

# A Resource-specific Approach to Building Information Services: Experiences from the LEAD Project

Yogesh L. Simmhan, Beth Plale, and Dennis Gannon  
Computer Science Department, Indiana University, IN 47405 USA  
{ysimmhan,plale,gannon}@cs.indiana.edu

## Abstract

*(FIXME)Several contrasting ways to build an information service. Driven by the type of resource we are cataloging. Data and service are two main resources registered with the ResCat. Data comes in batches at certain periods. Services are more irregular and transient. Volume of data is much more than number of services. (Pull vs. Push; Soft-state lifetime, or no lifetime support). Data is published in other catalogs (rescat is meta-catalog). Data has a single schema and specific fields on which it is queried. Services have a more generic and heterogeneous schema. (XML vs. indexing) Data usually queried by users. Services by programs (complex, but rich query vs. boolean filters)*

## 1. Introduction

Grids and Science Gateways [10] provide uniform and secure access to community resources based on a layered service oriented architecture (SoA). These gateways are allowing scientists to engage in multi-disciplinary projects that span geographical and domain boundaries while having uniform access to computational and storage infrastructure. Such seamless transparency is achieved by virtualizing the physical resources through well defined interfaces that allow computational clusters and massive storage repositories to be accessed as services. In addition, several core services provide the necessary capabilities for a science gateway to function effectively [?]. Security services for provide authentication and identity mapping; data services move, store, replicate, and access data [24]; execution management services plan, schedule, and manage the lifecycle of jobs run on the Grid; and information services [16, 6] catalog the current status of Grid resources and enable their discovery by other components in the Virtual Organization.

Information services act as brokers for information interchange between different parts of a Grid. By maintaining metadata about resources, they act as directory services that can be contacted to query for and locate a resource. They

provide name and location transparency and allow collaborating users discover shared resources in the system without requiring prior knowledge about them. By maintaining current information about resources, they assist in dynamic matchmaking of resource needs to availability. The resources described by the information services can be diverse, and there lies one of the challenges in designing generic information services. Common physical resources in a Grid are computational clusters, data storage locations, network bandwidth, and instruments and sensors. R-GMA [5] used in the EU DataGrid project and GridRM [2] are examples of information service for physical resources. In a SoA, these resources have service counterparts that virtualize the access to the resources. There may exist application services that wrap the capability to launch scientific applications on a cluster [12]. Data products can be streaming in or be generated from computational runs and staged in various storage repositories. These services and data products are virtualized resources and this paper deals with information services for such resources. Resources can be owned and shared by the community or owned by a user. While the superficial difference between information services for these public and private resources is the degree and granularity of access control to the resource metadata, information services for the latter have a user focus that provides capabilities for users to organize their resources along conceptual groups or hierarchies. Examples of this can be found in myLEAD [15] and myGRID [22]. We restrict our scope to community owned resources.

The fundamental capability provided by an information service is to register metadata about a resource and allow it to be queried for an returned, and there are several systems that exist that do this. Indeed, a relational database or an in-memory hashmap structure seem to satisfy this definition, but do not because of the unique set of facilities required off an information service in a Grid environment [16, 6]. The information service has to handle diverse metadata representations. It should be able to recover from failures in the substrate and of the resources. It needs to provide lifetime management capabilities for the resource metadata

and scale to the extent necessary. It must provide query capabilities that meet the requirements and handle different types of metadata providers. And all these requirements will vary with the type of resource that is being managed by the information service. From our experience in the *Linked Environments for Atmospheric Discovery (LEAD)* project [9], an all in one information service that handles every kind of resources is unlikely to be effective and information services will have to be configured according to cater to individual resource types.

In this paper, we describe our efforts in building an information service for community resources in the LEAD Cyberinfrastructure. The Resource Catalog is a meta information service that manages resource information for applications, services, and data products in LEAD, taking a resource specific approach to managing the metadata. It is service since the Resource Catalog can be conceptually distinguished into two component services: the Service Catalog which acts as an application and service registry, and the Data Catalog which manages metadata for community data products. These two catalogs expose independent service interfaces that are combined together in the Resource Catalog service interface for convenience, to give the impression of a single information service. We describe the architecture and implementation Service and Data Catalogs and highlight the unique characteristics of these two primary types of resources that require distinct attention. The paper is organized as follows: after a brief introduction of the LEAD Cyberinfrastructure and the Resource Catalog in Section 2, we first describe the architecture of the Service Catalog in Section 3 followed by the Data Catalog in Section 4. A discussion of the two information services' characteristics ensues in Section 5 and we end with future plans and conclusions in Section 6.

## 2. Information Services in the LEAD Cyberinfrastructure

(FIXME: Fill 0.5-1 column with LEAD info)

The rescat's functionality can logically be split into the Service Catalog and the Data Catalog. They are respectively used to manage information about various Service resources and community Data resources present in the LEAD Cyberinfrastructure.

Brief description of LEAD Grid – workflows, services, personal and community data, meteorology simulations on clusters, portal interface

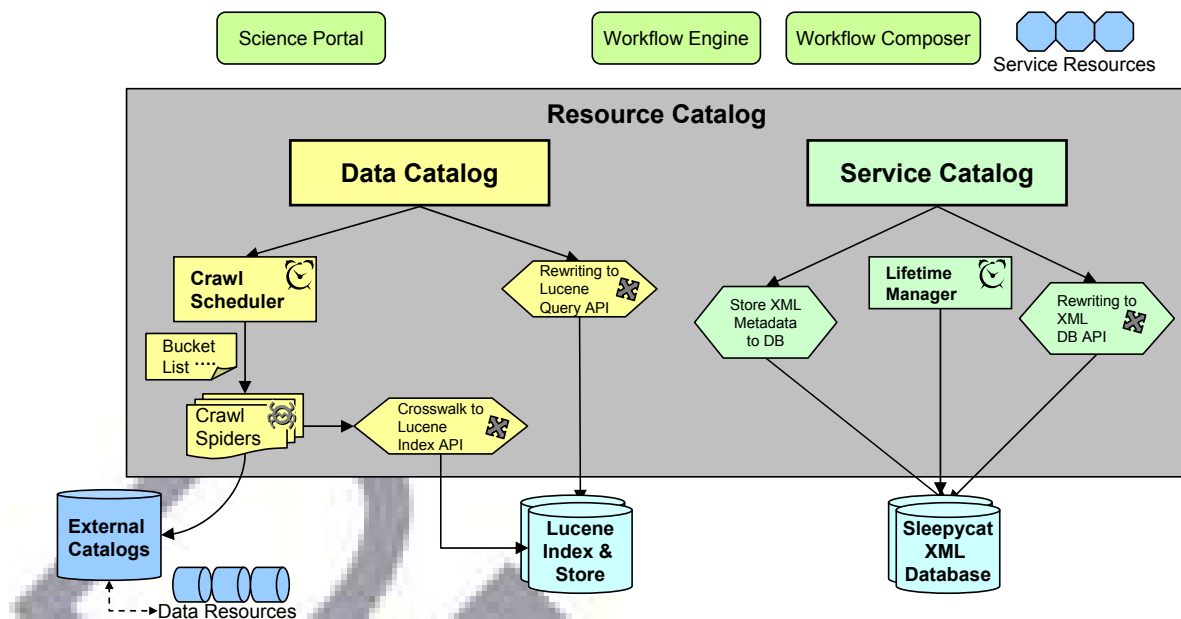
Information services – myLEAD, ResCat, Notification Broker, THREDDS, Provenance service

## 3. Service Catalog

Application services [12] are important actors in a science gateway and are responsible for performing the scientific computations and data transformations. A Service Catalog, other than registering currently available service, also needs to record resources that describe service templates and necessary information to instantiate a new service. Several metadata documents go towards describing the lifecycle of a service, making them multi-faceted entities.

LEAD application services usually wrap a scientific application, which can be any command-line executable and is usually a FORTRAN program or a shell script [12]. They are designed as web services and described using a WSDL document that contains syntactic information on the messages accepted by the service interface. A web service that is active and running also has its endpoint reference contained in the WSDL, and the document is termed a *concrete WSDL* document. The concrete WSDL is only valid during the lifetime of the service. Services that have been configured to launch an application but have not yet been activated are abstract services. These are described using a *Service Map document*, which has a reference to the application it wraps, and syntactic and semantic descriptions of the interface provided by that service. Service Maps are defined by the application service provider and used to automatically generate a corresponding *abstract WSDL*. The external application referenced by the service is itself described by an *Application Description* document, that contains deployment information and application runtime requirements. Finally, *Host Description* documents describe the facilities available at a certain host machine or cluster, such as versions of software, type of scheduler, and so on that help determine if an application can be run on a particular host. All these documents are represented in XML using a well defined schema.

A service instance, described by the concrete WSDL, is built by matchmaking information from the Service Map, the abstract WSDL, the Application Description document, and Host Description document, and the Service Catalog needs to support registration and querying over all of them. When a user needs to run an application or launch a workflow, they initially search for service instances i.e. concrete WSDLs based on some attributes in the concrete WSDL or in the Service Map for that service. If a service is not already running, it needs to be created and launched. This requires selecting the Service Map for the service, retrieving the Application Description document for the application referenced in the Service Map, locating a host where the service can be launched, creating the abstract WSDL for the service and annotating it with the service endpoint to generate the concrete WSDL for the service. All these



**Figure 1. Architecture of the Resource Catalog, comprising of the Data and Service Catalogs**

documents are registered and discovered from the Service Catalog.

### 3.1. Architecture

The Service Catalog needs to record different types of XML description documents for a service and provide rich querying capabilities to support brokering and searching needs. It also needs to be extensible to other service related resources that may need to be defined future. To handle these aspects, the Service Catalog uses a native XML database in the backend to store the XML documents, while providing complex query support using XQuery.

**REGISTERING.** The Service Catalog web service provides a simple API, built on top of the Sleepycat Berkeley DB XML database (now part of Oracle) [14], to allow clients to register the five different types of service related metadata description documents. The API also enforces dependencies and restrictions that exist between the metadata documents. For example, an abstract WSDL document cannot exist without a Service Map document present for it. Each type of resource metadata is stored in a separate XML collection in the database.

**QUERYING.** The Service Catalog also provides a query API to perform common search queries on the different documents. The simple queries include locating a service instance (concrete WSDL) by its porttype QName (service interface type), or a host (Host Description document) by its hostname. These queries are mapped to equivalent XQueries that are executed on the DB XML database to

retrieve matching resource documents. In addition, complex queries on any element or attribute in the metadata description, and joins across different metadata documents can be directly performed by clients using XQueries. Clients use pre-defined names for the metadata collections when writing the XQuery and these are internally mapped by the Service Catalog to the real name of the collection used in the database. Using this API, a resource broker can, for example, perform matchmaking and retrieve all documents required to create a service using a single XQuery [13].

**LIFETIME MANAGEMENT.** An other important capability that the service catalog provides is lifetime management of metadata for transient resources. Service instances in LEAD are usually short-lived, existing for the duration of a workflow they are part of and then terminating. Services normally unregister themselves from the Service Catalog when they terminate. In a distributed environment such as the Grid, there is potential for failure of different hardware and software resources. So, it is not possible to guarantee that a service will always be able to die gracefully by unregistering itself, or that the Service Catalog will be reachable when the service instance terminates. We use the technique of softstate lifetimes assigned to metadata for transient resources for handling such scenarios. When concrete WSDLs, the primary transient metadata, is registered, it is assigned a lifetime (or a lease) for a short duration of time (default 15mins). If the service instance does not renew its lease within this time, the concrete WSDL is removed by a garbage collection thread running in the service catalog, assuming that the service instance died without unregistering

itself. If the service were still running and it could not renew its lease due to some resource failure, it can re-register itself when the failure is rectified. However, the concrete WSDL metadata would have been missing for the interim time when the lease had expired. This ensures a graceful degradation of the quality of service commensurate with the failure(s) in the system.

In addition to metadata being removed due to lifetime expiration, users can also manually unregister/delete the resource metadata using APIs similar to the query API. This is the method used by persistent resource metadata such as host description document and service map documents.

**ARCHIVING AND RECOVERY.** Information services such as the service catalog form a central piece in discovery of resources and need to guarantee an acceptable quality of service. In addition to disk failures that can cause corruption of metadata stored in the service catalog, use of prototype software libraries can also affect the contents of the XML metadata database. In order to ensure timely recovery in the event of such failure, the Service Catalog maintains an exponential and rolling backup of the contents of the XML database. A separate archive thread in the Service Catalog creates a series of logs of the raw XML metadata from the database at configurable periods, presently at 1, 2, 6, 12, 24, and 72 Hour intervals. In the event of a failure or database corruption, the catalog can automatically be switched to the most recent usable archive. These snapshots also allows us to track the services that have been running in the past to mine them for resource usage patterns or help track provenance [19].

## 4. Data Catalog

Data products form a very important resource in LEAD. Data products may be available as files, collections, and catalogs, and describe raw instrument data, model generated data, streaming data, experiments, and logical collections of these. LEAD makes the distinction between personal data owned or shared by a user, and public data available to the community. While metadata on private data is managed by the myLEAD [15] personal catalog, our focus in this section is metadata on community data managed by the Data Catalog part of the Resource Catalog.

In order to describe diverse earth sciences data in a standard manner, LEAD has adopted a variation of the Federal Geographic Data Committee's Spatial Data Transfer Standard [1]. Called the LEAD Metadata Schema (LMS) [17], this XML description captures spatial and temporal attributes, citation, provenance, data quality, and format description, and also allows extensible entities and attributes. This schema is exclusively used to describe every data product in the LEAD domain. It also allows hierarchical composition of resources through the notion of collections that

can reference other collections or data products through the global ID for the LMS document.

Community data products enter the LEAD domain primarily through data flowing into the THREDDS Data server. The THREDDS data server (TDS) [8] is a middleware often used by the earth sciences community as a repository for data from various sensors and instruments. Data, coming in periodically, is organized hierarchically as files and directories according to predefined configurations, and an XML document describing the contents of each directory is created and updated based on a manually defined template for each directory. These XML documents are called THREDDS catalogs [8] and describe spatio-temporal attributes, data format and access protocols, publisher, and so on. These catalogs are accessed from the TDS through a HTTP URL. The hierarchical structure of the directories means that the catalogs themselves can point to child catalogs in addition to listing data products. These catalogs form one of the main sources of public data for the Data Catalog.

Data products can also be published from the myLEAD personal catalog to be shared among the LEAD community. These data products are already described using LEAD Metadata Schema and can be directly added to the Data Catalog.

### 4.1. Architecture

The Data Catalog service can either have data product metadata in the LMS pushed to it for publishing, or have THREDDS catalogs registered with it as a source for pulling in data product metadata. The former scheme is used for shared data published from myLEAD and the latter is the source of external data entering the LEAD community. This LMS metadata is indexed and stored by the Data Catalog to allow for complex queries based on attributes defined over the LMS.

**CRAWLING.** The Data Catalog uses a register and crawl approach to pulling information from the THREDDS catalogs. One or more THREDDS catalogs, encompassing community data products of interest to LEAD, are registered with the Data Catalog along with a period of crawling that approximates the data at which data comes into the THREDDS catalogs. This is usually at the frequency of every hour. At the prescribed frequencies, the Data Catalog launches spiders [4] to start crawling from the registered THREDDS catalogs to all other catalogs linked from it recursively, searching for data products present at the leaf of the catalog tree. When a data product is found, the THREDDS metadata description for it is extracted and added to a separate crosswalk queue in the Data Catalog. Several crosswalk threads are waiting on this queue for new THREDDS metadata objects to come in, and they asyn-

chronously convert the THREDDS metadata to the LEAD Metadata schema. Once converted, these LMS documents are indexed using the Apache Lucene library [11] and stored.

When THREDDS catalogs are registered with the Data Catalog, they are dropped into different buckets based on the frequency at which they need to be crawled. A monitoring thread starts a crawler to crawl over all THREDDS catalogs in a bucket at the frequency specified for a bucket. THREDDS catalogs, when registered, are put in the bucket whose crawl frequency is closest to the crawl frequency of that THREDDS catalog. This optimizes the crawling by using a batch crawl mode instead of crawling each THREDDS catalog independently.

**PUBLISHING.** Data products and collections published from the myLEAD catalog are already described using LEAD Metadata Schema. While data is pulled from the THREDDS catalogs, users push data products and experiments they wish to share from myLEAD to the Data Catalog. Also, publishing from myLEAD is supported for collections of related data products and experiments, and not just individual data products. These cohesive collections, while composed of several individual LMS documents, can also be retrieved as a single collection with all related LMS instances for, say, import into myLEAD.

**INDEXING.** The Data Catalog needs to support storing and querying over hundreds of thousands of data products and collections. Usually, a trade-off occurs between the efficiency of querying and the generality of queries. Given the scalability requirements in the Data Catalog, we have opted for an indexing and store strategy that shreds and indexes important and commonly queried metadata attributes from the LMS XML document while also retaining the original LMS document instance to be returned as the result of queries. A configurable set of 22 commonly queried typed metadata attributes were identified and an index built on each of those dimensions using the Apache Lucene [11] index library. Some of the attributes like temporal and spatial ranges are multi-dimensional while others like data format and citation have a single dimension.

When a LMS document instance needs to be indexed and stored in the Data Catalog, an XML parser extracts the attributes configured for indexing and passes it through special tokenizers written for specific attribute types since Lucene natively supports only String types. So a numeric data type like latitude or longitude will have to be padded with zeros and signed to enable lexical indexing and comparison of numbers. The tokens generated for the attribute value are added to the index for that attribute and associated with the unique resource ID for the LMS document. In addition, all attributes are tokenized as strings and added to a separate index that allows free text search on the LMS document. The index for each attribute is a B-tree that is

persisted in the file system by Lucece. Each crawled bucket maintains its own set of B-trees for documents from its crawl.

The Data Catalog maintains two sets of indices for each bucket to allow the availability of an index for querying while a new index is build from the periodic crawl. While a “live” index is used for answering queries, there may be another “background” index being built from a fresh crawl. Once the crawl completes and a new index is built for the THREDDS catalogs in a bucket, the “background” index is swapped with the “live” index and henceforth used to answer queries. The separate index maintained for the published data is only appended to with additional documents when data products are published directly to the Data Catalog, and no new index is built. Since correlating existing documents in the index with those in the THREDDS catalog is costly, such a complete recrawl strategy effective when the number of data products updated in each catalog update is high. However, an incremental approach is preferred and scalable when the delta change between crawls is smaller. We are currently working on this.

**QUERYING.** The Data Catalog allows queries over the predefined metadata attributes in addition to a free text search over all of the LMS metadata. Basic comparison and range queries can be defined over individual attributes and these can be combined recursively with AND, OR and NOT operators to form arbitrarily complex boolean queries. The comparison operators supported are equals, greater than, less than, substrings, and wildcards, while range queries for spatial and temporal attributes allow matching on subsets, supersets, and intersects of a value ranges. The queries are represented by the user using an XML schema and translated by the Data Catalog to query terms compatible with the Lucene API. This involves mapping the attribute names in the query to the attribute names used in Lucene, tokenizing the attribute values to match the tokenization of the indexed attribute values, and recursively converting the XML query predicates to equivalent Lucene query terms in Java. The resource ID of LMS documents matching the queries are returned by the Lucene query. The raw LMS document is then retrieved using this ID and returned to the query client. Currently, the top “N” data products are returned by the Data Catalog, sorted according to their temporal ranges. Work is on to allow incremental return of matching documents.

## 5. Discussion

Building the Service and Data Catalogs gave us insight into the distinct challenges posed by the different resources. There are several tools available to build Information Services and keeping the implementation simple by leveraging these tried and tested tools helped. The Berkeley XML DB

and the Apache Lucene libraries are cases in point. Below, we discuss the various key differences between the Service and Data Catalogs.

### 5.1. Push vs. Pull

Metadata can either be explicitly created and sent to the information service, or the information service can be notified to (periodically) retrieve the metadata from a network location. The approach to use depends on the type of resource and the metadata provider. The two factors that are traded-off are scalability of the system and sensitivity to change in the resource properties. These depend on various attributes such as the number of resources, lifetime of a resource, and rate and periodicity of change (add/update/delete).

When the number of resources is large, a pull model is better since the information service can play an active role in staggering the resource updates to enhance scalability. This can be enhanced if multiple resources can be simultaneously pulled from a single provider and if the resource updates are periodic. These optimizations can improve the scalability without severe loss of sensitivity. However, if resources have a short and intermittent lifetime with a high rate of change, a pull model is less sensitive to the changes since the state between pulls is not available. Increasing the pull frequency will affect the scalability. Another obvious hazard is when the resource is behind a firewall or cannot be reached for a state update. The issue of lifetime management in both these cases is discussed later too.

In the ResCat, the number of service resources are in the order of hundreds and their lifetime usually ranges from several minutes to a few hours. This prompted us to use a push model where the services register and unregister themselves. The service resources were also time sensitive since they are used during workflow composition, configuration, and orchestration. The data resources, however, numbered in the thousands and were streaming in batches every hour to an external catalog. A pull model is appropriate here to scale to the large number of resources.

### 5.2. Persistence Mechanism

Information services persist metadata about resources to an external storage service. In-memory storage is insufficient to guarantee the survival of data upon service failure or to scale with the number of resources. Several persistence mechanisms exist and their selection depends on the complexity of the metadata and the richness of query functionality required. We restrict our discussion to XML metadata since that is the most popular metadata representation used.

If the query functionality required is just a lookup based on a unique ID of the resource metadata, persistence can just mean writing each metadata document to a unique file in the file system. But this is rarely sufficient. Two approaches that can be used for more common cases are databases (in different flavors, such as relational, XML, object, and semantic), and an index-and-store approach. The former is more commonly used. It can mean defining tables or collections to map the resource metadata to and translating metadata addition and updates to database update queries. Likewise, the stored metadata can be queried for by mapping the Information service's query API to the native database language. The advantage of using databases lie in the flexibility and generality they provide. It is possible to map almost any metadata representation to a generic database schema and they support complex update and rich querying capabilities. Mature database systems also offer scalability. However, the downside is the complexity involved in mapping the metadata schema and queries to the underlying database.

Another persistence and query mechanism that has been used in the web search community but can be equally well adapted to Grid information services is one of building indexes for the metadata [7]. A simple implementation of a web search engine would crawl web pages and index different attributes of the page, such as the title, text content, and meta information present in the page. These are mapped back to the URL for the webpage, which can also be cached. A search on this engine does a lookup of the index and returns the matching URLs. A similar approach can be used in information service, with the metadata forming the web page, indices built upon attributes extracted from the metadata and mapped to the metadata's ID, and the metadata itself stored (cached). Searches can be based on the attributes that are indexed and more complex than free text searches done by a search engine. If the metadata is not updated frequently and the number of attributes on which they are queried upon is limited, then this is an elegant solution that merges the indexing efficiency of databases and the storage simplicity of file systems with minimal programming overhead. Indexing libraries, such as Apache Lucene, allow the creation of index structures and even storing raw documents, with support for boolean and range queries.

As mentioned earlier, the Service Catalog uses a native XML database to store and query over the service metadata. This is due to the wide variety of metadata schemas that need to be accommodated, the complexities of the schemas, and the need for complex queries over arbitrary elements across schemas. The generic nature of storing and querying over XML documents using XQuery provided by the Sleepycat XML DB was well suited for this. The Data Catalog uses an index and store approach using the Apache Lucene library. This choice was guided by the fact that metadata

on data resources arrive in batches that were suited for en-mass indexing, the number of attributes to query upon was restricted and well known, and the types of queries were limited to comparison and spatio-temporal range queries.

### 5.3. Query Functionality

The kind of query capability required depends on the complexity of the metadata describing the resource and the needs of the applications querying the catalog. Both the services and data products were described using XML metadata. Though the individual metadata descriptions for each service resource was simple, there were 5 different metadata schemas and the need to query across them using joins. For example, a typical search of the Service Catalog can be for service instances that are running a certain application and on specific hosts. This would involve a join between information in the concrete WSDL, Service Map, Application Description and Host Description collections. These are easily expressed using the XQuery language and the queries themselves are usually executed by brokers or workflow engines, making it easier to automatically generate these queries without human intervention. The rich query requirement is one of the key drivers of using a native XML database. If simpler querying over limited attributes sufficed, a relational database or even an indexing and store technique would be acceptable.

The Data Catalog has more direct users as clients who perform queries from an online Portal and they are more comfortable with the kinds of boolean queries supported by the Data Catalog. The web interface has different sections to fill in values for the various attributes, such as the spatial and temporal ranges, data types and formats, publisher information and keywords. Once filled, these can be combined using AND and OR operators to form boolean queries that match data products. This solution is also more scalable since the number of data products are 2 orders of magnitude more than services – typically 100's of services and 10,000's of data products. Also, despite the LMS being more complex, the key attributes that are searched upon are more limited and a free text search upon the other attributes suffice.

### 5.4. Lifetime Management

Lifetime management is a key issue in information services for distributed resources to ensure the service has a reasonably accurate representation of the current state of the system. For resources that change very rarely, active lifetime management may not be required. While the best practise would be for a resource to unregister itself before it terminates, this may not always be possible and the overhead of softstate lifetime management for persistent re-

sources may not be required. Instead, clients that discover stale resources can report them to the information service that would schedule its removal. Examples where this could be used are for Application and Host Descriptions and for abstract services. Alternatively, the resources can also be polled at frequent intervals using ping messages to see if they are alive. Softstate lifetime management works best for transient resources such as service instances. Here too, the selection of the lease time should be done keeping in mind the tradeoff between overhead for renewing leases frequently and the sensitivity of the system to stale entries.

Passive resources like data products, that are not aware of when they terminate – for example, when a scour script archives or garbage collects old data – pose a more interesting problem. Either their metadata provider, for data products in LEAD, the THREDDs catalogs, need to notify the Data Catalog of the deprecation of data products when it detects the data being deleted, or the Data Catalog should poll this information from the metadata provider. The periodic recrawls behave as a form of polling. These crawls effectively refresh the lifetime of data products that still exists, remove entries for data products that are absent, and add entries for new data products. Though this does not happen in precisely this manner, i.e. as an incremental update of the index, instead rebuilding the entire index, this does achieve the lifetime management of the data resources.

### 5.5. Handling Failures

(FIXME: Retain section of there is space)  
Exponential backup metadata,  
backup catalogs and recrawl,  
softstate lifetime,  
exponential backoff when crawling,

### 5.6. Related Work

(FIXME: Organize)

Most Grid systems have some form of information service. The Metadata Directory Service in the Globus project [6, 20, 21] has extensions that support the Grid Resource Information and Registration Protocols. ... Grimoires is a service registry that supports semantic annotations [23]... UDDI and LDAP provide standards for directory services and several projects have extended these protocols to their needs [18, ?]... Several web service registries also exist [?, 3, ?]... myGrid provides service annotations. myLEAD provides data catalogs.

## 6. Future Work and Conclusion

The Resource Catalog has been deployed and in use for over 18 months in the LEAD project. It presently has 100's

of service resource entries and about 60,000 data products registered. It is continuously being improved with an intention of scaling to several 100,000 data products. There are two key improvements that can achieve this. The crawling of external catalogs needs to be more *adaptive*. This can be done by first doing an incremental rebuilding of indices when crawling, propagating only the delta changes about newly added and removed data products. Additionally, the crawl frequency can be adapted dynamically so that the THREDDS catalogs can migrate between buckets based on the frequency at which the crawls find new updates. For example, the crawl frequency may have been set at 30mins for all THREDDS catalogs by an administrator, but if some catalogs are found to be updated only every 120mins, we can use heuristics to index those catalogs at a lower frequency. We are also working to improve the query scalability using Lucene by generating *pre-filtered indices* based on temporal ranges. Temporal ranges inevitably form a part of every data product query and building specialized subsets of indices over predefined intervals, say “Now through 24 hours back” or “within the past week” would reduce the search space drastically and reduce the memory footprint required to execute a query.

We have described our experiences building information services for different types of resources in the LEAD Cyberinfrastructure project...(FIXME: clean end)

## References

- [1] Spatial data transfer standard. Technical Report ANSI NCITS 320-1998, ANSI, 1998.
- [2] M. Baker and G. Smith. Gridrm: An extensible resource monitoring system. In *Cluster Computing*, 2003.
- [3] M. Bubak, T. Gubalstroka, M. Kapalstroka, M. Malawski, and K. Rycerz. Grid service registry for workflow composition framework. *LNCS*, 3038:34–41, 2004.
- [4] S. Chakrabarti. *Mining the Web: Discovering Knowledge from HyperText Data*. Science & Technology Books, 2002.
- [5] A. W. Cooke, A. J. G. Gray, L. Ma, W. Nutt, J. Magowan, M. Oevers, P. Taylor, R. Byrom, L. Field, S. Hicks, J. Leake, M. Soni, A. J. Wilson, R. Cordenonsi, L. Cornwall, A. Djaoui, S. Fisher, N. Podhorszki, B. A. Coghlan, S. Kenny, and D. O’Callaghan. R-gma: An information integration system for grid monitoring. 2888:462–481, 2003.
- [6] K. Czajkowski, C. Kesselman, S. Fitzgerald, and I. Foster. Grid information services for distributed resource sharing. In *HPDC*, 2001.
- [7] D. Deniman, T. Sumner, L. Davis, S. Bhushan, and J. Fox. Merging metadata and content-based retrieval. *Digital Information*, 4(3), 2006.
- [8] B. Domenico, J. Caron, E. Davis, R. Kambic, and S. Nativi. Thematic real-time environmental distributed data services (thredds): Incorporating interactive analysis tools into nsdl. *Digital Information*, 2(4), 2002.
- [9] K. K. Droegemeier, D. Gannon, D. Reed, B. Plale, J. Alameda, T. Baltzer, K. Brewster, R. Clark, B. Domenico, S. Graves, E. Joseph, D. Murray, R. Ramachandran, M. Ramamurthy, L. Ramakrishnan, J. A. Rushing, D. Weber, R. Wilhelmson, A. Wilson, M. Xue, and S. Yalda. Service-oriented environments for dynamically interacting with mesoscale weather. *Computing in Science and Engg.*, 7(6):12–29, 2005.
- [10] D. Gannon, B. Plale, M. Christie, L. Fang, Y. Huang, S. Jensen, G. Kandaswamy, S. Marru, S. L. Pallickara, S. Shirasuna, Y. Simmhan, A. Slominski, and Y. Sun. Service oriented architectures for science gateways on grid systems. In *ICSOC*, 2005.
- [11] E. Hatcher and O. Gospodnetic. *Lucene in Action*. Manning Publications, 2004.
- [12] G. Kandaswamy, L. Fang, Y. Huang, S. Shirasuna, S. Marru, and D. Gannon. Building Web Services for Scientific Grid Applications. *IBM Journal of Research and Development*, 50(2/3):249–260, 2006.
- [13] A. Kini, S. Shankar, J. F. Naughton, and D. J. Dewitt. Database support for matching: limitations and opportunities. In *SIGMOD*, pages 85–96, 2006.
- [14] Oracle Corporation. Berkeley DB Java Edition, 2006.
- [15] B. Plale, J. Alameda, B. Wilhelmson, D. Gannon, S. Hampton, A. Rossi, and K. Droegemeier. Active management of scientific data. *IEEE Internet Computing*, 09(1):27–34, 2005.
- [16] B. Plale, P. Dinda, and G. von Laszewski. Key concepts and services of a grid information service. In *International Parallel and Distributed Computing Systems*, 2002.
- [17] B. Plale, R. Ramachandran, and S. Tanner. Data management support for adaptive analysis and prediction of the atmosphere in lead. In *Interactive Information Processing Systems for Meteorology, Oceanography, and Hydrology*, 2006.
- [18] A. ShaikhAli, O. F. Rana, R. Al-Ali, and D. W. Walker. Uddie: An extended registry for web services. In *SAINT*, 2003.
- [19] Y. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34(3), 2005.
- [20] G. Singh, S. Bharathi, A. Chervenak, E. Deelman, C. Kesselman, M. Manohar, S. Patil, and L. Pearlman. A metadata catalog service for data intensive applications. In *Supercomputing*, 2003.
- [21] W. Smith, A. Waheed, D. Meyers, and J. Yan. An evaluation of alternative designs for a grid information service. *Cluster Computing*, 4(1):29–37, 2001.
- [22] R. D. Stevens, A. J. Robinson, and C. A. Goble. mygrid: personalised bioinformatics on the information grid. *Bioinformatics*, 19(90001):302i–304, 2003.
- [23] S. C. Wong, V. Tan, W. Fang, S. Miles, and L. Moreau. Grimoires: Grid registry with metadata oriented interface: Robustness, efficiency, security — work-in-progress. In *Work in Progress Session in Cluster Computing and Grid*, 2005.
- [24] Yogesh L. Simmhan and Sangmi Lee Pallickara and Nithya N. Vijayakumar and Beth Plale. Data Management in Dynamic Environment-driven Computational Science. In *IFIP Working Conference on Grid-Based Problem Solving Environments (WoCo9)*, 2006.