation, and notification;
- clients need to disable host name verification explicitly when using HTTPs mode;
- lack of comprehensive documentation for USMT architecture and development;
- issue relating to the destruction of dynamically created resources that still appear in the USMT nucleus's list of resources even when they have been destroyed.

Solutions to the above issues, which are significant enhancements to the USMT platform, are described below (along with their problem-report ticket number from the ADMIRE Trac PR system).

Service instance creation from non-request threads (Trac #112)

In USMT, every service is a stateful Web service. The creation and management of these stateful service instances are through `StatefulInstanceManager` provided by JAX-WS [5]. However, one crucial issue of this manager lies in the method that it uses to create `EndpointReferences` for service instances. During the creation of a service instance, that method needs to populate the `Addressing` field of `EndpointReference`, and the only way to get the `Addressing` value is to look it up in the execution environment in which the current request message is running – in other words from the same thread that is handling the Web Service request. On the other hand, OGSA-DAI has many typical use cases in which an instance is not created from a request thread, but from one of many threads that spawn from a workflow at runtime.

The combination of circumstances resulting from this scenario makes creation through the JAX-WS provided `StatefulInstanceManger` impossible. An initial thought to address this issue was to develop an `USMTServiceInstanceManager`, and an `USMTServiceInstanceResolver`, which directly subclass JAX-WS's `StatefulInstanceManger`, with extra functionality to get the `Address` from outside the request environment. However, the JAX-WS RI implementation defines the `StatefulInstanceManger` as a final class; this effectively prevents this approach and leads USMT v1.6 to adopt an alternative approach, which is to subclass the superclass of `StatefulInstanceManager`, and completely bypass the `StatefulInstanceManger`, re-implementing all of its functionality. This allows the runtime process to get the `Addressing` value from the lightweight `Endpoint` container's owner environment, thus relaxing the limitation that instance creation has to originate from a request thread. Despite the fact that this solution fits well into the USMT hosting environment, recommendations to the SUN's JAX-WS RI team are needed to suggest that Sun either remove the final keyword on the `StatefulInstanceManager` class, to allow extensibility, or to modify the `create` method to get the `Addressing` value from several different alternative sources when the default source fails. This also benefits the JAX-WS platform with better quality and flexibility.

WS-Naming support (Trac #114)

In USMT, service instances (i.e. WS-Resources) are primarily identified by WS-Addressing `EndpointReferences` that must be used when communicating with the respective WS-Resource. However these service instances are identified by a resource ID in OGSA-DAI. To make the communication between these two systems easier, WS-Naming is indeed needed. Although promised as a core function at the start of the project, it is not yet fully supported in USMT v1.6. The implementation follows the final specification [7] of WS-Naming, and uses JAX-WS to tool-up schemas and WSDLs. The OGSA-DAI resource ID is injected to the metadata section of the `EndpointReference`, represented by the `QName` of `EndpointIdentifier`, which is conceptually an abstract name. The injection is done after the creation of the service instance and consequently, the `KnownServiceInstance` resource property of the running Nucleus can be updated accordingly. WS-Naming serves as a bridge between these two different architectures, and provides the OGSA-DAI client toolkit with a mechanism to communicate to the OGSA-DAI backend via the OGSA-DAI USMT presentation layer.

Runtime resource property creation, and notification (Trac #115)

USMT uses JAVA reflections to scan the instances of predefined USMTServiceProperty variables in the service implementation class at compile time. In another words, all the resource properties in the USMT framework have to be private variables of a service implementation class, and have to be statically declared at compile time. One side effect of this approach is that it only allows compile time resource properties. Any resource properties created at run time will not be able to be inserted into this implementation class. The solution to this issue in USMT v1.6 is to add a hash table to the implementation class to store such runtime resource properties. This is an acceptable workaround to the problem, but is not a perfect solution because it breaks the integrity of the overall architecture. A better approach will be considered in a future release.

Explicit host name verification disable (Trac #111)

Since JDK1.4, hostname verification (a part of the security context creation process) is enabled by default when a SSL protocol is used. The main purpose is to prevent URL spoofing, e.g., when a client sends a request for some URL, some man-in-the-middle in between intercepts and sends his own certificate, and the original sender remains under the impression that he received the certificate of the requested server. This constitutes a security threat. This mechanism is inflexible when the client certificate is not created based on a hostname. To turn off the verification, the client always has to execute explicit code. However, since this feature is enabled by default since JDK1.4, the USMT client package provides a simple utility operation to disable verification.

Documentation (Trac #63)

A comprehensive USMT developer's guide is included in USMT v1.6 release, which describes in detail how to build application services on top of the USMT platform. The first section explains terminologies used throughout the USMT infrastructure. For eager developers, the cookbook section gives a straightforward start on how to build an application service. The architecture section complements the need to develop good and in-depth services that maximize the capabilities of USMT.

Generic USMTFactory (Trac #113)

USMT v1.6 provides a unified way of creating service instances – a USMTFactory. Any type of service, as long as they are sub-classed to USMTService, can be created via this generic factory at anytime – bootstrapping, compile-time or runtime. The approach is to take the implementation class name and use its default constructor to create an instance out of that class. The downside, which will be improved, is that this requires all service classes to have an empty constructor. In the mean time, most of the OGSA-DAI service instances only have a constructor that takes an OGSA-DAI resource, which matches the corresponding OGSA-DAI-USMT instance at the presentation layer. Therefore, some other mechanism has to be employed to set this value after service creation; this consequently delays OGSA-DAI's initialisation. An enhanced method needs to be in place to address this type of infrastructure in the next release to improve the USMTFactory class as a whole.

Dynamically Created Resources not removed after destruction (Trac #114)

This is a bug that carried over from USMT v1.5 to v1.6 and is fixed in v1.6.1. The symptom is that given a dynamically created service instance, which the client destroys operationally, the instance itself is destroyed but the resource property of KnownServiceInstance on the running USMTNucleus still lists the destroyed instance. The problem was discovered when OGSA-DAI's client tries to list the KnownServiceInstance again for some other purpose after the destruction. The bug is fixed by adding a topic notification listener to the DestructionTopic topic of a service instance, so that when the instance is destroyed, a notification message is sent out to the hosting Nucleus to remove the instance's QName from the list. Although this fixes the specific problem, a more comprehensive architecture will be researched to handle service instances destruction in a distributed environment as a whole in a more general sense.

Sample code for OGSA-DAI Usecases (Trac #63)

The closed-source platform nature of USMT makes the OGSA-DAI USMT integration work harder to carry out, even with relatively detailed documentation. One proposed solution is for the OGSA-DAI team to write up a use case document that simulates the OGSA-DAI working environment, and let USMT team provide sample code as open source. This activity is in its trail period; if it proves to be an efficient way of porting OGSA-DAI to USMT, then USMT will continuously provide such kind of service.

2.4.3 Deviations from plan

Due to the departure of the main developer of USMT, and because of the unforeseen technical difficulties noted above, the final USMT v1.6.x release, and hence the integration between OGSA-DAI and USMT has been delayed by approximately one month. However, with the immediate reassignment of a new staff, the maintenance and enhancement of the code base is expected to speed up in a relatively short period, to catch up with the delay.

2.5 Other Activities

No other activities were active this period. For reference these are:

- Activity 4.2: Enhanced Monitoring (FLE, UVIE).
- Activity 4.5: WEEP Integration (FLE, UVIE).
- Activity 4.6: OGSA-DQP Integration (FLE, UEDIN).
- Activity 4.8: Standards Alignment (FLE, UVIE, UEDIN, UPM).

3 Risks and Issues

3.1 Project issues

As noted above, both technical difficulties and staffing changes raised issues this period. These were formerly noted as

- Issue I2 – Incompatibilities in USMT/OGSA-DAI naming and EPR handling (OPEN), and
- Issue I3 – Key USMT developer leaving FLE (CLOSED)

The combination of these has raised the issue of the delay to USMT v2 and thus to the first ADMIRE Platform software release, D3.2, recorded as

- Issue I4 – Deliverable D3.2 delayed (OPEN).

3.2 Significant problem reports

Issue I2 is recorded in the project's Trac PR system as ticket #97. Issue I3 is #140.

The technical problems noted above include their Trac ticket numbers.

3.3 New risks

Despite the issues and problems reported, there are no new risks identified in WP4.

4 Plans for next Period

The DoW has the following activities for M13-18:

- M13-18: Improve performance of data oriented processes as supported by USMT capabilities, including advanced dataflow management.

The updated deliverable description is:

Development Progress Report: requirements for USMT V3

It is expected that USMT deployment on the ADMIRE test platform will be fairly stable by the end of this period. The work in this period is expected to focus on external components and USMT improvements that increase performance in the ADMIRE platform. This report will describe the design of any such extensions to USMT and the performance improvements gained as a result.

In addition, the report will describe the work identified to integrate a workflow enactment engine into the ADMIRE platform, by developing the necessary services to support workflow technology in USMT. It is expected that a number of USMT service interfaces are required for the technology to be integrated with the ADMIRE platform.

4.1 Activity 4.1: OGSA-DAI Integration (FLE, UEDIN)

The bulk of OGA-DAI integration is completed, but work will continue to ensure a stable platform.

4.2 Activity 4.2: Enhanced Monitoring (FLE, UVIE)

From the Description of Work: *USMT provides an extensive, standards based framework for monitoring a variety of services. However, there are a number of properties specific to the monitoring of data integration processes that will need to be integrated into the infrastructure. The University of*

Vienna has done extensive work in this area and will work with FLE to integrate extended monitoring functionality into the infrastructure.

USMT provides an infrastructure for passing information about changes in the properties of service instances. In Deliverable 4.1 [10], event aggregation was identified as an extension to the current notification support that may be required. This is contributing to an important requirement in light of ADMIRE, namely service monitoring, but additional monitoring extensions will have to be considered in order to provide an infrastructure for advanced data mining and integration as discussed below:

- *data source monitoring*: The behavior and overall needs of a process are heavily dependent on the characteristics of its input data source. This is especially true for a complex and diverse source as a RDBMS, one of the primary data providers in ADMIRE;
- *resource monitoring*: This monitoring requirement, introduced and discussed in Deliverable 4.1, amounts to gathering information about the service execution environment;
- *progress monitoring*: For distributed and often long running tasks it is desirable to get more information about its current status than just “processing”, “error” or “finished”. A question to be answered is on which level to introduce it – e.g. workflow level inside OGSA-DAI or service level – as this also influences the granularity, size and scope of possible intermediate progress reports.

Regarding monitoring, in this phase the activity focuses on the design and implementation of database monitoring in a service-oriented manner. A key requirement is the provisioning of a unified interface to the RDBMS metadata.

4.3 Activity 4.3: Semantic Registry Integration (FLE, UPM)

In Section 2.2 there is a description of the semantic Registry and the main components with which the Registry will interact. This integration activity will combine these components and coordination with the partners responsible for them will be required. There will be two uses of the Registry: as local Registry for the user’s Workbench; and as a Registry for the ADMIRE Gateways. Therefore, integration of the semantic Registry consists of two sub activities:

- ADMIRE Workbench and DMI Process Designer tools integration;
- ADMIRE Gateway integration.

Integration will be focused mainly on agreeing the interfaces offered to other components using the Registry and afterwards on improving integration with the ADMIRE Gateway.

4.4 Activity 4.4: Semantic Provisioning Services Integration (FLE, UPM)

The task described in Section 2.3 provides a description of the semantic provisioning services that will be available in the Process Designer. Thus the work of integrating semantic provisioning services with the other Process Designer components (this work of this task) will be spread across the respective component development tasks. What are not included in these development tasks are the interactions with the semantic Registry, and with the SKSA tool that will need to access to reasoning and ontology storage services. These interactions will be developed in the context of the Service and Data Description task of WP5. The repository selection for the metadata service will be done according to the growth of the ontology and the elements represented in it. The possible ontology repositories candidates are:

- **OWL API**: The OWL API is a Java interface and implementation for the W3C Web Ontology Language OWL. The latest version of the API is focused towards OWL 2 including OWL-Lite, OWL DL and some elements of OWL-Full. The OWL API is open source and is available under the LGPL Licence;

- **OWLIM Semantic Repository:** OWLIM is a semantic repository developed in Java. It is packaged as a Storage and Inference Layer (SAIL) for the Sesame RDF database. OWLIM is based on TRREE – a native RDF rule-entailment engine. The supported semantics can be configured through rule-set definition and selection. The most expressive pre-defined rule-set combines unconstrained RDFS with most of OWL Lite (as indicated on the OWL fragments map);
- **Jena:** Jena is an open source Semantic Web framework for Java. It provides an API to extract data from and write to RDF graphs. The graphs are represented as an abstract “model”. A model can be sourced with data from files, databases, URLs or a combination of these. A Model can also be queried through SPARQL and updated through SPARUL (SPARQL update language).

4.5 Activity 4.5: WEEP Integration (FLE, UVIE)

From the DoW: *This work package will integrate a workflow enactment engine that is capable of fulfilling the requirements of highly dynamic and interactive workflows, and which can be fully controlled by a user. Primarily, the engine is to be oriented toward data-intensive workflows and based on Grid services. These data-intensive workflows have a strong requirement for delivering and managing large volumes of data in scientific and commercial processes.*

No work planned for this period.

4.6 Activity 4.6: OGSA-DQP Integration (FLE, UEDIN)

From the DoW: *The OGSA-DQP software augments OGSA-DAI with query planning and distributed query capabilities, which allow large queries to be expressed and split across federated data resources. An objective of this work package is to deploy the OGSA-DQP extensions to the ISB, ensuring that the query planner benefits from the increased information provided about resources to make planning decisions. The University of Edinburgh will work with FLE in this activity.*

No work planned for this period.

4.7 Activity 4.7: General Enhancement to USMT

See Section 2.4.1 for the description of this activity.

The enhancements to the USMT in the coming period will still be driven by the requirements that arose during OGSA-DAI USMT integration activity. Apart from the features that already delivered in USMT v1.6 release, the following enhancements are expected in the next USMT v1.7 release:

- add the ProcessElement portType to the USMT resource, which corresponds to an ADMIRE Processing Element (PE):
 - hierarchical ServiceGroup-like construction;
 - recursive service instance destruction;
 - aggregate monitoring, including transformation made by the Gateway;
 - capture whether members are local or remote to the ProcessElement;
 - handle the fact that members can be contained in multiple ProcessElements.
- JavaBean object creation – JNDI files for resource creation;
- security, explicit trust delegation.

Moreover, general bug fixes and performance improvements will always be considered to the quality enhancement to USMT code base.

4.8 Activity 4.8: Standards Alignment (FLE, UVIE, UEDIN, UPM)

The DoW states:

The development of USMT has been based on Web services standards. More specifically, USMT is largely based on the Open Grid Services Architecture (OGSA) of the Open Grid Forum (OGF) and the WS-RF specifications developed in OASIS. Future USMT developments, and any modifications made as required by the DMI platform, will continue to be based on existing and developing standards.

There are two identifiable standards activities that apply to the coming period.

1. The Web Service Resource Access working group has started in W3C. This is the latest in a sequence of activities to provide the WSRF like capabilities of USMT. This development has been severely complicated by conflicts major vendors (notably IBM and Microsoft). This latest effort seems likely to resolve the issue finally. If successful this effort may suggest that USMT and the Globus be redeveloped to support these new standards.
2. Since our Registry approach will follow the WS-DAI-RDF interface UPM will track this activity.

A USMT Design and Implementation

The USMT Architecture is built on top of a set of fundamental core components. These components are engineered to contain a USMT foundation API layer and a Web Services communication API layer. The USMT API layer serves as the basic building block to its corresponding Web Services infrastructure. There are four components in the USMT platform, namely:

- Resource Property
- Resource Lifetime
- Notification
- Subscription

The design and implementation of these components are described below:

A.1 Resource Property

This component is designed to get access to and manipulate the value of properties of a resource. These properties should be created to reflect the characteristics of the underline resource. There are three sets of methods provided by USMT, for both WS level and API level access are as follows:

```
public GetResourcePropertyResponse
getResourceProperty(GetResourceProperty request);

public <ElementType>
List<ElementType> getResourceProperty(QName name);
```

```
public GetMultipleResourcePropertiesResponse
getMultipleResourceProperties(
GetMultipleResourceProperties request);

public List<Object> getMultipleResourceProperties(List<QName> names);
```

```
public UpdateResourcePropertiesResponse
updateResourceProperties(UpdateResourceProperties request);

public <ElementType> void
updateResourceProperty(QName targetQName, List<ElementType> value);
```

The USMT API shares the same name as WS level access operations, but strip the WS related XML overhead off the method parameters: instead of the “GetResourceProperty” parameter (which contains one QName only), the corresponding API level method directly accepts the Resource Property’s QName. The return parameter follows the same pattern: instead of returning an instance of the JAVA class modelling the SOAP response message, the API level method returns the appropriate runtime content of the requested Resource Property (or, void for updating a Resource Property). The methods make use of JAVA Generics whenever possible.

It has proved that, during the development of USMT, it is good practice to provide Resource Property specific operations, particularly in case of well-known Resource Properties, e.g. those are important for the USMT infrastructure. The following operations return the specific Resource Properties as indicated by the method name. The JAVA interface Javadoc documentation provides more detail on the semantics of these Resource Properties:

```
public List<W3CEndpointReference> getResourceEndpointReference();
public List<QName> getResourcePropertyNames();
public List<QName> getWSResourceInterfaces();
```

```
public QName getFinalWSResourceInterfaceRP();
```

A.2 Resource Lifetime

USMT provides infrastructure for automating Service Instance lifetime management. The “Lifetime” of a Service Instance is defined as follows:

“The lifetime of a Service Instance spans the existence of the underpinning Java class instance in the current Java Virtual Machine, from the invocation of any Class constructor up until the garbage collection of the class instance.”

A.2.1 Resource Creation

The process of a resource creation is, for the implemented service itself, a two-step process:

1. Constructor: A JAVA class constructor is called to instantiate the underpinning JAVA class instance. The USMT generic resource factory requires the default constructor present in the service implementation class. Specialised factory services may call other non-default constructors.
2. *onCreate(List<W3CEndpointReference> eprs)* method is called in the final phase of the Web Service Instance creation process. This allows the service developer to post-initialise and prepare the Service Instance before it is put into active service.

A.2.2 Resource Destruction

Service Instance destruction is straightforward. The destruction process may be initiated at any time during the lifetime of a Service Instance. USMT provides for two alternative methods of Service Instance destruction: Immediate, manual destruction, and automatic, timed destruction.

Immediate destruction may be initiated via Web Services operation invocation, or by invoking the corresponding USMT Service API method. In fact, the transparently implemented Web Service operation makes use of the USMT Service API method to accomplish immediate destruction.

The two corresponding operations for immediate termination are:

```
public DestroyResponse destroy(Destroy request);  
public void delegateDestroy(boolean graceful);
```

USMT also provides for Service Instances the concept of a configured “termination time”. This termination time, if not nil, defines the point in time when the Service Instance shall automatically initiate the destruction process as described for immediate Service Instance destruction. This point in time is modelled as a USMT Singleton Resource Property “TerminationTime”. A second Resource Property “CurrentTime” is provided modelling the server’s current time. The two Resource Properties together allow discovering timing skews between clients and the Service Instance, overcoming errors in time-related operations such as Service Instance lifetime. The generic Resource Property related operations are described earlier in this document. The specific operations are defined as follows:

```
public SetTerminationTimeResponse  
setTerminationTime(SetTerminationTime request);  
public void delegateSetTerminationTime(Duration duration);  
public void delegateSetTerminationTime(XMLGregorianCalendar dateTime);
```

The first method models the Web Service operation. The last two methods provide USMT Service API level manipulation of the *TerminationTime*. Note that the Web Service level method allows for

providing either an absolute time or a relative duration as contents in the request SOAP message model, *SetTerminationTime*.

A.3 Notification

Dynamic, interactive Web Services are often passive and unresponsive unless some sort of event notification mechanism is provided. Event notifications allow for more asynchronicity and interactiveness for Web Services and their clients.

USMT provides an event notification mechanism so that Web Services can notify interested clients of certain events of importance within the Web Service.

USMT organises event management and identification around the concept of topics. A topic describes a certain circumstance within the USMT Web Service for which certain events may happen. Topics are very abstract and can model extremely different circumstances.

In USMT, a topic is uniquely identified using XML QNames. A Web Service endpoint may subscribe to notifications on a topic with a Web Service that exposes that topic for subscriptions. The subscribing endpoint is called the “notification consumer”, and the endpoint emitting topic notifications to subscribers is called the “notification producer”. When an event occurs on a topic the notification producer sends a notification message to each notification consumer that is subscribed to that topic.

Often, Resource Properties and Notification Topics on some or all Resource Properties coincide. USMT supports this use case by providing a simple Boolean switch on Instance Properties. The QName of the corresponding Notification Topic is identical to the QName of the underlying Instance Property. No separate declaration of Notification Topics is necessary.

USMT provides transparent infrastructure for generic payload management for Notification Topics on Resource Properties. As soon as the `setContent()` or `setValue()` (for Singleton Resource Properties) method is called the subscribed notification consumers get notified of the updated Resource Property if it is configured as a Notification Topic. This is the single most important reason for the design of Resource Property content modifications as it is.

A.4 Subscriptions

Each time a notification consumer subscribes to a topic available on a USMT Web Service an instance of a Subscription Web Service is created. The Subscription Web Service is a default Web Service that is always deployed when a USMT Nucleus is started. A USMT Subscription is all the same a USMT Web Service, i.e. it provides Resource Properties, Resource Lifetime management and even Notification Topics.

Usually USMT manages Subscriptions transparently to your Web Service. Sometimes, however, Web Services are themselves consumers of notifications, i.e. act in the role of a notification consumer in this framework. In many cases the notification producer will be another USMT Web Service.

As stated above, USMT Subscriptions provide all management features that any other USMT Web Service provides as well:

1. USMT Subscriptions are created using a factory pattern.
2. USMT Subscriptions have a default lifetime of “indefinite”, i.e. the initial termination time is set to “nil”
3. USMT Subscriptions may be destroyed immediately.
4. USMT Subscriptions may receive a non-nil termination time, causing it to automatically destroy itself at the set time
5. USMT Subscriptions may be paused.
6. USMT Subscriptions may be activated, deactivated, commissioned, etc.

B Services Integrated with the USMT Platform

B.1 OGSA-DAI Services

The public face of OGSA-DAI comprises six services that give access to one of six corresponding resources. Each resource has a set of resource properties, some of which are common to all resources such as properties relating to the resource's lifetime (eg. TerminationTime). All services have a common set of operations that give access to these resource properties and also manage the resource lifetime. Some services have additionally operations specific to the resources they access. We list here the details of each service but omit the common lifetime management operations and resource properties and also those common operations for accessing resource properties. Note that resource properties all have full qualified names but here we omit the namespace part for simplicity.

Data request execution service (DRES). Clients use service to submit workflows, create sessions and the get the request status of synchronous requests. It has the following resource properties:

- SupportedActivities: gives access to the OGSA-DAI activities that can be targeted at this resource. For the DRES this is all those activities that are targeted at any data resource but simply process data without any databases or other resources with state. These activities are typically transformation and merging type activities.

The DRES has the following operations:

- Execute: executes an OGSA-DAI workflow either synchronously or asynchronously.

Data resource information service (DRIS). Clients use this service to this to query information about a data resource, e.g. product name, vendor, version. It has the following default resource properties but different data resource implantations may specify other resource properties specific that that data resource:

- SupportedActivities: gives access to the OGSA-DAI activities that can be targeted at this resource.
- Product: the product of the data resource being exposed.
- Vendor: the vendor of the data resource being exposed.
- Version: the version of the data resource being exposed.

The DRIS has no specific operations.

Data sink service. Clients can use this service to push data into OGSA-DAI workflows. It has the following resource properties:

- SupportedActivities: gives access to the OGSA-DAI activities that can be targeted at this resource.
- DataSinkStatus: gives to the status of the data sink, e.g. waiting, processing or completed.

The data sink service has the following operations:

- PutBlock: puts a single block of data into the data sink.
- PutNBlocks: pust N blocks of data into the data sink.
- PutFully: puts a complete set of data into the data sink.

Data source service. Clients can use this service to extract data from OGSA-DAI workflows. It has the following resource properties:

- **SupportedActivities:** gives access to the OGSA-DAI activities that can be targeted at this resource.
- **DataSourceStatus:** gives to the status of the data source, e.g. waiting, processing or completed.

The data source service has the following operations:

- **GetBlock:** gets a single block of data from the data source.
- **GetNBlocks:** gets N blocks of data from the data source.
- **GetFully:** gets a complete set of data from the data source.

Request management service. Each new request sent to the DRES generates a new request resource that can be accessed via the request management service. Clients can use this service to monitor asynchronous request and receive the final data associated with the request. The service has the following resource properties:

- **RequestExecutionStatus:** a very simple high level status of the current request useful for polling or notification.
- **RequestStatus:** a details status of the request typically only read when the request has terminated. This includes the status of each activity, any error messages or data returned.

The request management service has no specific operations.

Session management service. Clients can use this to manage the lifetime of sessions. It has the following resource property:

- **SupportedActivities:** gives access to the OGSA-DAI activities that can be targeted at this resource.

The session management service has no specific operations.

The core OGSA-DAI functionality has been designed and implemented by the OGSA-DAI development team outside of the ADMIRE project. More details of the design can be found at the OGSA-DAI project website (www.ogsadai.org.uk).