A Semantic Data Grid for Satellite Mission Quality Analysis

Reuben Wright[1], Manuel Sánchez-Gestido[1], Asunción Gómez-Pérez[2], María S. Pérez-Hernández[3], Rafael González-Cabero[2], and Oscar Corcho[2]

[1] Deimos Space, Spain
{reuben.wright,manuel.sanchez}@deimos-space.com
[2] Departamento de Inteligencia Artificial. Facultad de Informática, UPM, Spain
asun@fi.upm.es, rgonza@delicias.dia.fi.upm.es, ocorcho@fi.upm.es
[3] Departamento de Arquitectura y Tecnología de Sistemas Informáticos. Facultad de Informática, UPM, Spain
mperez@fi.upm.es

Abstract. The combination of Semantic Web and Grid technologies and architectures eases the development of applications that share heterogeneous resources (data and computing elements) that belong to several organisations. The Aerospace domain has an extensive and heterogeneous network of facilities and institutions, with a strong need to share both data and computational resources for complex processing tasks. One such task is monitoring and data analysis for Satellite Missions. This paper presents a Semantic Data Grid for satellite missions, where flexibility, scalability, interoperability, extensibility and efficient development have been considered the key issues to be addressed.

1 Introduction

Earth Observation is the science of getting data about our planet by placing in orbit a Hardware/Software element with several observation instruments, whose main goal is to obtain measurements from the Earth surface or the atmosphere. The instruments on board the satellite act like cameras that can be programmed to take images of specific parts of the Earth at predefined times. This data is sent to Ground Stations and then processed in order to get meaningful scientific information.

Parameters for instrument operations and for the satellite configuration constitute the Mission Plans issued by the Mission Planning System. These plans are issued regularly (e.g., on a weekly basis), and can be modified until they are sent to the satellite. Catastrophic events such as earthquakes, volcanic eruptions, and hurricanes are examples of events that can cause last minute re-planning. These plans and their modifications are sent to the Flight Operation Segment (FOS), which in turn resends that information to a Ground Station and from there to the satellite antenna of the spacecraft. A computer on board the satellite stores the list of MCMD (MacroCommands) that request an instrument or any other part of the satellite to perform an action. These include loading a table, triggering an operation and getting internal status

A. Sheth et al. (Eds.): ISWC 2008, LNCS 5318, pp. 818–832, 2008.

information. Images from each of the instruments are stored onboard (in the satellite computer memory) as raw data and when the satellite over-flies the Ground station that data is sent to the Ground Station antenna (Data downlink). Conversion from the raw data to higher level "products" (adding identification labels, geo-location data, etc.) is performed sequentially at the Ground Station and various Payload Data Segment facilities. Fig. 1 shows the overall scenario. A more detailed explanation of the whole system can be found in [1].

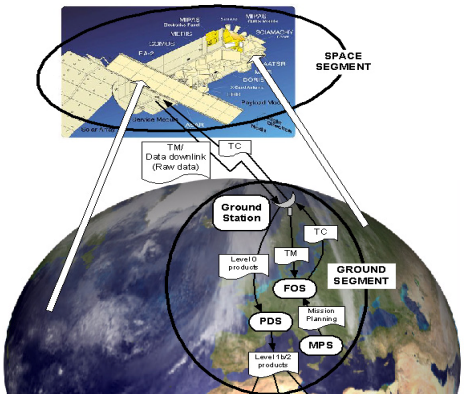

Fig. 1. General overview of an Earth Observation Satellite system (Envisat)

Among the currently active Earth Observation Satellites we find **Envisat**, which monitors the evolution of environmental and climatic changes, and whose data facilitates the development of operational and commercial applications. The satellite carries 10 different instruments and is extensively described in [2]. The work presented in this paper is focused on giving support to this system.

Data circulates within the system as various Plan, MacroCommand and Product Files, with well-defined structures. There can be a variety of hardware or software problems that can occur within the process, hence there is a need for the system to be monitored. **QUARC** is a system that checks off-line the overall data circulation process and in particular the quality of the instrument product files. It checks that the satellite and instrument have performed successfully the measurements (taking images of the Earth), that these images have been stored onboard and transmitted as Raw Data to the Ground station and then processed correctly. QUARC returns reports and plots, which help in the production of new plans. Additionally, the QUARC system is designed to assist in decision making when an instrument or the whole system malfunctions and to detect, in a semi-automated fashion, that something incorrect has occurred in one part of the product generation or data circulation.

The operational QUARC system is located in a single location (ESA-ESRIN, in Italy), which communicates with the archive containing all the products generated from the beginning of the mission and with all the other facilities. The Data Ingestion Modules, one per file type, read the files and convert their contents into parameters that are meaningful to the QUARC data model. The system has been specifically built

for this purpose and has bespoke user interfaces. It took several years to build it and there are significant maintenance and development costs as new reports are required and new missions are launched.

Our objective is to replicate some of the features of the QUARC system, namely the comparison between the planned activity and the production of data by the satellite and the further processing of that data in ground systems, demonstrating that we can achieve greater degrees of flexibility, scalability, interoperability and extensibility than the existing system, together with a more efficient development of new functionalities. The existing QUARC system stores implicit metadata about the files that it manages (e.g., hidden in file names) and exposes this metadata through bespoke interfaces. Our approach consists in extracting this metadata to build an explicit semantic representation of the information that is managed, and store it in a way that exposes flexible query interfaces to users and where data is distributed.

In the rest of the paper we look at some detailed use cases for the system, then consider the technical approach and implementation. Finally we summarise the key advantages of the semantic grid approach taken and consider the next steps to be taken in uptake of this approach.

2 Advanced Requirements for Quality Analysis

In addition to functional and non-functional requirements from the existing system we produced the following Use Cases to support incremental, distributed development. These translated directly into Test Cases for evaluation of the system.

Use Case 1: Instrument unavailability. This is a Use Case to ensure our new system is capable of replicating the core functionalities of the existing system. A user needs to find out what planned events and generated products exist for a given time period and instrument, and to plot these results against each other in a timeline. A simple interface is needed, with no underlying complexity exposed to the user.

Use Case 2: Check for the quality of the products in Nominal mode. Certain sorts of products have internal parameters giving a measure of quality of data. The specific situation for this use case at present would be the extraction of one of these quality parameters, over a period of time, for an instrument in a particular mode, being "Nominal". The product files we were to work with did not include this quality data so we expanded this to the more general requirement to be able to extract any piece of metadata from a set of product files. Extracting a new parameter from a file is quite simple for an experienced QUARC user, but it is time consuming and error prone.

Use Case 3: Update of Functionalities with no Software Update. A crucial perceived advantage of the semantic approach was the flexibility with which the system could be adapted. A mission may last 10-15 years and since we are largely investigating anomalous behaviour not all useful queries will be known ahead of time. We needed to know how easily we could develop new queries over our data, and parameterise them for use by ordinary users. This is a complicated process with the current system.

Use Case 4: Data Lifecycle. Satellite plans are not static and the system needed to be able to remove or update metadata from its stores. This needed to be done automatically, and only in the correct circumstances of a new plan covering the same time period and from the provider of the original plan. When querying, the user must be given information about the final, executed, plan. This is managed within an existing QUARC system by virtue of it being centralised, with a single source of data ingestion. In a networked, distributed environment there would be no such facility.

Use Case 5: Modularity of Metadata Service. The desire to be able to change the metadata store comes from wanting flexibility in extending the system. The approach was to design and build a loosely-coupled, service-oriented architecture. In particular we would ensure we could change the metadata store and query engine, but more generally we would use modular components defined by their interfaces. Choices between components can be made on various characteristics including cost, scalability, reliability, and performance. Crucially the user shouldn't have to worry about implementation details. There is no equivalent to this in the existing QUARC system, it is tied to specific versions of the underlying relational database.

3 A Semantically and Grid-Enabled QUARC System

In this section we describe the approach taken for the design and implementation of a semantically and Grid-enabled version of the QUARC system. We start describing the architecture in which the development is founded, and then move to the other ingredients of the development, namely annotation, storage and querying.

3.1 An Architecture for a Semantic Data Grid

We have used the S-OGSA architecture [3] for our development. S-OGSA extends the OGSA model [4], which is commonplace in Grid middleware and applications, and includes two service categories called Semantic Provisioning Services and Semantically Aware Grid Services, as described below.

S-OGSA Information Model. The S-OGSA model identifies three types of entities:

- *Grid Entities* - anything that carries an identity on the Grid, including resources and services [5]. In this system they include planning systems, planning files, satellite instruments, product files and product processing facilities.
- *Knowledge Entities* - Grid Entities that represent or could operate with some form of knowledge. Examples of Knowledge Entities are ontologies, rules, knowledge bases or even free text descriptions that encapsulate knowledge that can be shared. In this system we had a single ontology including classes for times, planning systems, macrocommands, satellite instruments, and the details of the various plan and product file metadata. Ultimately there needs to be a set of ontologies to cover the whole satellite mission domain. In the Satellite Mission Grid an annotation process creates knowledge entities (sets of RDF statements) for the different types of files.

- *Semantic Bindings* - Knowledge Entities that represent the association of a Grid Entity with one or more Knowledge Entities (that is, they represent semantic metadata of a Grid Entity). Existence of such an association transforms the subject Grid entity into a Semantic Grid Entity. Semantic Bindings are first class citizens as they are modeled as Grid resources with an identity and manageability features as well as their own metadata. Grid Entities can acquire and discard associations with knowledge entities through their lifetime. In our system the files are made into Semantic Grid Entities by attaching the created annotations.

Semantic Provisioning Services. These are Grid Services that provision semantic entities. Two major classes of services are identified:

- *Knowledge provisioning services*. They manage Knowledge Entities. Examples of these services are ontology and reasoning services. Ontology services are implemented using the RDF(S) Grid Access Bridge, an implementation of WS-DAIOnt [6], under standardisation at OGF.
- *Semantic Binding provisioning services.* They produce and manage Semantic Bindings. They include annotation services that generate Semantic Bindings from planning and product files, implemented with Grid-KP [7]. They also include a semantic binding storage and querying service [8], which is implemented twice, using the Atlas distributed RDF(S) storage system [9] and Sesame [10].

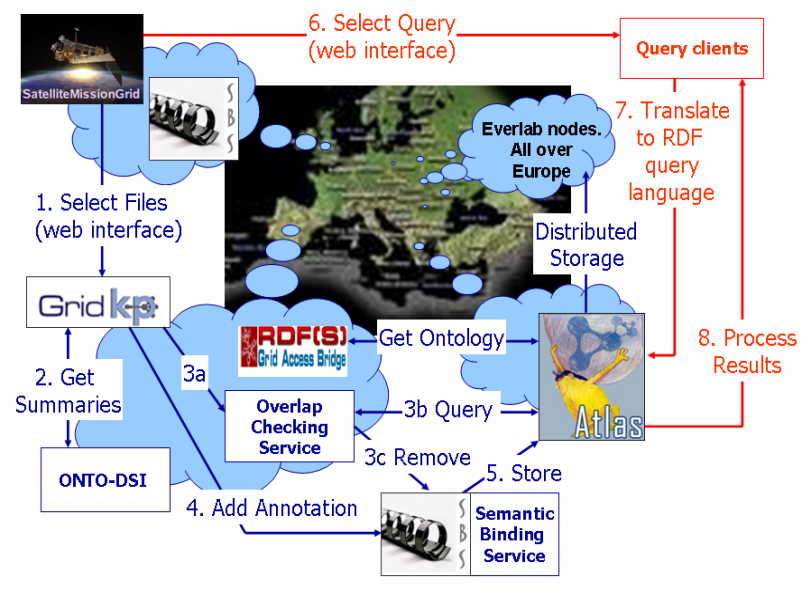

Fig. 2. System architecture

Semantically Aware Grid Services. They are able to exploit semantic technologies to consume Semantic Bindings in order to deliver their functionality. They consume the semantic entities held by Knowledge and Semantic Binding provisioning services,

and use their services. The user interface for the Satellite Mission Grid is a Semantically Aware Grid Service, making use of all the aforementioned elements in order to deliver its enhanced functionality.

Figure 2 shows the geographical deployment of the developed system. Software was deployed at 3 sites – Manchester, Madrid and Athens, and Atlas further uses the Everlab cluster of machines throughout Europe. The numbered actions 1-5 and 6-8 show the activity flow for annotating and querying data respectively.

3.2 Annotation: Making Metadata Explicit

Data circulates in the existing systems as files with many common generic features. They are slightly different for planning and product files, and the information about these planned events and generated products is usually bound up with the data involved. Standard ASCII formats encode the information in keyword-value pairs, which are stored as headers for the various files. This is a special format defined for the Envisat mission with an enormous amount of software and documentation generated through years of development. This structure can be simply translated to a fairly flat XML structure. Once this is performed on the planning and product files, the system uses XML software tools.

Product files consist of an ASCII header and a binary part encoded in an ESA proprietary format. The header is just a few Kbs out of an image file size of Gbs. The Onto-DSI [11] component was used to extract and provide just the headers from these files to avoid a large system overhead whilst annotating them.

Much of the metadata is encoded in specific, amalgamated identifiers, with “implicit semantics”. For example rules had to be created for product filenames like "RA2_MW__1PNPDK20060201_120535_000000062044_00424_20518_0349.N1". This is decomposed into an Event type (RA2_MW), Processing level (1P) and centre (PDK), a Sensing start time (2006-02-01:12.05.33) and so on. Generic metadata (applied across all captured metadata) and the ontology further add, for example, that the Event type (RA2_MW) is executed by a particular instrument, the Radar Altimeter. A parser extension to Grid-KP carries out the extraction of the relevant file properties.

Another issue was conversion of units. One example of this was converting from date formats, as given above (and given to the users in the webforms) to another standard time format used in space missions, MJD2000. It is the number of seconds (and milliseconds) to have passed since the year 2000, including leap seconds. The conversion routine was wrapped as a Web service using SOAPLAB [12].

Migration of other data to the system would be much simplified by this process and these tools being in place. In addition, the annotation services were deployed in different locations, which supported the distributed nature of the data sources.

3.3 Storage: Managing a (Meta)Data Lifecycle

The Annotation Service was able to use exactly the same mechanisms as the user interface to communicate with the Semantic Binding Service to ask if its current file overlapped with any existing Plan files. This design has two advantages; firstly, no new specific code needed to be written as we already had the query interfaces. Secondly, although the logic needed here was quite simple, we have allowed ourselves

full access to the flexibility of RDF querying, which means that if more complex rules are needed in future we will be able to accurately encode them. RDF can be updated using the standard mechanisms provided by metadata stores.

Managing RDF in identifiable, separate Semantic Bindings allows us to better manage the overlaps, and the lifetime of the metadata when several annotations may be created.

3.4 Querying: Exploring Data

We worked with a flexible "Free Querying" interface as we considered how the system would be incrementally improved and developed. This interface simply allowed the user to create queries (in the language of their choice: SPARQL, SeRQL or RQL) and get the results back in a tabular form.

As an example we looked at how we could abstract our queries over the data to a level where new sorts of data could be added to the system and still be identified by our queries. An initial implementation of one of the queries for Use Case 1 was looking for all planned events (DMOP event records) which were using the Radar Altimeter. We matched at the low level of Event identifiers using the implicit metadata that "events with identifiers containing RA are carried out by the Radar Altimeter instrument". The nature of RDF as a web of statements and the existence of an ontology to formalise the existence of different properties made it easy to move these queries to an improved, semantic level.

We were initially searching on the event_id property of the DMOP_er class (DMOP event records), which look like `"RA2_IE_00000000002372"`. It matches `REGEX(?EVENT_ID,".*RA.*")` in the SPARQL regular expression syntax. This query works, but we were able to see in the ontology that a better level was possible.

The individual data items about planned events use the event ids, but our system was able to augment that with the knowledge about which event types use which instruments. This was enabled by having an ontology which included instruments and types of events as objects independent of the individual events which they classify. Figure 3, showing part of the Satellite Ontology, specifies that the `DMOP_er` class (top left) is related to the `Plan_Event` class by the `represents_plan_event` property, and that `Plan_Event` instances have their own identifiers – `plan_event_id`. They represent the different types of events that can be planned. The next level of abstraction is the `Instrument` class and the Radar Altimeter is identified as one such task, with `instrument_id` of `"RA"`.

We translated the WHERE clause of our SPARQL query from

```
?EVENT event_id ?EVENT_ID ;
FILTER ( REGEX(?EVENT_ID,".*RA.*"))
```

to

```
?EVENT event_id ?EVENT_ID ;
       represents_plan_event ?PLAN_EVENT_TYPE .
?PLAN_EVENT_TYPE executed_by_instrument ?INSTRUMENT .
?INSTRUMENT instrument_id "RA"
```

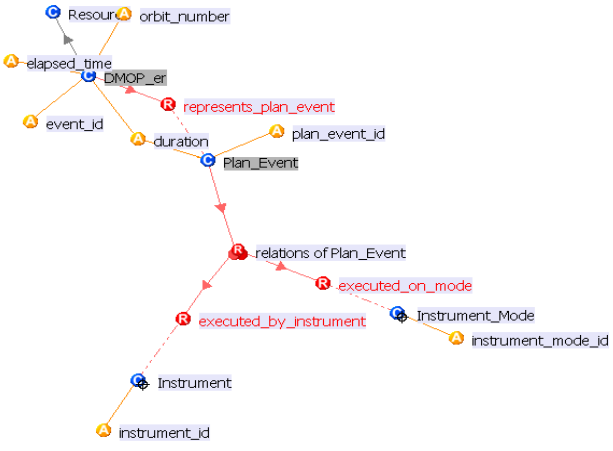

Fig. 3. A section of the Satellite Ontology showing the specific events (DMOP-er), their types (Plan_Event) and the Instruments which carry them out. Diagram from NeOn Toolkit [13]

While this is longer it is both clearer to understand and to implement as a webform where the user will select an instrument. It is also likely to execute more quickly as it is looking for an exact match of `instrument_id` rather than having to rely on regular expression parsing of a string value.

The biggest gain is that it is much more robust in the face of changing data. We can continue to use these "semantic level" queries about instruments even if we add new event types which use this instrument or change the unique identifiers for individual DMOP event records. If further data in the system contained new sorts of events planned and carried out by the Radar Altimeter then our queries would automatically match them. In any of these extended cases a simple statement associates the new event type with an existing instrument or new events with an existing event type. The exact same query (for use of an instrument) will then also report about these new events. We shifted from talking about details of identifiers to the actual objects which the user is concerned about, i.e. we moved to a more semantic level. This process is shown in more detail in an OntoGrid demonstration video [14].

3.5 Accessibility: Providing Tools for End Users

Having developed the queries in a language such as SPARQL we very easily built webforms to allow users to provide the values for any such queries. A web application (running on Tomcat) was developed to serve Java Server Pages which presented interfaces and results to users (see Figure 4), and converted to queries according to RDF data. The results were provided in a simple XML structure and we generated from that either tables or graphs.

The flexibility of the data model, and its level of abstraction from the interface also allowed us to look at different ways of accessing it. A separate tool was developed later to allow the programmatic augmentation of data in the database. As well as

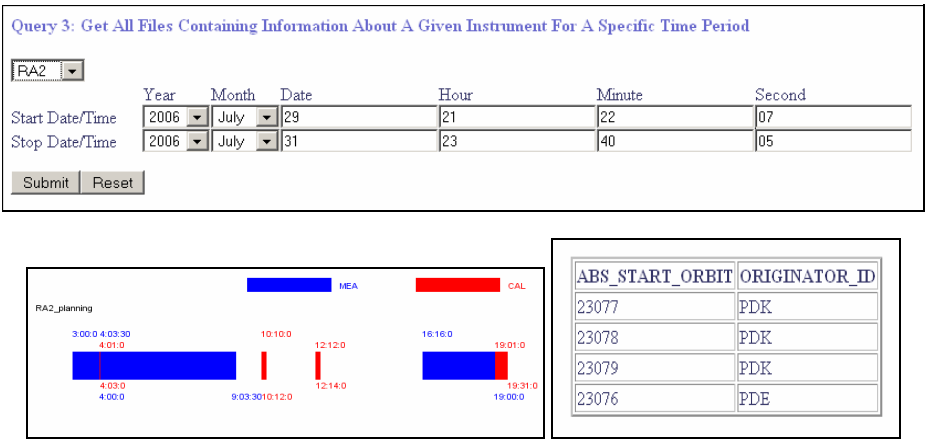

Fig. 4. Webform for user input and timeline and tabular outputs

allowing the adaption of data, for example adding in UTC datetime values as well as the MJD2000 ones supplied in the data, this can help with organising the data. For example, a set of subtypes of instrument types could be created to describe power consumption. Then each of the instruments could be labeled as belonging to one of the subtypes. This would take just a few statements in RDF and the millions of pieces of event information organised by which instrument generates them would now also be connected to this set of subtypes, allowing a user to make queries based on the power consumption.

A webform lets a user define a SPARQL query to identify various RDF nodes, i.e. bound variables in query results. The user can then choose one of these variables as the subject of a new property, and can construct a formula (using another of the variables, and the subset of XPATH expression syntax which does not involve nodes and sequences) for calculating the value of the new property. They also select a name, namespace, and type for the property. For every result the query returns, the system will then calculate the value for the new property and add it to the RDF store.

It does not add the results directly, but to a protected user area (“sandbox”). The user can query, examine or export the contents of their Sandbox to check that the calculated properties are as they expected. They can then choose to add the data to the main RDF store, where it will be available for all queries. This is managed technically with the Sesame concept of “context” for any piece of information. They are put into a user context initially, and can then be moved to the main “core” context.

Another tool allowed the simple creation of reports and graphs (see Figure 5). The idea here was that a web application for report designers could produce from their input a webform which would be used by normal users. These user webforms have public, visible selection fields for the user to use, but also hidden fields detailing the base SPARQL query the designer wants to run and the sort of graphical output they want. The report processor combines the base query with the additional parameter

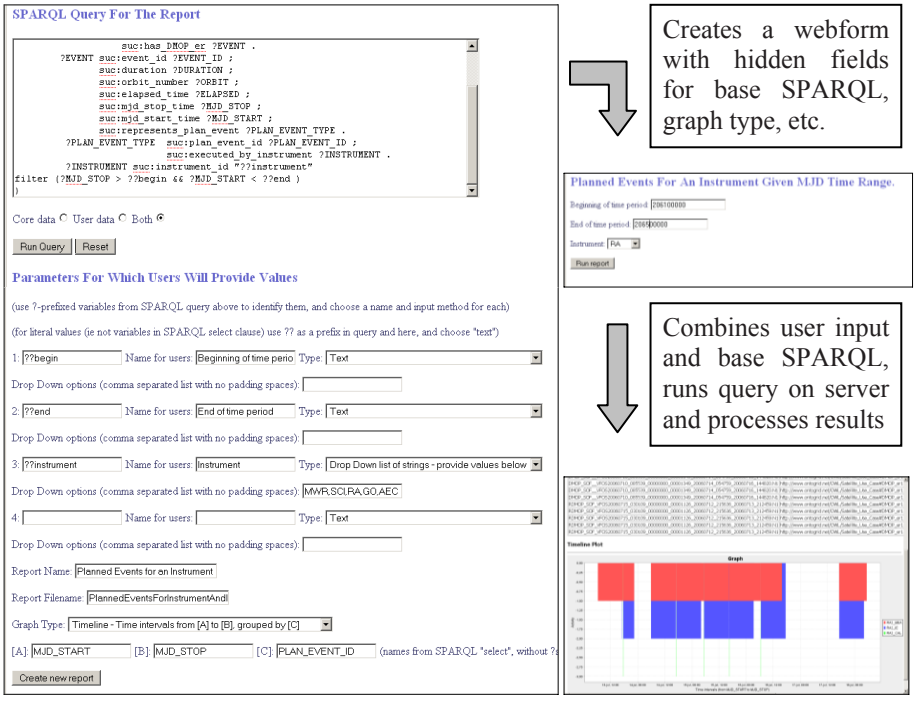

Fig. 5. Process of creating and using one of these auto-generated reports

values provided by the user (e.g. a start and stop date, or instrument type) to create a SPARQL query to run. The user parameters can be used for either specifying bound variables or literal values. The results are then processed according to the report designer's instructions and presented back to the user in the browser.

The user interface was developed using Java Server Pages, Java, XSLT and SPARQL. The Sesame 2 data store and interface were deployed on Apache Tomcat.

A more sophisticated design would combine these two pieces, utilising the idea of processing pipelines. These enable processing to be built up from simple components run one after the other. For example one component could add some data to the "active" dataset, another would run the query, a third would process results to generate a graph, a fourth would generate a summary table, a fifth would generate the results tables and a final stage would collate the outputs of the previous 3 to create the final report as presented to the user.

4 A Grid-Enabled QUARC System

The QUARC system involves a complex process in which distributed data belonging to different organizations must be queried, processed and transferred. The previous version had some clearly defined limitations, namely:

- Data Transfer: It is necessary the transfer of large files among different organization. This was made in a traditional style, through different FTP servers and ``ad hoc" solutions. Furthermore, the volume of data that is transferred may be decreased, since only a small part of the files is significant for the correct performance of many of the functionalities.
- Security: There was no definition of sophisticated access control mechanisms neither virtual organizations.
- Scalability: In order to deal with a huge number of files, the scalability issue must be addressed.
- Previously the location of the resources and the processing of data were made in a *wired* way, according to filenames and content of these files in an “ad-hoc” format.

All of these limitations have been relieved by means of the intensification of the “griddy" features of the QUARC system. Indeed, the use of a grid framework provides a flexible way of locating required resources and the virtualization of these resources by means of (Semantic) Grid Services.

Several services and functionalities have been developed for tackling these aspects:

- For improving data transfer, we have used GridFTP[15]. We have developed a new Data Storage Interface (DSI) to GridFTP, suitable to deal with Satellite files.
- Regarding security, the involvement of several organizations implies the establishment of different access policies and the definition of virtual organizations. The role of each specific actor within an organization also defines its privileges as a member of the virtual organization. Globus [16] allows us to establish security and define different policies.
- Our system enhances the performance of the data transfer, transferring only the necessary part of the information. This provides a higher scalability.
- Grid provides capabilities for locating services according to metadata.

5 Validation of the System

The implementation shows several advantages compared to a conventional approach, in terms of flexibility, reduction of software running costs, maintainability, expandability, and interoperability.

Legacy Formats and Systems. One of the key issues to bear in mind when implementing a completely new design in a system of this size and complexity is how to manage migration of systems. Some parts of the system (which may belong to other organisations or facilities) are not changed at all, or only partially updated, and there is no simultaneous modernisation of all parts in the system. Legacy systems can be considered in terms of data formats and software.

- **Data Formats.** Although current data files are structured and in well documented formats there remain hundreds of different file types. Much of the metadata is only implicit, such as the information stored within filenames or specialised code systems. For these files we have made this metadata explicit. This makes easier

our work in the Use Case 2. More generally, this can help query developers and systems that must manage the lifecycle of metadata. The use of the ontology to store information about properties will make migration of future data simpler. The mapping simply has to be done once between the data and the ontology. This should be especially easy as these existing systems have very strictly defined inputs and outputs and increasingly the formats used are XML based. The process of writing the specific code to extract the information from the datafile and re-encode it as RDF is much simplified, and can be more easily managed.

- **Software.** In the Envisat mission, complex software elements with well-defined interfaces are used in both planning and product generation. Some of these functionalities were incorporated in the prototype (e.g. time conversion utilities) by enabling them to be accessed as Web Services. In a distributed architecture, such as that used in this Semantic Grid, encapsulation allows external systems to interact with individual parts of the system. For example, during a transitional period they might simply make use of a query interface to extract information or an RDF interface to provide some data to an RDF store or annotation component. Use Case 1 ensured we could support existing functionality as well as new techniques.

Flexibility. The system allows a user who is familiar with one of the supported query languages to develop queries directly and iteratively. The process of creating forms which allow other users to provide the specific values for these queries is simple, and made even simpler by the existence of 'common' methods for converting between time formats. In this way we have been able to demonstrate how the system can be enhanced without any significant technical changes being required. This is crucial in an analysis system, where not all relevant queries are known or defined at the beginning of the use of the system.

It is also possible to add in some new relationships or properties without having to change the existing data at all. If a new way of grouping some part of the existing information was required by operators of the system then it could be added into the RDF directly. This aspect of Use Case 3 was explored extensively in the development of user and developer tools as described in the Accessibility section above.

Semantic Technologies and Standards. An Earth observation mission may last 10-15 years and the completely flexible query interface allows exploring of data and development of new queries. This is crucial as anomalies are discovered and examined. Furthermore, new missions may provide information which it would be useful to combine with this existing data. The metadata format and query language defined purposely for the current QUARC implementation, although powerful, cannot be exported and directly used in other systems. The Satellite Mission Grid uses the RDF standard for the storage of the metadata, and the specifics of the format are described by OWL and RDF(S) schemas.

The developed Satellite Ontology has enabled communication, and rapid extension of functionalities as outlined by Use Cases 2 and 3. The addition of "generic" information (such as the linking of event types to instruments) allows us to have semantically rich annotations. This semantic approach to queries where they are moved up to greater levels of abstraction gives us much more flexibility and robustness over time, as we are querying what the users need to know (usage of the instrument) rather than

what we traditionally have recorded (the list of codes used for the event types). This shows the general technique of modeling what it is that users wish to search and discuss. As well as making development more efficient it reduces the time to acquaint new users with the system.

RDF allows us to use standard query languages like SPARQL or RQL which are incorporated in tools which have been already tested and proved adequate for re-use in other systems. The use of standard formats also allows different metadata stores to be used, depending on circumstances. For example, we used Atlas and Sesame as two interchangeable components in line with Use Case 5. Another advantage is in not having to train developers in specific new skills but to be able to use what they already know. However, the existence of several query languages can add an overhead in terms of the required skills for developers.

Data Life Cycle. We have shown that controlled updates can be made to stored data. These are automated but only allowed from authorised data sources. This ability supports the fact that data is not static, and that giving access to the correct information can involve removing out-of-date statements as well as adding new ones, as described in Use Case 4. More generally, we hope to be able to integrate data from many sites and missions and reuse the same system across these missions. As such we have created the methodology and tools for adding new data sources. Lightweight annotation components can convert from a legacy system to common RDF which is made available (via the semantic binding service) to the query engines.

There are high volumes of data being produced by Envisat – anticipated to reach a petabyte of data in 10 years of operation [17]. There are also another four Earth Explorer Missions being launched in the next two years. Extraction and management of just metadata (rather than all the data itself) is necessary for any ongoing analytical systems. In the Satellite Mission Grid, we create a single amalgamated dataset from many geographically dispersed data silos. We then store that dataset across many machines and resources, linked by Grid technology. We move from disconnected data islands having to send their data to a central server to having a single virtualised dataset, spread across the machines. Crucially, all the complexity is absent from the user perspective, i.e. they don't have to know about it at all. The resources and computation are distributed but the user has simple, local browser-based interfaces for annotation and querying.

Modularity and Extensibility. The abstraction of components in a loosely coupled system means we have all the advantages of modularity. Interchangeable components can be selected depending on the particular application of the system, as detailed in Use Case 5. The approach allows the users to enjoy a single interface into whichever systems are determined to be best suited to the current scale of data and complexity of query. We also gain in extensibility; it opens up any further development to using "best-of-breed" components, be they new versions of existing ones, or completely new development. Throughout our work we have adopted an incremental development approach as suggested by Use Case 3.

6 Conclusions and Next Steps

A semantic approach, where metadata is created explicitly and managed as a "first class object" gives advantages of flexibility and extensibility. A Grid approach where data and computations are distributed across many sites and machines gives improvements of scalability and robustness. The prototype system has shown itself capable of carrying out the current functionality of mission analysis systems, but across a geographically distributed dataset. It has also shown itself to be easy to extend in capability without significant development effort.

In the Semantic Data Grid community we have helped focus on lightweight protocols and making components more easy to integrate with existing systems. This vision supports a movement towards SOKU – Service Oriented Knowledge Utilities [18]. The next industry steps include the incremental updating and extension of existing systems, where we will add to the metadata we store and make explicit what was formerly implicit.

Acknowledgments

The authors would like to thank the other members of the OntoGrid consortium and European Space Agency (ESA) representatives Olivier Colin (ESA-ESRIN) and Pierre Viau (ESA-ESTEC) for providing access to the actual products and auxiliary tools from the Envisat mission.

References

1. Sánchez Gestido, M.: OntoGrid Business Case and User Requirements Analysis and Test Set Definition For Quality Analysis of Satellite Missions. Deliverable D8.1, OntoGrid (2005), http://www.ontogrid.net
2. ESA bulletin number 106, EnviSat special issue, http://www.esa.int/esapub/pi/bulletinPI.htm
3. Corcho, O., Alper, P., Kotsiopoulos, I., Missier, P., Bechhofer, S., Goble, C.: An overview of S-OGSA: a Reference Semantic Grid Architecture. Journal of Web Semantics 4(2), 102–115 (2006)
4. Foster, I., Kishimoto, H., Savva, A., Berry, D., Djaoui, A., Grimshaw, A., Horn, B., Maciel, F., Siebenlist, F., Subramaniam, R., Treadwell, J., Reich, J.V.: The open grid services architecture, version 1.0. Technical report, Open Grid Services Architecture WG, Global Grid Forum (2005)
5. Estebán-Gutierrez, M., Gómez-Pérez, A., Corcho, O., Muñoz-García, O.: WS-DAIOnt-RDF(s): Ontology Access Provision in Grids. In: The 8th IEEE/ACM International Conference on Grid Computing (2007), http://www.grid2007.org/?m_b_c=papers#2b
6. http://www.isoco.com/innovacion_aplicaciones_kp.htm
7. Corcho, O., Alper, P., Missier, P., Bechhofer, S., Goble, C.: Grid Metadata Management: Requirements and Architecture. In: The 8th IEEE/ACM International Conference on Grid Computing. ACM Press, New York (2007), http://www.grid2007.org/?m_b_c=papers#2b
8. Miliaraki, I., Koubarakis, M., Kaoudi, Z.: Semantic Grid Service Discovery using DHTs. In: 1st CoreGrid WP2 Workshop on Knowledge and Data Management (2005)

9. Sesame, Aduna Open Source project, http://www.openrdf.org
10. ONTO-DSI, OntoGrid, http://www.ontogrid.net
11. Senger, M., Rice, P., Oinn, T.: Soaplab - a unified Sesame door to analysis tools. In: Proceedings of UK e-Science, All Hands Meeting, Nottingham, UK, September 2-4 (2003)
12. NeOn toolkit, http://www.neon-project.org/web-content/
13. Wright, R: Ontogrid Satellite Use Case demonstration video, http://www.youtube.com/watch?v=TSbb_8vmKvk
14. Allcock, W., Bester, J., Bresnahan, J., Chervenak, A., Liming, L., Meder, S., Tuecke, S.: GridFTP Protocol Specification (September 2002), http://www.globus.org/research/papers/GridftpSpec02.doc
15. Globus (accessed 2008), http://www.globus.org
16. European Space Agency Information Note 13, http://www.esa.int/esaCP/ESA0MDZ84UC_Protecting_0.html
17. Next Generation Grids Expert Group: Future for European Grids: Grids and Service-Oriented Knowledge Utilities, ftp://ftp.cordis.europa.eu/pub/ist/docs/grids/ngg3-report_en.pdf