



ADMIRE D4.3 – Development Progress Report

Project Title	ADMIRE
Document Title	ADMIRE D4.3 – Development Progress Report
Deliverable Number	D4.3
Authorship	Vivian Lee, Alex Wöhrer, Radek Ostrowski, Carlos Buil, Dave Snelling
Document Filename	ADMIRE-D4.3-DevelopmentandProgressReport.docx
Document Version	1.0
Distribution Classification	Internal
Distribution List	<i>ADMIRE Project Team</i>
Approval List	<i>Marcin Choinski, Gagarine Yaikhom, Project Manager, Executive Board</i>

Document History

<i>Personnel</i>	<i>Date</i>	<i>Comment</i>	<i>Version</i>
V.Lee	27/08/2009	First draft	0.1
A. Wöhrer	02/09/2009	Input from Alex Wöhrer	0.2
R. Ostrowski	03/09/2009	Input from Radek Ostrowski	0.3
C.Buil	07/09/2009	Input from Carlos Buil	0.4
V.Lee	08/09/2009	Input from Vivian Lee	0.5
V.Lee	21/09/2009	Review feedback incorporated	0.6
R.Baxter	25/09/2009	Final edit and signoff	1.0

Contents

Contents..... 1

1 Executive Summary..... 3

2 WP 4: Summary: Project Month 18 4

2.1 Products delivered to other work packages..... 4

3 Progress against Planned Activities 5

3.1 Activity 4.1: OGSA-DAI Integration (FLE, UEDIN) 5

 3.1.1 Planned activity..... 5

 3.1.2 Actual activity 5

 3.1.3 Deviations from plan 5

3.2 Activity 4.2: Enhanced Monitoring (FLE, UVIE, UEDIN)..... 5

 3.2.1 Planned activity..... 5

 3.2.2 Actual activity 6

 3.2.3 Deviations from plan 6

3.3 Activity 4.3: Semantic Registry Integration (FLE, UPM)..... 6

 3.3.1 Planned activity..... 6

 3.3.2 Actual activity 6

 3.3.3 Deviations from plan 7

3.4 Activity 4.4: Semantic Provisioning Services Integration (FLE, UPM)..... 7

 3.4.1 Planned activity..... 7

 3.4.2 Actual activity 7

 3.4.3 Deviations from plan 8

3.5 Activity 4.7: General Enhancement to USMT (FLE, UVIE, UEDIN, UPM) 8

 3.5.1 Planned activity..... 8

 3.5.2 Actual activity 8

 3.5.3 Deviations from plan 10

3.6 Other Activities 10

4 Risks and Issues 11

 4.1 Project issues 11

 4.2 Significant problem reports 11

 4.3 New risks..... 11

5 Plans for next Period 11

 5.1 Activity 4.1: OGSA-DAI Integration (FLE, UEDIN) 11

 5.2 Activity 4.2: Enhanced Monitoring (FLE, UVIE)..... 11

 5.3 Activity 4.3: Semantic Registry Integration (FLE, UPM)..... 12

 5.4 Activity 4.4: Semantic Provisioning Services Integration (FLE, UPM)..... 12

- 5.5 Activity 4.5: WEEP Integration (FLE, UVIE) 12
- 5.6 Activity 4.6: OGSA-DQP Integration (FLE, UEDIN) 13
- 5.7 Activity 4.7: General Enhancement to USMT..... 13
- 5.8 Activity 4.8: Standards Alignment (FLE, UVIE, UEDIN, UPM)..... 13
- 6 References..... 14**
- A USMT Design and Implementation 15**
 - A.1 Resource Property 15
 - A.2 Resource Lifetime 16
 - A.2.1 Resource Creation 16
 - A.2.2 Resource Destruction 16
 - A.3 Notification 17
 - A.4 Subscriptions 17
- B Services Integrated with USMT Platform 18**
 - B.1 OGSA-DAI Services 18
 - B.2 Enhanced Monitoring..... 19
 - B.2.1 Design and Implementation of DSMON..... 19
 - B.2.2 DSMON Services 20
 - B.3 Semantic Registry Design and Implementation 21
 - B.3.1 Semantic Registry Design 21
 - B.3.2 Semantic Registry Implementation..... 22

1 Executive Summary

This document is ADMIRE Deliverable D4.3 and describes progress on Work Package 4 during months 13 to 18 of the ADMIRE project.

At project month 18 ADMIRE WP4 is essentially on target against original plans, having successfully deployed the V1.6.2 version of USMT. In terms of PM18's explicit goals for WP4, progress can be summarised as follows:

Implement basic USMT Gateway service: (carried over from PM12) this period saw the first successful deployment of an ADMIRE Gateway based on OGSA-DAI data services hosted within a USMT infrastructure.

Implement USMT presentation layer for OGSA-DAI: (carried over from PM12) integration of USMT and OGSA-DAI was completed with the porting and successful deployment of the two remaining services (DataSinkService and DataSourceService).

Add DMIL parsing service to Gateway: as part of the completion of the Gateway service the first version of the DMIL parser was successfully incorporated within the USMT infrastructure.

Develop first Registry: the first Semantic Registry has been developed and deployed successfully within the ADMIRE Testbed (running at a node at EPCC).

Extend Gateway with enhanced Monitoring service: the new enhanced resource monitoring service has also been successfully completed.

The integration of OGSA-DAI with USMT and the initial integration of Enhanced Monitoring and Semantic Registry provide the core of the PM18 ADMIRE Platform Release 2 (see [2]). Development on these three activities is still ongoing, but will be more focused on performance enhancement and modifications that adapt to the potential changes that maybe brought in by a new USMT platform.

The WEEP Integration activity was on hold, and will not be considered in the future. The reason is that the workflow capability the current OGSA-DAI platform provides is sufficient for the ADMIRE platform, and therefore there is no need to integrate WEEP further as originally intended. The effort planned for this activity will be spent on wider investigations in data management, which will bring in extra value to the ADMIRE project as a whole.

2 WP 4: Summary: Project Month 18

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, 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 13 to 18 of ADMIRE, WP4's main focus has been to deploy enhanced USMT to the project. One version of USMT has been distributed to the project, providing better stability and 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 and Semantic Registry services to USMT. All these integration activities have proceeded well, and versions of all these services were released as part of the ADMIRE Platform Release 2 in PM18.

More requirements on USMT have been gathered through these activities, and these will be investigated and implemented in the forthcoming 6 months.

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 v1.6.2 was released to ADMIRE on 27 May 2009.

Other releases from WP4 package work include:

- OGSADAI-USMT v0.9.0 on 23 June 2009;
- Database Monitoring Service (DSMON) v0.5 on 15 July 2009;
- ADMIRE Registry v0.1 on 14 September 2009.

3 Progress against Planned Activities

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

3.1.1 Planned activity

The OGSA-DAI [4] 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.1 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 last report on this work [12] noted that the basic functionality of all the six OGSA-DAI services had been ported to USMT. What significantly helped to complete this task was a different approach to the collaboration element between UEDIN and FLE. UEDIN provided a series of use cases which exhibited all the functionality required by OGSA-DAI. FLE provided solutions in an open source project that simulate the required environment. This also resulted in the documentation enhancement and code examples for other developers.

The process of unifying OGSA-DAI's resource ID based approach with USMT's EndpointReference (EPR) based approach was completed. New OGSA-DAI activities that work with remote data source and data sink services in an EPR fashion had been written.

The task of upgrading to USMT v1.6 (USMT v1.6.2) was also completed. This enabled the use of both dynamic and runtime static resource properties. This has also fixed the bug relating to the destruction of dynamically created resources.

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

3.1.3 Deviations from plan

As the current functionality of the OGSADAI-USMT is sufficient for the ADMIRE Gateway and other services, the work relating to implementing Exception and Error propagation has been pushed down the priority list.

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

3.2.1 Planned activity

USMT provides an infrastructure for passing information about changes in the properties of service instances. In Deliverable 4.1 [11], 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”.

Regarding monitoring, in this phase the activity focuses on the following two areas:

- data source monitoring via 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.
- progress monitoring via the enhancement of the OGSA-DAI workflow execution environment to allow progress monitoring of a running OGSA-DAI request.

3.2.2 Actual activity

Regarding the enhancement of progress monitoring, first improvements have been designed and implemented in the OGSA-DAI services. These extensions were implemented to allow progress monitoring of a running OGSA-DAI request. The monitoring framework can be added to an OGSA-DAI service and tracks the number of blocks that are read from and written to each activity input and output. The current status report is available as a resource property on the request resource. A client can request this resource property to monitor progress of the request. Note that it is not possible to calculate the percentage of processed data because the OGSA-DAI framework streams data blocks and therefore the total number of data blocks is unknown until processing has completed.

Regarding the enhancement of data source monitoring, a first USMT-enabled version of a data source monitoring tool was implemented/migrated based on UVIE's earlier work described in [16]. Appendix B.2 gives details of the design and implementation of the data source monitoring service DSMON.

3.2.3 Deviations from plan

No deviations from the plan to report.

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 [14], the semantic registries of the NeOn Toolkit [15] or the Metadata Registry of the Semantic Web Framework [13].

In this period an initial version of the ADMIRE Registry has been implemented. This version is integrated within the ADMIRE Gateway and the DMIL Processor. A more detailed description of the design and implementation of the registry can be found in Appendix B.3 of this document.

3.3.2 Actual activity

The Registry is composed by an OGSA-DAI activity, which is queryable by using the SPARQL language. 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 this restriction into account. For fulfilling it, a new OGSA-DAI resource plus activities for accessing it have been implemented. This new resource provides access to an RDF repository which contains references to existing ADMIRE Processing Elements.

The ADMIRE Registry provides access to the Processing Elements designed by the ADMIRE end users. It is a semantic registry in nature based on the WS-DAI-RDF specification. Primary users of the ADMIRE Registry are the Process Designer, the SKSA and the ADMIRE Gateway.

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).

3.3.3 Deviations from plan

No major deviations from plan. The Registry has been developed as an OGSA-DAI activity and resource for accessing RDF repositories. The initial plan was to use the OGSA-DAI-RDF¹ implementation of the WS-DAI-RDF specification, but we decided to implement an internal version of this specification, to avoid using redundant packages.

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) (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) [10], 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.

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 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 and D5.2 [10]. This activity heavily relies 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

¹ <http://wiki.dbgrid.org/index.php?OGSA-DAI-RDF>

organization in charge of the integration of them with other components active tracking of this activity will be straightforward.

3.4.3 Deviations from plan

The development of the SDDA plug-in for the Process Designer started in month 18. The other semantic provisioning services (the ontologies and the registry) have already finished their first version. Therefore once the SDDA is finished the integration will start. Up until now no deviations have been identified.

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

3.5.1 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 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.

3.5.2 Actual activity

In the third period of the project, the development mainly focused on improving USMT to accommodate requirements from the ADMIRE platform. At the second stage, most of the requirements arose from the OGSA-DAI/USMT integration activity, requiring enhancement on basic service management functions as service instance instantiation, lifetime management, and richer functionalities on resource properties; these requirements have already been addressed by USMT release 1.6.2. At the third stage, communications between different service components have become increasingly important to the ADMIRE platform as a whole. For instance, a Gateway service developed by WP5 need to be able to discover hence connecting to OGSADAI-USMT developed by WP4, in order to submit a DMIL task for the OGSADAI backend for workflow processing. Therefore, service instance discovery in a distributed environment is a major challenge.

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

Static EndpointReference for Singleton Service deployment

USMT employs the IP multicast technique to handle service instance discovery in general, e.g. when a new Nucleus is up and running, it sends a multicast message to the network, announcing its presence. Meanwhile, it also starts to listen to other multicast messages sent by other Nuclei. Once they have discovered each other, information about current running service instances can be detected by sending a query on the KnowServiceInstance resource property. However, due to the fact that multicast typically only works within the same subnet, not to mention problems with services that are running behind a firewall, service discovery in a distributed environment that crosses network boundaries is impossible to achieve. Many solutions to this problem have been considered, including a global Registry service, and a Proxy service sits at each network router. A final implementation decision has been made to choose the static EndpointReference approach, based on practicality and time constraints.

The normal service instantiation procedure is that the hosting environment creates an instance of the service type, with a randomly generated ReferenceParameter as a unique ID to the EndpointReference. The static EndpointReference approach provides a flexibility that allows a singleton service instance be created with a default ID specified by the developer. This default ID, which can be set by the `services.xml` file at configuration time, will be picked up by the USMT hosting environment at the bootstrapping stage, and will be used as the value of ReferenceParameter. In addition, since other elements in an EPR are relatively stable, e.g. once it has been decided which machine will run the service, the Addressing field of an EPR is relatively static, therefore, the whole EPR can be made static. The static EPR can be advertised to other components by email, and will be stable for a certain period of time regardless the number of times the Nucleus service is restarted.

Notification Filter

To reduce notification message traffic overhead, and hence improve the performance of USMT in the ADMIRE platform, a *MessageContent* filter has been implemented in the subscription request. This filter allows clients to specify criteria as to which notifications they want to receive when the server side resource property changes. For example, a client would like to be notified when the status of an OGSA-DAI workflow execution is complete. Without a *MessageContent* filter, the client can only subscribe to changes to the status value, which can range from Non-Started, to Started, Processing, and Complete. In this case, the *MessageContent* filter helps to reduce the traffic of unwanted notifications and only sends when the status is Complete. The format of *MessageContent* is an XPath string.

Nuclei Discovery

A USMTNucleiDiscoveryUtil class was developed to further help USMT application developers to discover service instance within a multicast range. The class allows one to discover a Nucleus by certain criteria, e.g. host-name IP address, serviceTypes that are running on the Nucleus, so that only desired Nuclei are returned.

Destroy operation fails if there are more than one nucleus exists in the current environment

A bug has been identified through the OGSA-DAI/USMT development, such that if there is more than one nucleus present in an environment, any execution of the Destroy operation on any of the service instances that are running on any of the Nuclei throws a NullPointerException occurs, which prevents further execution on the operation.

The actual cause was that when a remote Nucleus registers itself with a local one, it only registers its portType and EPR, but not the instance. When the destroy operation is executing, the instance assignment to a local variable fails since there is no instance stored from the previous register method. The fix to this is just simply disallow the destroy action on a remote Nucleus, for security and

authorization reasons. However, a solution that covers Nucleus creation, registration, remote access, operations on the remote nucleus as a whole, needs to be carefully architected at a later stage.

Client does not get notified on runtime resource property value change

If a subscription is taken out on a resource property that is created at *runtime* instead of *compile time*, the notification infrastructure does not work properly. This is due to the fact that the `TopicExpressionType` of the runtime RP is not added into the notifiable list. The fix to this bug provides a flexible solution that allows update on the list either in the `onCreate` method, or any of the application method.

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.

Gateway USMT integration

Apart from the above activities that enhances USMT to achieve better performance for ADMIRE platform, WP4 also provided substantial supports in helping WP5 to build the Gateway service on USMT.

3.5.3 Deviations from plan

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

3.6 Other Activities

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

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

4 Risks and Issues

4.1 Project issues

The WP4 development over the third 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 M17-19 and M19-22:

- M17-19: Include the Enactment Engine platform and capabilities.
- M19-22: Enhance related lifecycle management capabilities supporting enactment engine functions.
- M22-24: Third round of deploying DMI platform services to the USMT-V3.

The updated deliverable description is:

Development Deployment Report for USMT V3: capabilities for USMT V3

This report will be an update of deliverable D4.3 as it is expected that the work on the WEEP integration with the ADMIRE platform will continue to be the main focus of the ISB enhancements in this period

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.

The first version of the ADMIRE Registry is completed. The integration within other ADMIRE components that will use the Registry has also started. Initially the Registry has to be integrated within two other components in ADMIRE:

- The first component to integrate the Registry with has been the DMIL Processor and the ADMIRE Gateway. The integration within these two ADMIRE components has been based in the Java interfaces created for the former. These interfaces described a simple access to a local registry. We have implemented the interfaces needed for accessing the RDF store which contains the information about the PEs. The implementation of these interfaces has been done using OGSA-DAI activities to access the Registry. These activities allow querying a RDF repository (the Registry itself) using SPARQL.
- The second ADMIRE component to be integrated within the registry is the Process Designer. The process of integration within this component relies on one of the plug-ins attached to it: the Semantic Data Description Assistant (SDDA). The implementation of this plug-in will start in PM 18 and a follow up of the development of it will be carried out in order to produce a correct integration within the registry.

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

Semantic Provisioning Services are mainly encapsulated within the SDDA. The SDDA will be integrated with the Process Designer and the ADMIRE Registry. Its implementation started in PM 18 and the integration of it with the other ADMIRE components play a key role in its design. A close following of the development of the SDDA is being carried out.

The RDF repository selected has been Jena. Jena offers the most appropriate functions to access first both local and remote RDF repositories and second it provides means for managing OWL ontologies. It offers an RDF storage mode in SQL databases (MySQL, HSQLDB, PostgreSQL, Oracle and Microsoft SQL Server) as well as an OWL API for accessing the ADMIRE registry model.

5.5 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.

We have found that this planned activity will now no longer be necessary, the reason being that the workflow enactment capability provided by the recent versions of the OGSA-DAI platform is sufficient enough to fulfil the requirements of highly dynamic and interactive workflows. Instead, the effort will be spent in looking more widely into potentially useful data management technologies, including streaming data, Hadoop, map-reduce, etc., which potentially will bring in extra value to the ADMIRE project.

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.*

No work planned for this period.

5.7 Activity 4.7: General Enhancement to USMT

See Section 3.5.1 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 integration between different components and the 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;
- Notification PullPoint.

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. At the very last month of the project, we will consider possibility to migrate USMT to adopt the new standards developed by WSRA working group, and look into potential impact this might have on workflow management.

6 References

- [1] The ADMIRE Consortium. ADMIRE: Description of Work , Feb 2008.
- [2] The ADMIRE Project. ADMIRE Platform Release 2, Sep 2009.
- [3] ADMIRE Risks and Issues Log, July 2008, <http://www.admire-project.eu/trac/browser/projman/plans/ADMIRE-risksIssues.doc>
- [4] The OGSA-DAI Project <http://www.ogsadai.org.uk/>
- [5] OGSA-DAI Component Registry: <http://www.ogsadai.org.uk/contribs/registry.php>
- [6] JAX-WS reference Implementation: <https://jax-ws.dev.java.net/>
- [7] Apache Ant: <http://ant.apache.org/>
- [8] WS-Naming: <http://schemas.ogf.org/naming/2006/08/naming/>
- [9] 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.
- [10] 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.
- [11] 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.
- [12] 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.
- [13] R. García-Castro and M.C. Suárez-Figueroa (eds), “D 1.2.4 Architecture of the Semantic Web Framework”, Knowledge Web Deliverable, Feb 2007.
- [14] 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.
- [15] 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
- [16] A. Wöhrer, L. Novakova, P. Brezany and A. Tjoa, “Grid-aware approach to data statistics, data understanding and data preprocessing”, Int. J. High Performance Computing and Networking, Vol. 6, No. 1, 2009
- [17] A. Wöhrer and P. Brezany, “Towards a Novel Metadata Information Service for Distributed Data Management”, Journal of Networks, vol. 2, no. 6, December 2007

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 USMT Platform

B.1 OGSA-DAI Services

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 a 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).

B.2 Enhanced Monitoring

B.2.1 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 [15]:

- 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 cumbering 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.
 - Loosly 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, PostGRE 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.2.2 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

DSMONTTable 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].

B.3 Semantic Registry Design and Implementation

B.3.1 Semantic Registry Design

The ADMIRE registry is a component that provides access to the Processing Elements developed within the project. The references to these PEs are stored in a RDF file and a SPARQL based access is provided to them.

Two of the features of representing the PE as RDF data are the integration with the different ADMIRE ontologies and the possibility to query using SPARQL the registry. The first feature allows the ADMIRE Process Designer to annotate the PEs created in the ADMIRE Process Ontology. The instances of this ontology are the elements stored in the ADMIRE registry in RDF. The users can benefit from the usage of the reasoning mechanisms provided by the ontology modelled in OWL. The second key feature is the ability of querying the repository using SPARQL taking advantage of the relations represented in the graph like data repository.

The elements are stored in RDF which is a language for representing data. Therefore the services for accessing the RDF store implement part of the WS-DAI-RDF² specification. OGSA-DAI implements part of the WS-DAI specification and we added a new data resource to OGSA-DAI which is in charge of accessing the RDF repository containing the PEs references. The extension to OGSA-DAI is shown in Figure 1.

The extension to OGSA-DAI with the new data resource follows the same pattern as the other OGSA-DAI resources for accessing data sources. The new resource implements the DataResource interface which is the primary class for accessing data resources. It implements activities for managing different queries that can return several values if they are configured in this way. And the results of the resource are formatted into OGSA-DAI tuples so the results of the SPARQL query can be integrated with other data query activities.

² <https://forge.gridforum.org/projects/dais-wg>

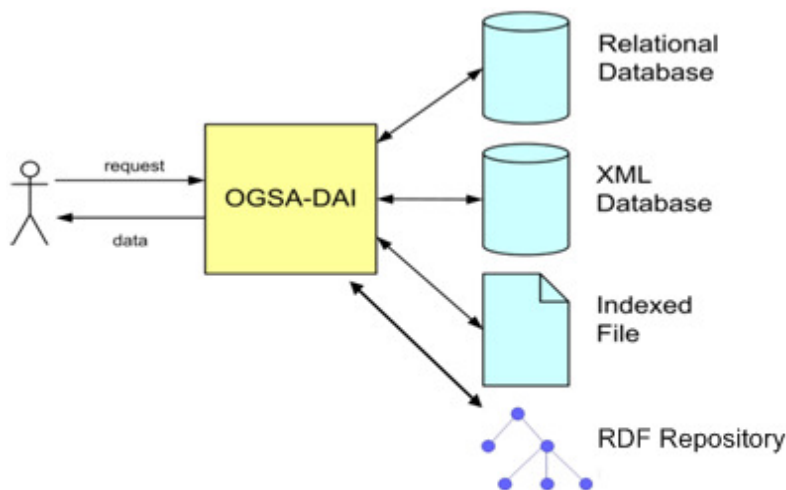


Figure 1 : RDF Resource Extension

B.3.2 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*, in charge of sending the SPARQL 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 a faster manipulation of the results and its transformation to OGSA-DAI tuples.

The way of working of the RDFResource for querying the RDF repository is the following:

- 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 accessing the configure 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