




---

## ADMIRE D4.4 – Development and Deployment Report for USMT v3

---

Project Title	ADMIRE
Document Title	ADMIRE D4.4 – Development and Deployment Report
Deliverable Number	D4.4
Authorship	Vivian Lee, Amy Krause, Carlos Buil, Radek Ostrowski, Alex Wöhler, Dave Snelling
Document Filename	ADMIRE-D4.4-DevelopmentandDeploymentReport.docx
Document Version	1.0
Distribution Classification	Internal
Distribution List	<i>ADMIRE Project Team</i>
Approval List	<i>Radek Ostrowski, Project Manager, Executive Board</i>

### Document History

<i>Personnel</i>	<i>Date</i>	<i>Comment</i>	<i>Version</i>
V.Lee	08/02/2010	First draft	0.1
A. Wöhler	09/02/2010	Input from Alex Wöhler	0.2
V.Lee	10/02/2010	Input from Vivian Lee	0.3
A. Krause	11/02/2010	Input from Amy Krause	0.4
C. Buil	15/02/2010	Input from Carlos Buil	0.5
R. Ostrowski	24/02/2010	Input, and Review from Radek Ostrowski	0.6
D. Snelling	25/02/2010	Input from Dave Snelling	0.7
R. Baxter	26/02/2010	Further review; minor restructuring	0.8
V.Lee	01/03/2010	Review feedback incorporated, Reviewed by Radek Ostrowski	0.9
R.Baxter	03/03/2010	Final edit and signoff	1.0

**Contents**

**Contents..... 1**

**1 Executive Summary..... 4**

**2 WP 4: Summary: Project Month 24 ..... 5**

**2.1 Products delivered to other work packages..... 5**

**3 Progress against Planned Activities ..... 6**

**3.1 Activity 4.1: OGSA-DAI Integration (FLE, UEDIN) ..... 6**

        3.1.1 Planned activity..... 6

        3.1.2 Actual activity ..... 6

        3.1.3 Deviations from plan ..... 6

        3.1.4 Further details..... 6

**3.2 Activity 4.2: Enhanced Monitoring (FLE, UVIE, UEDIN)..... 6**

        3.2.1 Planned activity..... 6

        3.2.2 Actual activity ..... 7

        3.2.3 Deviations from plan ..... 7

        3.2.4 Further details..... 7

**3.3 Activity 4.3: Semantic Registry Integration (FLE, UPM)..... 7**

        3.3.1 Planned activity..... 7

        3.3.2 Actual activity ..... 7

        3.3.3 Deviations from plan ..... 8

        3.3.4 Further details..... 8

**3.4 Activity 4.4: Semantic Provisioning Services Integration (FLE, UPM)..... 8**

        3.4.1 Planned activity..... 8

        3.4.2 Actual activity ..... 8

        3.4.3 Deviations from plan ..... 9

**3.5 Activity 4.5: Advanced Data Management Integration (FLE,UVIE,UEDIN) ..... 9**

        3.5.1 Planned activity..... 9

        3.5.2 Actual activity ..... 9

        3.5.3 Deviations from plan ..... 9

        3.5.4 Further details..... 9

**3.6 Activity 4.6: OGSA-DAI-DQP Integration (FLE, UEDIN) ..... 9**

        3.6.1 Planned activity..... 9

**3.7 Activity 4.7: General Enhancement to USMT (FLE, UVIE, UEDIN, UPM) ..... 9**

        3.7.1 Planned activity..... 9

        3.7.2 Actual activity ..... 10

        3.7.3 Deviations from plan ..... 11

---

3.7.4	Further details.....	11
3.8	Activity 4.8: Standards Alignment (FLE, UVIE, UEDIN, UPM).....	11
3.8.1	Planned activity.....	11
3.8.2	Actual activity .....	11
4	Risks and Issues .....	12
4.1	Project issues .....	12
4.2	Significant problem reports .....	12
4.3	New risks.....	12
5	Plans for next Period .....	12
5.1	Activity 4.1: OGSA-DAI Integration (FLE, UEDIN) .....	12
5.2	Activity 4.2: Enhanced Monitoring (FLE, UVIE).....	12
5.3	Activity 4.3: Semantic Registry Integration (FLE, UPM).....	13
5.4	Activity 4.4: Semantic Provisioning Services Integration (FLE, UPM).....	13
5.5	Activity 4.5: Advanced Data Management Integration (FLE, UVIE) .....	13
5.6	Activity 4.6: OGSA-DQP Integration (FLE, UEDIN) .....	14
5.7	Activity 4.7: General Enhancement to USMT.....	14
5.8	Activity 4.8: Standards Alignment (FLE, UVIE, UEDIN, UPM).....	14
6	References.....	15
A	USMT Design and Implementation .....	16
A.1	Resource Property .....	16
A.2	Resource Lifetime .....	17
A.2.1	Resource Creation .....	17
A.2.2	Resource Destruction .....	17
A.3	Notification .....	18
A.4	Subscriptions .....	18
B	ADMIRE Gateway Services Integrated with USMT Platform.....	19
B.1	DISPEL Gateway Service.....	19
B.1.1	DISPEL Parser .....	19
B.1.2	DISPEL Graph Processing and Enactment .....	19
B.2	OGSA-DAI Services .....	21
B.3	Enhanced Monitoring.....	22
B.3.1	Design and Implementation of DSMON.....	22
B.3.2	DSMON Services .....	23
C	Semantic Registry Design and Implementation .....	25
C.1	Updated Semantic Registry Design .....	25
C.2	Semantic Registry Implementation .....	26

---

---

**C.3 Semantic Registry Client Interface ..... 27**

---

## 1 Executive Summary

This document is ADMIRE Deliverable D4.4 and describes progress on Work Package 4 during months 18 to 24 of the ADMIRE project.

At project month 24 ADMIRE WP4 is essentially on target against original plans, having successfully deployed the V1.7.0 version of USMT. In terms of PM24's explicit goals for WP4, progress can be summarised as follows:

**Enhanced USMT Gateway service (achieved):** (carried over from PM18) this period saw the enhanced ADMIRE Gateway based on OGSA-DAI data services hosted within a USMT infrastructure.

**Enhanced USMT presentation layer for OGSA-DAI (achieved):** (carried over from PM18) integration of USMT and OGSA-DAI was completed with the porting and successful deployment of the two remaining services (DataSinkService and DataSourceService).

**Second version of the Registry Service (achieved):** the first Semantic Registry has been deployed successfully at EPCC and later at UPM. The second release that containing a new Update function will be deployed at UPM at a later time.

**Extend Gateway with Repository service (achieved):** an initial prototype of the ADMIRE Repository has been developed as part of enhanced function provided by the Gateway service.

The Gateway service and Semantic Registry are the main development focus during months 18 to 24. The integration of OGSA-DAI with USMT and the integration of Enhanced Monitoring activities is still ongoing, but was more focused on performance enhancement and modifications that adapt to the potential changes that maybe brought in by a new USMT platform.

The initial identification of benefits provided by advance data management technologies activity was on hold due to time constrains. However, the project sees the technologies adoption by the Repository in the very near future.

---

## 2 WP 4: Summary: Project Month 24

The objective of Work Package 4 is to develop and enhance the core service infrastructure that form the foundations of the DMI platform which has been 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, 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 is being 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-DAI 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 18 to 24 of ADMIRE, WP4's main focus has been to develop enhanced Gateway services and the Semantic Registry. A new version of USMT has also been distributed to the project, providing enhancements to other tasks and work package of the project. Other activities included the completed porting of OGSA-DAI to USMT, and integration of Enhanced Monitoring. All these integration activities have proceeded well, and versions of all these services were released as part of the ADMIRE Platform Release 3 in PM24. A basic Repository service has also been developed as part of the Gateway functionality. This Repository will be further refined to adopt advanced data management technologies as planned in Section 5.5 in the near future.

### 2.1 Products delivered to other work packages

FLE released one version of USMT to the ADMIRE project, primarily to address the requirements from UEDIN during the Gateway development and integration of OGSA-DAI with USMT:

- USMT for Admire v1.7.0 was released to ADMIRE on 26 February 2010.

Other releases from WP4 package work include:

- OGSADAI-USMT v1.0.0 on 2 March 2010;
- Database Monitoring Service (DSMON) v0.7 on 10 January 2010;
- ADMIRE Registry v0.2 on 25 February 2010.

---

## 3 Progress against Planned Activities

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

#### 3.1.1 Planned activity

The OGSA-DAI [1] 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.2 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.

#### 3.1.2 Actual activity

The planned task of implementing Exception and Error propagation from USMT via the presentation layer to OGSA-DAI has been moved to the next period. A new, and more important task was identified. It consisted of implementing additional activities that extend OGSA-DAI and provide support for the ADMIRE use cases. Generic Activities allow the client to execute scripts on the server. Three patterns have been identified and implemented:

- **GenericActivity:** this activity can run any script that reads from OGSA-DAI activity inputs (BlockReader) and writes to OGSA-DAI activity outputs (BlockWriter).
- **TupleJoinActivity:** this activity provides forms the product of two data input streams and applies a script to the product.
- **TupleTransformActivity:** this activity transforms an input data stream by applying a script.

This makes it easy for developers to install their own extensions, for example to collect statistics on a dataset, whilst keeping the computation close to the data.

New functionality of the Gateway, the processing of distributed DISPEL [3][4] requests, required a new activity to support data transfer between distributed requests. This activity contacts a remote Gateway and initiates the data transfer when data becomes available.

#### 3.1.3 Deviations from plan

It was found more benefiting to work on Generic Activities, rather than on planned implementation of Exception and Error propagation, which has been postponed until the next project phase.

#### 3.1.4 Further details

Further technical detail on the integration of OGSA-DAI services with USMT within the ADMIRE Gateway can be found in Appendix B.2.

### 3.2 Activity 4.2: Enhanced Monitoring (FLE, UVIE, UEDIN)

#### 3.2.1 Planned activity

The initial implementation of a USMT enabled DSMON is completed, but work will continue to ensure a stable service. In this period this activity will focus on the following two areas:

- **Collect and process monitoring information:** from the implementation of DSMON and OGSA-DAI workflow monitoring we now have tools at hand to produce a rich set of interrelated monitoring information. The next step is to collect it and store it in a maintainable manner to allow pattern mining on top of it, which will, in the last step, feed back to various optimizations.

---

Resource monitoring: this task amounts to gathering information about the service execution environment, which gets increasingly difficult in today's virtualized environments. We have to elaborate which information (gathered in different ways by different tools) is relevant to and complements the rest of our monitoring activity.

### 3.2.2 Actual activity

In the latest project period, DSMON has been migrated to the latest version of USMT.

### 3.2.3 Deviations from plan

There are minor deviations from the plan. DSMON has been deployed at several sites, but the actual collection of monitoring information is behind schedule due to too few available real databases and associated re-occurring usage. With the planned establishment of various distributed databases for performance analysis of the overall ADMIRE infrastructure this issue will be resolved.

### 3.2.4 Further details

Further technical detail on the integration of Enhanced Monitoring services with USMT within the ADMIRE Gateway can be found in Appendix B.3.

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

### 3.3.1 Planned activity

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. Grimores [5], the semantic registries of the NeOn Toolkit [6] or the Metadata Registry of the Semantic Web Framework [7].

### 3.3.2 Actual activity

During the project month 18 and 24, a Registry has been deployed at UPM, NeSC and EPCC; and is being accessed regularly by other partners in ADMIRE. The Registry at UPM is the most used one and the other two are deployed mainly for testing purposes for the use cases. An Update function has been added to the Registry and now it is possible to register a PE with its related information (structural type, inputs and outputs) such that a set of instances is created in the RDF file associated with the Registry. Regarding the update of the registry (a RDF graph) it might be worth to keep track of the SPARQL Update<sup>1</sup> submission to the W3C. To integrate all the components in a simplicity way, a new interface was developed and implemented by a Registry client that is available to the ADMIRE components. The client interface offers three different operations. One operation retrieves the information for a given PE name, other operation retrieves all the PEs in the registry with their related information too, and the third operation updates the registry with a new PE and its related information. The WS-DAI-RDF specification has been discarded due to the current internal use of the Registry and the specific operations needed by the Process Designer. Currently the ADMIRE Registry is only accessed by ADMIRE components and there is no access from applications outside the ADMIRE project

The current focus of Registry development is the Update function, since the use cases of the PEs in ADMIRE are much clearer now, and the PEs themselves are described in the Platform ontology. DMI

---

<sup>1</sup> <http://www.w3.org/Submission/2008/SUBM-SPARQL-Update-20080715/>

---

workflows are built in the PD and tracking systems for the workflow running instances are needed. To keep track of these instances they have to be recorded in the Registry and therefore an Update function is needed. Currently, the Update function is implemented as a new activity for the RDF resource, which accesses the RDF file. This activity gets as input the PE name, structural type, inputs and outputs and creates a set of triples in the RDF file in which are stored the other PEs.

### 3.3.3 Deviations from plan

No deviations.

### 3.3.4 Further details

Further technical detail on the design and implementation of the ADMIRE Semantic Registry service can be found in Appendix C.

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

### 3.4.1 Planned activity

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) with the Workbench used to create PEs, and with the services that will use the annotations. In ADMIRE the current semantic provisioning services are:

- Annotation services: due to the requirements of the ADMIRE Registry, a new annotation service is needed. One of these requirements is an Update operation. This operation has an input PE with its related information, and the result of it is the RDF file containing all the existing PEs with a new one. Formally this is an annotation tool because it adds a semantic description (according to the Platform ontology, the basis of the Registry) to the PE.
- Metadata services: the metadata services in the ADMIRE context is the ontology repository in which the ontologies will be stored.

Due to the evolution of the ADMIRE ontologies (especially the Platform ontology) and the evolution of the ADMIRE Registry through addition of a new update operation, the Service and Data Description Assistant (SDDA) has become redundant. As explained in D5.4 [8], the SDDA has been replaced by the Registry's update operation, which is a type of semantic provisioning service, and sub-typed as an annotation service.

### 3.4.2 Actual activity

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

- The ADMIRE registry accesses the elements of the Platform ontology. The registry is able to query using SPARQL instances of this ontology and it is also able of updating the RDF repository. Both the gateway and the PD access the registry via the registry client toolkit for both, querying and updating the RDF repository.

Still to be completed, if necessary:

- 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);
- a distributed query system that queries several registries may be necessary. If a gateway needs to know if a specific PE is located in any of the available registries a distributed query system may be necessary;
- a reasoning service is available if necessary for the ADMIRE services.

A Registry client interface was created as part of the Gateway. This interface provides methods to query the Registry for processing elements by name or list all available processing elements. An implementation of this API interacts with the ADMIRE Registry and it is used by the Gateway to

---

validate processing elements referenced in a DISPEL document. Over the next reporting period, the Registry client will be extracted from the Gateway and become a standalone library so it can be used by other components such as the workbench.

### **3.4.3 Deviations from plan**

As noted, the SDDA has been substituted by the Registry Update operation.

## **3.5 Activity 4.5: Advanced Data Management Integration (FLE,UVIE,UEDIN)**

### **3.5.1 Planned activity**

In the DoW this activity provides a catch-all for the introduction of additional DMI functionality as and when required by the ADMIRE Architecture and Application developments.

With the rationalisation of the scope of WPs 4 and 5 into capturing everything “below” and “above” the ADMIRE “hourglass” this activity becomes the best place to report progress on the provision of DMI language processing services within the ADMIRE Gateway. From this reporting period onwards we will capture progress on the ADMIRE Gateway DISPEL Service here.

The DISPEL Gateway Service and attendant DISPEL parser is the component which parses DISPEL documents and creates the corresponding DISPEL graphs.

### **3.5.2 Actual activity**

The DISPEL parser component was extended in this reporting period and now supports functions and the creation of composite processing elements. Several bug fixes were implemented.

The ADMIRE Gateway Service now supports processing of DISPEL requests that are distributed across remote Gateways and manages the necessary data transfers between processing instances. A number of new extension points were added to facilitate architecture experiments that modify and optimize DISPEL graphs. OGSA-DAI services that are available to a gateway can be specified in the configuration or discovered automatically. The Gateway has added support for registry updates to register new processing elements, however this is not supported by the current implementation of the registry.

### **3.5.3 Deviations from plan**

No deviations.

### **3.5.4 Further details**

Further technical detail on the integration of DISPEL processing services with USMT within the ADMIRE Gateway can be found in Appendix B.1.

## **3.6 Activity 4.6: OGSA-DAI-DQP Integration (FLE, UEDIN)**

### **3.6.1 Planned activity**

No planned activity this period.

## **3.7 Activity 4.7: General Enhancement to USMT (FLE, UVIE, UEDIN, UPM)**

### **3.7.1 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.

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 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.

### 3.7.2 Actual activity

In the fourth period of the project, major requirements to USMT platform have shifted from core functional enhancements to utility improvements. This is due to the work of the USMT platform integration to ADMIRE components is closing to completion, and it is time to move onto platform level of integration. Therefore, at this stage, capabilities of remotely control services instances, and communications between service instances across different networks have become increasingly important to the ADMIRE platform as a whole. For instance, to perform an automated system test, the capability of being able to shut down existing running USMT services instances remotely is highly crucial.

Solutions to the issues, general platform enhancement, and bug fixes during this period are described below (along with their problem-report ticket number from the ADMIRE Trac PR system).

#### Notification PullPoint

In many cases, where a client is connected to the Internet via a provider or a network address translator, the Notification cannot be delivered to the client who subscribed it, since the IP address assigned to the client from the Internet provider is not exposed to the outside world. To better solve this problem, USMT introduces a *PullPoint Notification* architecture, in addition to the current *Push Notification* model. This allows a client to get any number of notification messages accumulated at the server side about a subscribed topic at anytime. This solution solves the IP address issue, but it also has an obvious drawback, that the client cannot be notified synchronously as the *ResourcePropertyValueChange* notification happens. This feature was designed in the previous project period, and implemented in this fourth period.

#### A way of shutting down USMT server (ideally as an ant target) (Trac #145)

To integrate Gateway/USMT tests with the rest of the test framework, a way of stopping the server remotely is required, since system tests are automated and they need to restart all services prior to running the tests. An ant target to startup the server has already been provided, but shut down servers through ant task is not straightforward. Many solutions have been considered, and the final decision is to record the Java processing ID - when starting a server remotely using ant build task, a shell script is passed in, which reads the current Java processing ID into a temporary file. At shut down stage, another shell script is executed, which terminates the process with ID in the temporary file. All the

---

information contained inside the temporary directory inclusive is cleaned up after the server shut down.

### **Problem with DataSourceResource EPR (Trac #328)**

While executing request to Gateway, client that uses CTK (Client Toolkit) throws an exception. It seems that problem is linked to DataSourceResource EPR. A method “patchEPR()” that fixes similar issue has been attempted without success. Request is received by Gateway, but problem appears before retrieving results. This issue was finally resolved by force compiler to call the libraries from JAXWS [9] package, rather than the equivalent ones from JDK. This is realized by passing in a JVM argument into the ant task:

```
<jvmarg value = “-Djava.endorsed.dir=lib”/>
```

where “lib/” is the directory that contains the libraries that need to be loaded.

### **Use case for USMT resource property (Trac #324)**

A sixth use case has been added to the USMT dummy project, which shows how to create static resource property that contains classes generated from an XML schema as value.

## **Repository Gateway integration**

Apart from the above USMT platform enhancement related activities, WP4 also developed a first hash map based prototype Repository, which accepts “Create” and “Get” operations conveyed by a DISPEL document. This Repository is internal to the Gateway, as an additional capability that can be achieved by the DISPEL document. At a later stage, a key value pair database provided by Hadoop MapReduce [9] storage will be placed instead of hash map to massively enhance the scalability.

### **3.7.3 Deviations from plan**

There is no major deviation from the original plans during this period.

### **3.7.4 Further details**

The design of the core USMT service framework is discussed in more detail in Appendix A.

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

### **3.8.1 Planned activity**

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.

In addition to OGSA and OGF activities, the Web Service Resource Access (WSRA) working group has started in W3C, and is performing very actively. This is the latest sequence of activities to provide the WSRF like capabilities of USMT.

### **3.8.2 Actual activity**

In project months 18 to 24, the WS Resource Access TC met on regular teleconferences and held two Face-to-Face meetings in November and January. The January meeting was hosted by Fujitsu Labs Europe, using a venue provided by Fujitsu Labs America. Technical progress has been very good and the TC is aiming to do a Last call in the next few months. In the W3C, the Last Call is a final check within those interested in implementing the specification, but not active in the TC to check on the specifications. This period also saw a moratorium on new issues. This has meant that a number of minor issues are by default delayed, but this has allowed the TC to reach a point where only editorial issues remain.

---

The OGF (Open Grid Forum) Open Grid Services Architecture (OGSA), and its associated specifications, defines consistent interfaces through Web services to components of the Grid infrastructure. Both the Web and Grid communities stand to benefit from the provision of consistent and agreed Web service interfaces for data resources and the systems that manage them. The document draft on “Web Service Data Access and Integration – The DM Realisation (WS-DAI-DM) Specification” to be submitted for public comments presents a specification for a collection of access interfaces for a data mining process, which extends interfaces defined in the “Web Services Data Access and Integration” document (WS-DAI [11]). The specification can be applied in regular Web services environments or as part of a Grid fabric.

## 4 Risks and Issues

### 4.1 Project issues

WP4 development over this period is relatively stable. There is no major issue to report.

### 4.2 Significant problem reports

The technical problems noted above include their Trac ticket numbers.

### 4.3 New risks

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

## 5 Plans for next Period

The DoW has the following activities for M24-27 and M27-30:

- M24-27: Improve USMT performance.
- M27-30: Add automated self-management capabilities for reliability, resilience and scalability. Add OGSA-DQP services to the USMT.

The updated deliverable description is:

### **Development Progress Report: requirements for USMT V4**

This report will describe USMT performance enhancement work required and the evolution of the platform itself.

As part of the USMT enhancement work required by ADMIRE, it is expected that a number of self-management services will be developed for the USMT, which will allow for the automated monitoring and management of services within an ADMIRE specific context. This work will be described in this report, including the designs of the ADMIRE specific self-management services.

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

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

The next concrete task in this activity is to implement Exception and Error propagation from USMT via the presentation layer to OGSA-DAI.

### 5.2 Activity 4.2: Enhanced Monitoring (FLE, UVIE)

The initial implementation of a USMT enabled DSMON is completed, but work will continue to ensure a stable service. In the upcoming period this activity will focus on the following two areas:

- Collect and process monitoring information: from the implementation of DSMON and OGSA-DAI workflow monitoring we now have tools at hand to produce a rich set of

interrelated monitoring information. The next step is to collect it and store it in a maintainable manner to allow pattern mining on top of it, which will, in the last step, feed back to various optimizations.

- Resource monitoring: this task amounts to gathering information about the service execution environment, which gets increasingly difficult in today's virtualized environments. We have to elaborate which information (gathered in different ways by different tools) is relevant to and complements the rest of our monitoring activity.

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

In Section 3.3 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 a 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.

This integration is done through the Registry client interface. This interface has the following operations:

- ProcessingElementDescriptor lookupProcessingElement(String name)
- List<ProcessingElementDescriptor> lookupProcessingElementList();
- List<String> lookupProcessingElementsName()
- void registerProcessingElement(ProcessingElementDescriptor descriptor, String implementation)
- String lookupImplementation(String name)

This interface is implemented by the Registry client toolkit and used by both the Process Designer and the DMIL parser in the ADMIRE Gateway. More detail about this interface can be found in Appendix B.

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

The SDDA has been substituted by the Registry's Update function. The integration is done through the interface implemented by the Registry toolkit. In this interface there are operations for query and update the Registry. The PD and the DMIL parser are currently using these interfaces for querying the Registry. The next stage is for the PD to use the Registry's update operation. Still, the RDF repository is Jena due to its capabilities for scaling up and inference.

### 5.5 Activity 4.5: Advanced Data Management Integration (FLE, UVIE)

From the DoW: *To further enhance the ADMIRE platform, more advanced data management technologies will be carefully investigated in a broader vision. Potential candidate of these technologies include streaming data, Hadoop, MapReduce, Amazon S3 and maybe more. Integration to the ADMIRE platform will be carried out once significant advantages are identified.*

The current implementation of the DISPEL language parser has matured into a prototype, which handles the requirements of the architecture scenario, therefore no further development is planned in the next project period, except reactive support for requirements raised by the architecture group and bug fixes.

Meanwhile, a first attempt to explore the potential benefit of these advanced data management technology is planned to adopt Hadoop MapReduce as storage for the Repository service in the very near future.

## 5.6 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.*

The first introduction of OGSA-DAI-DQP capabilities will be addressed in PM24-30. This work will be coordinated with the Gateway optimisation work planned for WP2.

## 5.7 Activity 4.7: General Enhancement to USMT

See Section 3.7 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 integration between different components and the OGSA-DAI USMT integration activity. Apart from the features that already delivered in USMT v1.7.0 release, the following enhancements are expected in the next USMT v1.7.x 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.
- security, explicit trust delegation;

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

## 5.8 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.*

The standard activity in the Open Grid Services Architecture (OGSA) has been quite recently. On the other hand, the Web Service Resource Access (WSRA) working group has started in W3C, and is performing very actively. This is the latest sequence of activities to provide the WSRF like capabilities of USMT. The specifications should be published within the next six months. The ADMIRE project has not yet decided if some implementation experience is needed.

In the OGF, the WS-DAI-DM specification will proceed with the submission process. Further elaboration and contribution are also expected.

---

## 6 References

- [1] The OGSA-DAI Project <http://www.ogsadai.org.uk/>
- [2] Vivian Lee and work package partners. ADMIRE – Development and Deployment Report for USMT V2: capabilities of USMT V2. Deliverable report D4.2, the ADMIRE Project, Feb 2009.
- [3] Malcolm Atkinson, Peter Brezany, Carlos Buil Aranda, Oscar Corcho, Jano van Hemert, Ivan Janciak, Alexander Woehrer, and Gagarine Yaikhom. Report on progress of model, language and ontology research. Deliverable report D1.6, the ADMIRE Project, Feb 2010.
- [4] Malcolm Atkinson, Peter Brezany, Amy Krause, Jano van Hemert, Ivan Janciak, and Gagarine Yaikhom. DISPEL: Grammar and Concrete Syntax, version 1.0. Deliverable report D1.7, the ADMIRE Project, Feb 2010.
- [5] 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.
- [6] W. Waterfeld, M. Weiten, P. Haase, H. Cunningham, M. Dzbor, R. Palma, (2007) Specification of NeOn Reference Architecture and NeOn APIs. Deliverable D 6.2.1, NeOn Project, March 2007
- [7] R. García-Castro and M.C. Suárez-Figueroa (eds), “D 1.2.4 Architecture of the Semantic Web Framework”, Knowledge Web Deliverable, Feb 2007.
- [8] Amy Krause, Ivan Janciak, Michal Laclavik, Branislav Simo, Maciej Jarka, Marek Lenart, Yuzhang Han, and Adrian Mouat. ADMIRE – Tools Development and Tools Integration Report. Deliverable report D5.4, the ADMIRE Project, Feb 2010.
- [9] JAX-WS reference Implementation: <https://jax-ws.dev.java.net/>
- [10] Hadoop MapReduce: <http://hadoop.apache.org/mapreduce/>
- [11] M. Antonioletti, M. Atkinson, S. Laws, S. Malaika, N. W. Paton D. Pearson and G.Riccardi. *Web Services Data Access and Integration – The Core (WS-DAI) Specification, Version 1.0.* Draft, Global Grid Forum, 2006.

## 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 ADMIRE Gateway Services Integrated with USMT Platform

### B.1 DISPEL Gateway Service

The DISPEL Gateway Service is the primary point of interaction between the tools of the ADMIRE Workbench and Portal and the data workflow enactment and monitoring services in the body of the Gateway. The Gateway Service's primary purpose is the handling of DMI requests captured in DISPEL documents.

#### B.1.1 DISPEL Parser

The DISPEL parser is the component that parses a DISPEL document and produces a DISPEL graph. It is part of the ADMIRE Gateway.

In the reporting period, the DISPEL Processor was extended to support the use of functions and the creation of composite Processing Elements. Composite Processing Elements can be used within the same DISPEL document, or can be registered and stored using the 'register' keyword. Constructed graphs must be marked for submission with the 'submit' keyword.

A DISPEL document may contain zero or more graphs submitted to a Gateway. Additionally a DISPEL document may contain zero or more composite Processing Elements to be registered. Composite Processing Elements are the only objects whose registration is supported by the DISPEL processor.

Functions may not be registered and stored for later use yet but they can be declared and called from within the same DISPEL document.

Several bugs were fixed.

Future development activity will cover the continued testing of the DISPEL parser infrastructure with use cases, fixing bugs as they arise and adding functionality as is required for use cases and architecture experiments.

#### B.1.2 DISPEL Graph Processing and Enactment

In the reporting period, extension points were added to the Gateway to facilitate architecture experiments. A framework that allows registration of composite Processing Elements was added although the Registry implementation does not support updates yet.

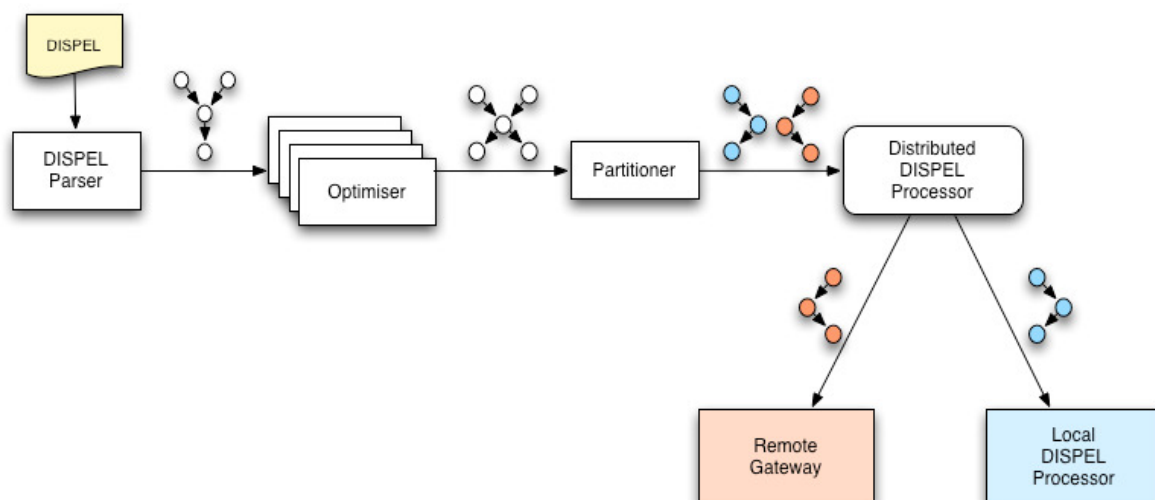


Figure 1: DISPEL processing in the Gateway

Figure 1 shows the flow of a DISPEL request through the distributed DISPEL processor in the Gateway. First, the DISPEL parser (see Appendix B.1.1) parses the DISPEL request and produces a set of submitted DISPEL graphs. To each resulting graph a chain of optimisers (or graph transformations) is applied. This chain of optimisers is configurable and new optimisers can easily be added to it. The transformed graph is then passed to the partitioner which decides how the graph is split and distributed across remote Gateways and the local processor. DISPEL documents are generated for graphs that are marked for remote execution and these are submitted to their respective target Gateways for enactment. Any required data transfers between Gateways are set up by inserting external outputs and external inputs in the DISPEL graph. Local processing is done by the local DISPEL processor (see description below).

If a DISPEL document contains a registration request, the composite Processing Element is extracted and a descriptor is produced, along with a DISPEL script that constructs the graph of the composite PE. This descriptor is registered with the Registry, while the implementation (the DISPEL script) is stored in the Repository. Note that the DISPEL processor extracts composite Processing Elements for registration but in the reporting period the Registry did not support updates. Hence this functionality could only be tested on a local, in-memory Registry.

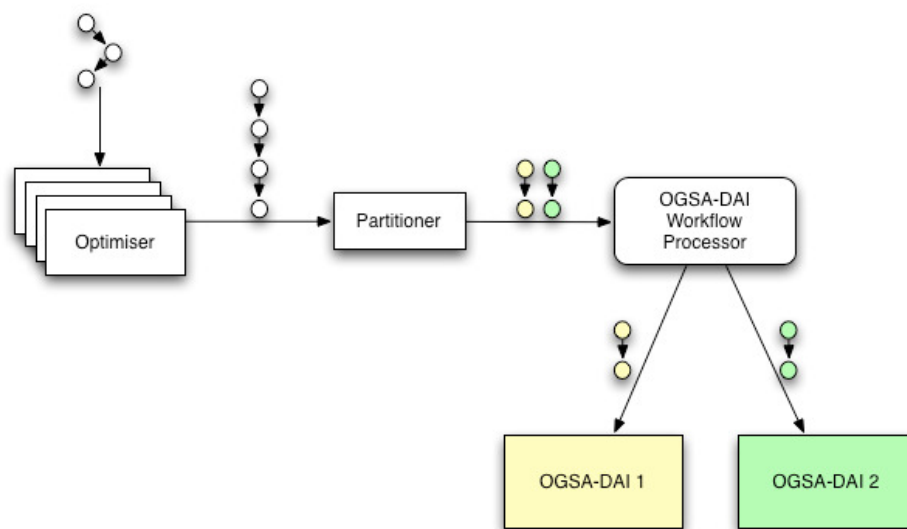


Figure 2: Local processing of a DISPEL graph in the gateway.

Figure 2 illustrates the local DISPEL processor which enacts a DISPEL graph on a number of OGSA-DAI services. The structure of the local DISPEL processor is very similar to the distributed DISPEL processor above. Again, a chain of optimisations, specific to this Gateway, is applied before the graph is partitioned. The current implementation of the partitioner takes the locations of data resources into account. However, this is an extension point to allow for other implementations with more advanced cost models to be plugged in easily instead. Once the graph has been split, it is converted into a set of OGSA-DAI requests which are submitted to their respective target services. Data transfers between services are set up as required.

If a DISPEL request contains a Processing Element that is marked as an external input, the local processor converts it to an OGSA-DAI data transfer activity. This activity contacts a specified gateway and waits for it to publish the EPR of a data transfer service (which is implemented as an OGSA-DAI data source). Data is then retrieved from the service at the published EPR. If an external output is encountered, an OGSA-DAI data source is created by the processor and the EPR of this service published as a resource property.

Requests that use registered composite Processing Elements are handled by an optimiser that looks up Processing Element descriptors in the Registry. If a PE is registered as a composite the optimiser retrieves the corresponding DISPEL script from the repository, constructs the graph, and substitutes it for the composite Processing Element.

Note that optimisers are pluggable; a different optimiser could apply a cost model and decide whether a composite PE is expanded or not depending on the availability of implementations or their cost.

## B.2 OGSA-DAI Services

ADMIRE has built its core dataflow enactment services on top of the OGSA-DAI data workflow engine. OGSA-DAI functionality is captured in six WSRF-compliant *services* and over 100 *activities* onto which ADMIRE Processing Elements are mapped.

OGSA-DAI is composed of six services that give access to one of six corresponding resources. Each resource has a set of resource properties, some of these are common to all resources such as properties relating to the resource's lifetime such as 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](http://www.ogsadai.org.uk)). A list of current OGSA-DAI activities available as ADMIRE Processing Elements can be found at <http://sourceforge.net/apps/trac/ogsadai/wiki/ActivityList>.

## B.3 Enhanced Monitoring

### B.3.1 Design and Implementation of DSMON

Data source related monitoring information is going to have a similar impact for efficient data access and integration on the Grid as the already utilized resource monitoring has, e.g. for fault tolerance and job scheduling, and is also crucial for various areas related to distributed data management.

In order to be applicable to a wide range of data access and integration related areas and scenarios, a data source monitoring component has to fulfill the following requirements:

- Support different monitoring granularities. The various consumers of the monitoring information will need only parts of the whole information provided. A health-status service might just be interested in the connection time to see if a certain data source is online while a data integration service will be interested in the attributes of a certain table or database. The interface has to support coarse grained (database level) as well as fine grained (table and/or column level) monitoring information.
- Customizable to various needs. A relational database management system exposed on the Grid can host multiple databases with a broad variety of schemas not relevant (also for security reasons) to

---

the outside world. Specific user tables are needed, while internal system tables are not. The monitoring component has to support mechanisms to define which database, schema and tables to include or exclude from the monitoring process in a flexible manner.

- Little encumbering of the target data source. The monitoring process is an important, additional overhead task for a data source. It shouldn't flood the data source with requests in a way that it affects day-to-day business. This implies to keep the requests as simple as possible to gather the needed monitoring information as well as finding a suitable frequency for this process.
- Support push concept rather than polling one. A consumer of the monitoring information shouldn't be forced to continuously poll our service if something has changed. Rather, the client should have the ability to let the service know in what type of changes he is interested (called subscription) and then be notified by the service. The interface has to support some kind of notification mechanisms.
- Virtualization. Available metadata information about data sources are highly heterogeneous in terms of how to get them and what they contain. The monitoring service has to provide a uniform interface to this information as well as a homogeneous view on them.
- Loosely coupled. The component is neither tightly coupled to a specific data access middleware nor necessarily tightly coupled with other components.

Our current research prototype of DSMON, based on USMT 1.6.3, comprises two services for clients: one representing the interface to database resources and the other one to its related table resources. DSMON currently supports MySQL, PostgreSQL and Oracle databases. Our implementation supports a homogeneous representation of all three commonly used histogram types (value based, height based, compressed) for numerical columns.

In order to be generally applicable, the request to gather the needed metadata information by the monitoring component is done via a pull model over JDBC. This means that it connects to the relational database and queries the system tables in a given frequency. Our DSMON component uses one JDBC connection per database (and its related tables), which gets closed again after its usage to collect the monitoring information. This is necessary to get a real connection time value as workload/performance indicator; although we know that opening a connection is a quite resource-expensive step in database transactions requiring multiple separate network round-trips.

The setup of our monitoring component for a database is done via two XML files. One holds the connect information to the database, the other allows to define what monitoring information should be provided and of which schemas and tables of a database to expose metadata.

### B.3.2 DSMON Services

The public face of DSMON comprises three services that give access to one of two resources, namely databases and associated tables. 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.

**DSMONSingleton service.** Loaded at hosting environment startup, this singleton service without any specific resource properties and operations makes sure that all target database management systems are represented by an instance of DSMONDatabase service and are correctly configured.

**DSMONDatabase service.** Represents a database resource for the client and dynamically creates/destroy's child DSMONTable service instances according to their current number of tables in the RDBMS. The service offers the following operations:

- `listResources`: allows to get list of EPR for other DSMONDatabase instances
- `listChildResources`: allows to get list of EPR for child tables of this RDBMS

The service has the following resource properties:

- connection time in ms
- refresh interval in ms
- product name
- product version

**DSMONTable service.** Represents a table resource for the client. The service offers no operations, just the following resource properties:

- table name
- logical schema
- indexes: contains information about available indexes like columns, order, type, etc.
- histograms: contains information about available histograms in a unified manner.
- exact continuous data statistics: contains a pre-defined set of exact statistics if available. This development is based on our work described in more detail in [14].

## C Semantic Registry Design and Implementation

### C.1 Updated Semantic Registry Design

A new interface for accessing the Registry has been implemented. Previously, access to the Registry required the creation of an OGSA-DAI client activity and a SPARQL query for accessing the desired Registry elements. This required the developer of the client activity to have some knowledge of SPARQL querying. Instead a Java interface was designed to access the Registry with a set of basic operations. These operations are:

- `ProcessingElementDescriptor lookupProcessingElement(String name)`
- `List<ProcessingElementDescriptor> lookupProcessingElementList();`
- `List<String> lookupProcessingElementsName();`
- `void registerProcessingElement(ProcessingElementDescriptor descriptor, String implementation)`
- `String lookupImplementation(String name)`

These interfaces are implemented by methods in which each one sends a specific query to the RDF resource and the results are processed accordingly.

This interface was firstly intended as an interface for a dummy registry but is now implemented by methods which access the RDF resource which in turn accesses the ADMIRE Registry proper. One of the last methods implemented has been the update method. The implementation of this operation required the adding of new activities to the RDF resource, both client and server side. Figure 3 represents this graphically.

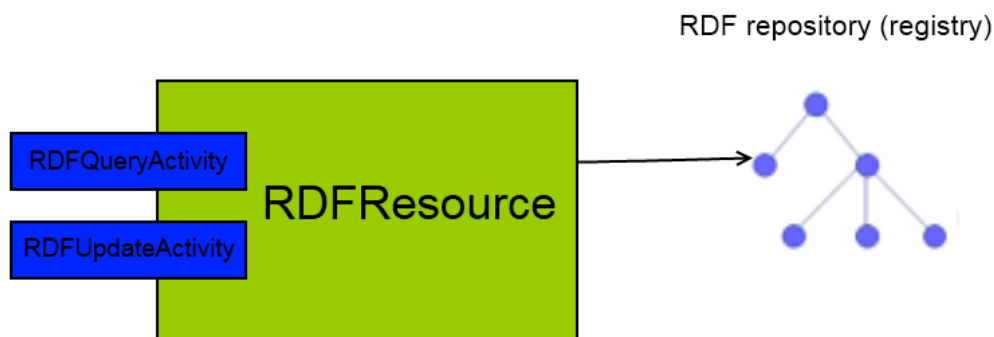


Figure 3 : New update operation

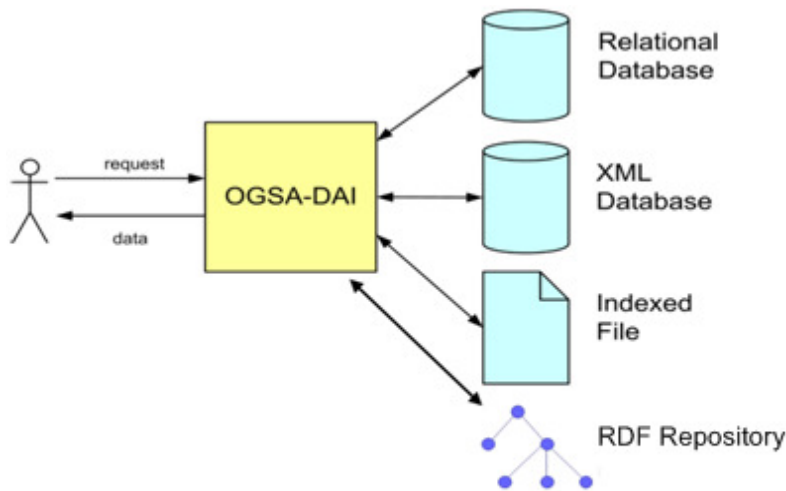


Figure 4 : RDF Resource Extension

## C.2 Semantic Registry Implementation

The main elements of the new OGSA-DAI RDF resource are:

- **RDFResource:** the RDFResource class implements the OGSA-DAI DataResource interface. The implemented methods are *getState()*, which returns the configuration parameters of the “RDFResource”, including the necessary elements for the correct execution of the data resource; *initialize()*, which creates the RDF data resource and initializes it, allowing the OGSA-DAI persistence and configuration components to configure the data resource class with its configuration (the RDF mapping file and other configuration files); and *queryRepository*, which sends prepared SPARQL queries to the SPARQL query processor. Other methods for managing the lifecycle of the resource are also implemented in this class.
- **QueryRepository:** this class is the class which queries the RDF repository. The RDFResource class passes the query to this class which is configured to query the repository.
- **RDFQueryActivity:** this class implements the server side activity in charge of activating the OGSA-DAI resource which accesses the RDF data.
- **RDFActivity:** The client side activity connects to the server side activity, requesting the execution of the RDF query. The activity gets as response a pointer where the results are being stored. Once the results are available it is possible to compose the output of the activity with other outputs, create workflows, etc.

The results returned by the RDFResource are in XML format for this specific case. It allows faster manipulation of the results and its transformation to OGSA-DAI tuples.

The RDFResource queries the RDF repository as follows:

- The client creates the activity that will access the data resource. This activity can be configured along with other activities accessing other data resources different from the ADMIRE Registry.
- The client activity submits the request of the execution of the SPARQL query to the server.
- The request is received by the server side activity.
- The server side activity sends the query to the RDFResource.

- 
- The RDFResource process the SPARQL query and accesses the ADMIRE Registry.
  - The results are returned to the OGSA-DAI server side activity which process the results of the query.
  - Once the results are processed (converted to OGSA-DAI tuples) they are returned to the client activity.

### C.3 Semantic Registry Client Interface

The semantic registry client interface implements the operations noted above. The implementation class is “SimpleRDFProcessingElementRegistry”. This class accesses specific classes which build the queries to the ADMIRE Registry.

- `ProcessingElementDescriptor lookupProcessingElement(String name)`: this operation receives a `String` which is the name of the PE we want to retrieve. The method “generateQuery” generates a SPARQL query which asks the registry for the PE with its inputs and outputs. It returns a `ProcessingElementDescriptor` which contains the results from the query.
- `List<ProcessingElementDescriptor> lookupProcessingElementList()`: this operation queries the ADMIRE registry for all the PEs. It returns a list of `ProcessingElementDescriptor` with all the PEs and their inputs and outputs.
- `List<String> lookupProcessingElementsName()`: this operation queries the registry for the names of all the PEs in the registry.
- `void registerProcessingElement(ProcessingElementDescriptor descriptor, String implementation)`: this operation updates the ADMIRE registry with a new PE. The functionality of attaching the implementation of the PE is still not implemented.
- `String lookupImplementation(String name)`: this operation queries the repository for an implementation of the given PE. It is still not implemented.