GES 2007 German e-Science

Available online at http://www.ges2007.de

Stellaris: An RDF-based Information Service for AstroGrid-D

M. Högqvist, T. Röblitz, A. Reinefeld

Zuse Institute Berlin, Takustrasse 7, D-14195, Berlin-Dahlem, Germany
{hoegqvist, roeblitz, reinefeld}@zib.de

Abstract

We present Stellaris, the information service of the community project AstroGrid-D. Stellaris is the core component of the AstroGrid-D middleware that enables scientists to share their resources, provides access to large datasets and integrates instruments such as robotic telescopes. Besides the many diverse types of resources, the information service also supports a wide range of use cases each using a specific schema for the metadata. In addition, Stellaris addresses the distributed and dynamic nature of collaborations in the astronomers' community. Stellaris satisfies these requirements by adopting RDF and SPARQL for storing and querying metadata. Our paper focuses on the requirements of the community, presents the architecture of the information service in detail and discusses experiences with the prototype already in use by partners within the project.

1 Introduction

AstroGrid-D is developing middleware and procedures for enabling astrophysics and astronomy institutes to share resources including compute clusters and workstations, storage capacity, network capacity and robotic telescopes. The scientists have a diverse initial set of use cases including simulations, data reduction and analysis and observations with robotic telescopes. The core component of the middleware is the information service which facilitates storage and discovery of metadata produced by AstroGrid-D users, resource providers and middleware components.

Scientists and software developers using grid technology often need to interface with a fragmented set of tools for managing and querying metadata [12, 7]. For example, a data management service like the Storage Resource Broker (SRB) provides information about the location of files, while a resource directory such as Globus MDS maintains information about compute and storage resources. A unified interface to access and a standard model to represent metadata will impact science beyond grid computing, because it provides an easier way to integrate with external services.

Besides unifying access to metadata, our information service also aims at supporting scientists in large scale collaborations by letting them annotate their compute activities with metadata. Thus, they may ask questions such as: *Was dataset X already analyzed with program Y and parameter set Z? Where is the*

output data from August 12th last year? Why did my last grid job fail? Who created the data producing the graph from the latest number of Science and where can I find it?

Within AstroGrid-D, we distinguish between four different types of metadata: (1) **resource metadata** describes properties of the shared resources, (2) **activity state** reflects the current and logged state of activities in the grid such as jobs and file transfers, (3) **application metadata** describes the program and its progress, and (4) **scientific metadata**, which includes information about the provenance of datasets, studies, etc.

The following requirements were considered vital for an information service supporting the AstroGrid-D users and middleware:

R1. A standard-based uniform interface compatible with existing tools,
R2. support for flexible and extensible metadata schemes,
R3. the integration of the above mentioned metadata types and
R4. authentication and authorization for access control.

Additionally, it is important that the metadata can be transformed into formats understandable by machines and readable by human beings.

In this paper, we present the AstroGrid-D information service, a metadata storage and discovery service with a uniform interface and flexible information model. One of the most important design choices is the use of the Resource Description Framework (RDF) [10] for metadata representation and SPARQL [11] for queries. Metadata vocabularies defined in RDF are easier to change than in the relational model, since a change does not require an explicit update to the database structure. Furthermore, it is straight-forward to add relations between metadata of different types, because the data model uses global identifiers for addressing metadata entries. The implementation of Stellaris also benefits from numerous tools and mechanisms to store, query, design and present RDF-based metadata, that have been and are being developed by the semantic web community.

The remainder of this paper is organized as follows. Section 2 describes the architecture and provides details on the unified information model. Thereafter, we briefly present early experiences from the deployment of a prototype implementation in Section 3 and conclude in Section 4.

2 Requirements and Approach

In the initial phase of the project, we collected requirements from 15 different use cases and analyzed the general characteristics of our grid environment. The main requirements identified after this phase are discussed in this section.

An early design decision was to provide a **unified interface (R1)** for metadata management and query. This decision was motivated by trying to avoid interface fragmentation over several metadata services, to avoid re-implementing software for different metadata systems and to ease the integration of the different metadata types in AstroGrid-D. However, a unified interface also assumes a

common metadata representation format and query language.

The second requirement comes from two observations. First, the use of metadata and how it will be implemented is not fully specified in many of the use cases. Second, we expect that there will be more use cases developed during the project life-time. Thus, we need a **flexible information model (R2)** that allows users to define and update schemas without making changes to the structure of the database. This type of flexibility is difficult to achieve with an RDBMS, where each schema update results in an expensive change of the relevant tables.

Metadata from the different metadata classes defined in the introduction, and from different schemas may reference each other. For example, let a schema describe files by a URI representing a location, a logical name and an owner. A schema for jobs could include an execution site, input/output files (using a logical filename from the file schema), an owner (from the user schema) and a job identifier. Additionally, instances of these schemas may be partitioned over several metadata stores. Therefore, we need a query language that in addition to normal query functionality, such as range and point queries, can **integrate metadata (R3)** from multiple data sources and different schemas.

Data integration between different sources of data is usually difficult for two reasons. First, data is structured differently, e.g. an address can for example be represented as a string or split up into a tuple. Second, the semantics or meaning of schemas may differ, an address can be a postal address or an IP-address. While both aspects are hard to achieve without explicit conversions, we avoid this as long as possible by combining AstroGrid-D specific schemas with established vocabularies (Dublin Core, RTML[1], GLUE Schema[2], etc.).

Central to our approach is the use of a data model for semi-structured [1] data called RDF [10]. Semi-structured data is not unstructured like raw data, but it is also not rigidly defined as the data contained in a relational database. Typical characteristics of semi-structured data, is that the vocabulary for structuring data is large and dynamic. Another difference is that the data in an entry may be partial, i.e. not contain all terms defined in the schema. These three properties, partial data entries, large vocabularies and frequent schema changes make an information model assuming semi-structured data a perfect fit for AstroGrid-D. Moreover, the use of a standard RDF syntax, RDF/XML [2], simplifies the import of data and allows exported data to be used by existing tools.

SPARQL [11] is a query language developed for RDF. In addition to basic query language features like point and range queries, it was designed to write queries using more than one vocabulary and to query multiple sources conveniently. These properties are vital in AstroGrid-D since we need to integrate data from multiple sources using different vocabularies.

The fourth requirement is **security (R4)**. For much of the metadata it is sufficient to restrict write access to the AstroGrid-D Virtual Organization (VO) and to allow public read access. Some of the use cases have more fine-grained requirements on write/read permission patterns for specific users. For example,

[1] Remote Telescope Markup Language

[2] http://forge.gridforum.org/sf/projects/glue-wg

```
@prefix file: <http://www.gac-grid.de/schema/files#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rtml: <http://www.rtml.org/v3.1a#>

<http://storage.gac-grid.de/test/eaglenebula.fits>
   rdf:type <file:DataObject>;
   file:owner "Telescope user";
   file:location <http://telescopes.aip.de/pictures/eaglenebula.fits>;
   file:filesize "259342";
   rtml:Telescope <rtml://de.aip.Robotel/STELLA> .
```

Figure 1: A sample data object description in N3 of a picture taken by a robotic telescope.

in a simulation scenario, a user owning metadata produced by her application may allow public access to see the location of the results and to retrieve the parameters of how the simulation data was generated. But, the current status of the simulation and possible failure information should only be accessible by the operator or a smaller group of users. Authentication and authorization of users is enabled by using X.509 certificates, which is already required for AstroGrid-D users to access other services. In order to accomodate for usage patterns similar to the simulation scenario, we will support user-managed Access Control Lists (ACLs) for the metadata belonging to their application.

3 Metadata Representation and Extraction

The main idea behind the RDF data model is to make statements about resources. A **statement** is divided into three parts; the resource itself, referred to as *subject*, a *predicate*, describing a uni-directional relation to the *object*. The (subject, predicate, object)-tuple is often called an RDF-triple. The subject and predicate are represented with URIs, while an object can be a literal (value) or a URI. A URI can represent both network accessible documents or any type of abstract idea. Moreover, since a URI is a globally unique identifier and the object is also allowed to be a URI, it is possible to join two statements by combining the two subjects or a subject with an object. Combining multiple statements together forms a uni-directional graph, referred to as an RDF-Graph.

Different RDF graphs stored in Stellaris are uniquely identified by a URI. This concept is called named graphs [4], or contexts. Since the name of a graph is a URI, it is possible to make statements about the graph itself. Contexts are used within Stellaris to organize create a tree structure for the stored metadata. Figure 1 shows an example with serialized RDF statements, using Notation 3 [3]. The example describes an image, accessible via the URI defined in the subject. The image has different predicates such as rdf:type, file:owner, file:filesize and rtml:Telescope. Note how predicates from different RDF-vocabularies are mixed within the description.

```
PREFIX file: <http://www.gac-grid.de/schema/files#> .
PREFIX rtml: <http://www.rtml.org/v3.1a#> .

SELECT ?location ?owner ?telescope
FROM <http://telescopes.aip.de/context/robotic_telescopes>
FROM NAMED <http://stellaris.astrogrid.net/context/files>
WHERE {
<http://storage.gac-grid.org/test/eaglenebula.fits> file:location ?location;
file:owner ?owner;
rtml:telescope ?telescope .
}
```

Figure 2: A sample SPARQL query extracting information about a data object.

The SPARQL query language [11], used to query RDF graphs, is syntactically similar to SQL, but diverges in expressiveness to better fit the RDF data model. The base of a query is defined as a conjunction of triple patterns with bound and unbound variables. The WHERE-clause in Figure 2 contains three triple patterns which matches the file location(s), a file owner and a telescope. The query is translated to *Return the file location, the owner and the telescope which generated the file represented by the URI* `<http://storage.gac-grid.org/test/eaglenebula.fits>`. The example also shows how RDF-data from different sources are integrated, using FROM, referencing a remote graph, and FROM NAMED, referencing a graph stored in a local RDF-store.

4 Architecture

The Information Service framework has three components (see Fig. 3): the RDF storage instance (left-hand side), the consumers (in the middle) and the producers of metadata (right-hand side). A single storage mimics the functionality of an RDF-database. This means that it stores RDF persistently (or in-memory depending on setup), provides a management interface and a query interface for fine-grained data extraction. The interface to the storage reflects these responsibilities as it is divided into a query part, implementing the SPARQL query protocol [5], and a part for metadata management, providing the methods create, retrieve, update and delete. A consumer is a client such as a service or an application that uses metadata extracted from the storage. Producers are either creating new metadata or aggregate existing information. Legacy information providers, which output metadata in a non-RDF format, may also be used if the information is translated into RDF. For XML-formats this can be achieved by using GRDDL [6].

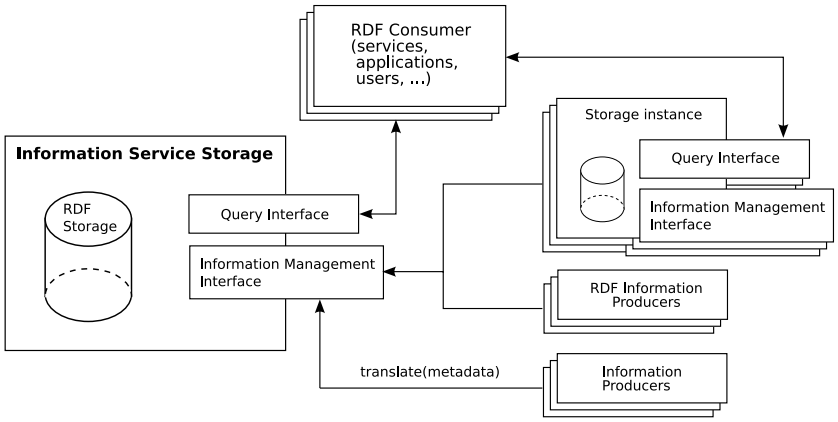

Figure 3: The Information Service framework.

4.1 The RDF-Storage Instance

This section details the functionality of an RDF-storage instance. We focus on the design of the external interface, the context- and collection-concept, the internal implementation and security. Since, future versions of Stellaris will be distributed, some of the implications for a distributed information service are also discussed.

Contexts and Collections. A set of RDF-triples are combined into an RDF-graph, the RDF-graph has a name which we call *context* [4]. A set of contexts are aggregated into a *collection*. We represent a context with a URI, e.g. `http://stellaris.astrogrid.net/files/AABBCC`, and the collection which a context belongs to as the URI up to the last slash of the path-part of the URI, e.g. `http://stellaris.astrogrid.net/files/`. The path represents a hierarchy of collections, however, operations are only performed on direct children of a collection since recursive operations implies a large performance penalty when traversing the hierarchy in a federated setup. Either a context or a collection can be compared to a table in an RDBMS, depending on how the application developer designs its application.

External Interface. The storage interface has the following methods: `create` is used to insert RDF data at a given context, `update` either adds or changes triples stored at a context, `retrieve` gets the metadata stored under a given context, `delete` removes a context and all RDF-triples stored under that context and `query` takes a SPARQL query as input and returns the matching RDF metadata in the standard XML or JSON[3] SPARQL result format. In order to change fine-granular data stored under a context, `update` is able to operate on

[3]http://www.json.org

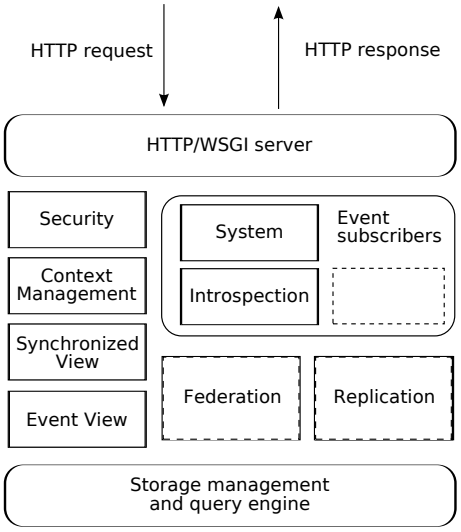

Figure 4: Overview of the internals of a single storage instance.

individual RDF-triples stored at a context. This is necessary when updating dynamic information such as current memory usage at a cluster. There is also a corresponding `remove` function which takes a set of RDF-statements and deletes them from a given context. The interface is implemented in HTTP [8], with the HTTP verbs GET (retrieve), POST (update), PUT (create) and DELETE operating on the URL representing a context. Mapping this interface to XML-RPC/SOAP or other RPC-based protocols would be straight-forward.

Implementation. Figure 4 shows the components of the current implementation of Stellaris. The implementation is based on Python using the RDFLib-library[4] and a Berkeley DB-backend for persistence. The HTTP/WSGI-server uses a thread-pool to handle incoming requests and prepare the responses. A request is first handled by the Security component, which authenticates and authorizes connecting clients. The authorization of users is based on either a static list, such as a gridmap-file, or dynamic information retrieved from a VOMRS service. An accepted request is dispatched to the Context Management module, which provides an internal interface for create/retrieve/update/delete and query. This module then forwards the request to the Synchronized View module, which ensures that concurrent requests are not conflicting. Finally, the Event View receives the request and forwards it to storage management or the query engine. The Introspection and System modules subscribe to the Event View to maintain metadata about contexts, collections and garbage collection. Events based on the type of operation outlined in the interface section. An event is

[4] http://rdflib.net

produced when a request successfully applies an operation to the storage management component. The components for Replication and Federation are still under development. The Replication module will be used to increase reliability for a single deployment. While the Federation component distributes metadata over several storage locations and supports distributed queries.

Since the RDFLib-library does not yet provide a transaction interface for handling updates including more than one triple, we added support for this in the Synchronized View module. In order to provide isolation of concurrent requests, a single-writer/multiple-reader serialization was implemented. Locking is on the granularity of a single context. Additionally, transactions operating on multiple contexts are not supported. Thus, the serialization is deadlock-free. Furthermore, to improve read performance, we cache the data from a context currently being written, enabling concurrent read operations during a write.

Security. In the initial versions, security will not be the main focus since resource metadata such as that retrieved from MDS or the grid activity is not seen as sensitive information. The current approach is to use X.509 certificates for authentication and a VO service such as the VOMRS[5], used within AstroGrid-D to authorize users. Reading metadata stored at a context or extracted from a query is public, while writing requires proper authorization. This allows us to restrict write permission to the group of users representing the AstroGrid-D VO. Stronger security requirements such as ACLs have been identified as necessary for some use cases and will be implemented when necessary. However, it should be noted that the enhanced security from using ACLs will have a negative impact on the query performance since the results must be filtered to comply with the user's access rights.

5 Deployment

Within AstroGrid-D there are a number of early adopters using the prototype of Stellaris. These early users match well with the four classes of metadata outlined in the introduction. First, we have two information producers (IPs) aggregating hardware resource metadata. The first IP is using the GLUE schema instances provided by Globus MDS to add metadata about active AstroGrid-D compute clusters and workstations to Stellaris. Figure 5 shows the different components part of a demo displaying the AstroGrid-D grid resources on a Google Map[6]. The XML-based data is mapped to RDF using a generic translator written in XSLT. The second IP uploads metadata on robotic telescopes derived from the RTML schema. Since RTML instances are in XML, this IP uses the XML to RDF-translator as well.

From the next metadata class, grid activity, three different IPs were developed. The data stream management uses Stellaris to register the different data stream content producers and the components available for a data-stream

[5] http://www.uscms.org/SoftwareComputing/Grid/VO/

[6] http://www.google.com/apis/maps/

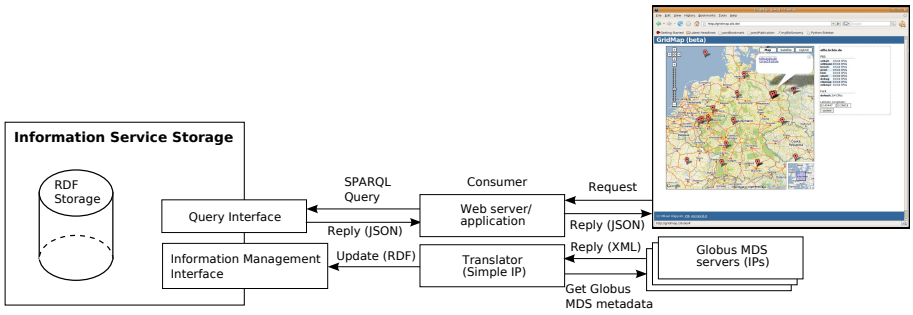

Figure 5: Resource map architecture overview. http://gridmap.zib.de/

processing task. The second grid activity IP collects information on the state of the basic Globus grid services available on the contributed resources within AstroGrid-D. Currently, it is possible to check the status of WS-GRAM, Globus Toolkit 2.x GRAM, GSISCP and GSISSH. The third grid activity IP is a wrapper script used for monitoring of a job submission. It reports job state, job id, job name, X.509 DN of the owner etc. Another demo application, accompanying the grid resource map, shows a timeline containing the history of old grid jobs and the state of current jobs. The demo is based on the SIMILE Timeline[7] and retrieves the up-to-date information from a Stellaris instance.

Finally, an IP for the Cactus use case [9] was developed and is currently used in production by researchers using the Cactus framework. The IP stores metadata related to integration tests of Cactus including the machine where the tests were run, the result status for each test and test suite, configuration options, etc. While the Cactus metadata management uses its own instance of Stellaris, the other IPs are using a common AstroGrid-D instance. Both of these early instances have been showing uptime of several months under different load.

6 Conclusions

We choose RDF and SPARQL as foundation for our information service to meet the diverse and demanding requirements of the scientific use cases in the AstroGrid-D project. A prototype is already in use by several partners within the project. Although, the approach is tailored for the specific requirements of the AstroGrid-D community, we believe that it is generic enough to meet the requirements of other communities and that Stellaris is suitable for the D-Grid Integration (DGI) project's software stack.

[7] http://simile.mit.edu/timeline/

Acknowledgements

This work is supported by the German Federal Ministry of Education and Research within the D-Grid initiative under contract 01AK804C. We would also like to thank Florian Schintke, Kathrin Peter, Tobias Langhammer and Felix Hupfeld for in-depth technical discussions and suggestions for improvements.

References

1. Serge Abiteboul. Querying semi-structured data. In *ICDT '97: Proceedings of the 6th International Conference on Database Theory*, pages 1–18, London, UK, 1997. Springer-Verlag.
2. D. Beckett. Rdf/xml syntax specification (revised). `http://www.w3.org/TR/rdf-syntax-grammar/`, February 2004.
3. T. Berners-Lee. Notation 3. `http://www.w3.org/DesignIssues/Notation3`, March 2006.
4. J. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs, provenance and trust. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 613–622, New York, NY, USA, 2005. ACM Press.
5. E. G. Clark. SPARQL protocol for RDF. `http://www.w3.org/TR/rdf-sparql-protocol/`, January 2006.
6. D. Connolly. Gleaning resource descriptions from dialects of languages (grddl). `http://www.w3.org/2004/01/rdxh/spec`, March 2007.
7. Ewa Deelman, Gurmeet Singh Singh, Malcolm P. Atkinson, Ann L. Chervenak, Neil P. Chue Hong, Carl Kesselman, Sonal Patil, Laura Pearlman, and Mei-Hui Su. Grid-based metadata services. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 393–402. IEEE Computer Society, June 2004.
8. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616, Hypertext Transfer Protocol – HTTP/1.1, June 1999.
9. T. Goodale, G. Allen, G. Lanfermann, J. Masso, T. Radke, E. Seidel, and J. Shalf. The cactus framework and toolkit: Design and applications. In *Vector and Parallel Processing - VECPAR '2002, 5th International Conference*. Springer, 2003.
10. F. Manola and E. Miller. RDF primer. `http://www.w3.org/TR/rdf-primer/`, February 2004.
11. E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. `http://www.w3.org/TR/rdf-sparql-query/`, February 2006.
12. N. Santos and B. Koblitz. Metadata services on the grid. In *Proceedings of Advanced Computing and Analysis Techniques (ACAT'05)*, May 2005.