Notes on Browsing tools for Replication Services

A. Domenici

December 7, 2011

Abstract

These notes describe two graphical browsers used to display information on replications services, i.e., the components of a Data Grid that create and manage replicas of data sets. These browsers have been developed within the European Data Grid project.

Keywords: replica management, grid computing, data grid, replica catalog, replica location server.

1 Introduction

A key aspect of a data grid is *replica management*, i.e., a set of grid services and user tools that enable applications to create and access copies of datasets. Replica management should in general be transparent to end users: a user should be able to submit a job by specifying just the intended computation and the datasets it uses or produces (plus any requirements on the computation environment). Such datasets should be identified abstractly by a logical name that does not reference or imply any physical location. A replica of each dataset would then be chosen from the available replicas, or created anew, by the replica management service, according to such criteria as performance, cost, or access policies. The replica management itself relies on some replica catalog service that maps logical names to the locations where replicas are stored.

However, tools that directly manipulate or expose the information maintained by a replica management service are needed both by the service administrators and by end users. Administrators need tools for monitoring and maintenance, while users may need to interact with the service, e.g., to find what datasets are available, or to explicitly create replicas.

These notes report about two graphical browsers for two replica catalog services deployed on the European Data Grid: `edg-rc-gui`, for the original Replica Catalog and `edg-rls-gui`, for the recently developed Replica Location Service.

2 The Replica Catalog Graphical User Interface

2.1 The EDG Replica Catalog

The *Replica Catalog* is a service that enables other Grid middleware or applications to find the physical location of *replicas* of files registered in the catalog. This service can be provided by a directory that maps the names of files to the (possibly replicated) physical instances of the same files. The replicas are identified by the hostnames of the storage elements holding them, a path to locate the replicas within the storage elements, and a protocol name and port number that specify how to retrieve the replicas.

The EDG Replica Catalog, based on the Globus Replica Catalog [6] is implemented as an LDAP directory [1]. The entries represent *logical collections* of file names, physical *locations* of replicas, and *logical file* entries that contain additional information about the files of a logical collection.

The information that identifies a location is given by its *URL constructor*, whose value is a string containing the protocol, the hostname, the port, and the path of the directory where the replicas are stored. If a storage element keeps replicas in more than one directory, or allows them to be retrieved through more than one protocol, that storage element will be registered in the catalog under multiple location entries, one for each directory and/or protocol. Note, however, that the definition of the URL constructor might change in the future, allowing for a more general and flexible mapping from logical names to replicas. Finally, each name of a file whose replica is in the location is a value of the location's `filename` attribute, and is equal to a name occurring in a logical collection.

Logical file entries have attributes for a file's size, for the file's creator and most recent modifier (both identified by their DN), and for creation and modification times. These last four attributes are admitted also in the other entries.

Grid applications can access the catalog through the Globus Replica Catalog C API, and a command line interface for end users is provided, the `globus-replica-catalog` command.

2.2 Requirements of the User Interface

This program allows *browsing* and *searching* of a Replica Catalog. By "browsing" we mean a top-down exploration aimed at finding the physical file names (PFN's) associated with logical file names (LFN's). LFN's are grouped into *logical collections* within a Replica Catalog. Browsing also involves finding such information as a file's size and modification times, stored in *logical file* entries. Sections 2.2.2-2.2.3 below refer to browsing operations.

By "searching" we mean the possibility of finding the logical collections or the Storage Elements containing files whose name match a given pattern; e.g., a user may type "*MuonHits*" in an input field and find the collections (or the locations) listing file names that contain the string "MuonHits". Section 2.2.4 below refers to search operations.

2.2.1 Selecting a Replica Catalog

A user selects the Replica Catalog to browse by entering the following contact information through a dialog window (Fig. 1):

- host nick: a user-supplied string used to refer to the RC;
- host name: name of the node hosting the RC, e.g., `testbed008.cern.ch`;
- LDAP port: port number of the LDAP server, e.g., 2010;
- RC dn: distinguished name of the RC, e.g.,

```
rc=TESTRC, dc=testbed008, dc=cern, dc=ch
```

A user can insert the contact information for a Replica Catalog into a private database. That Replica Catalog can afterwards be selected by supplying its nickname to the User Interface. Fig. 2 shows a simple interface to the database of Replica Catalog contact information, with a list of Replica Catalogs, and buttons to add new information and to connect to a selected Replica Catalog.

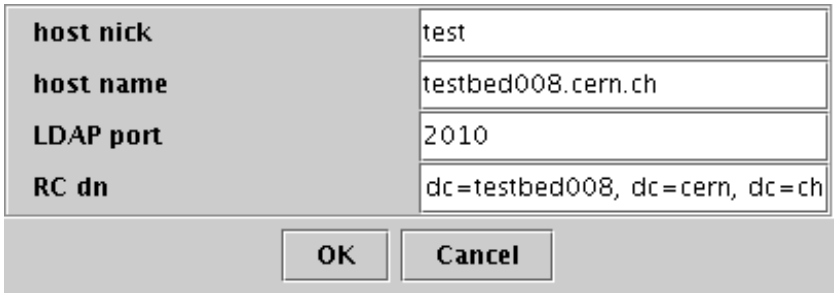

Figure 1: Entering Replica Catalog contact information.

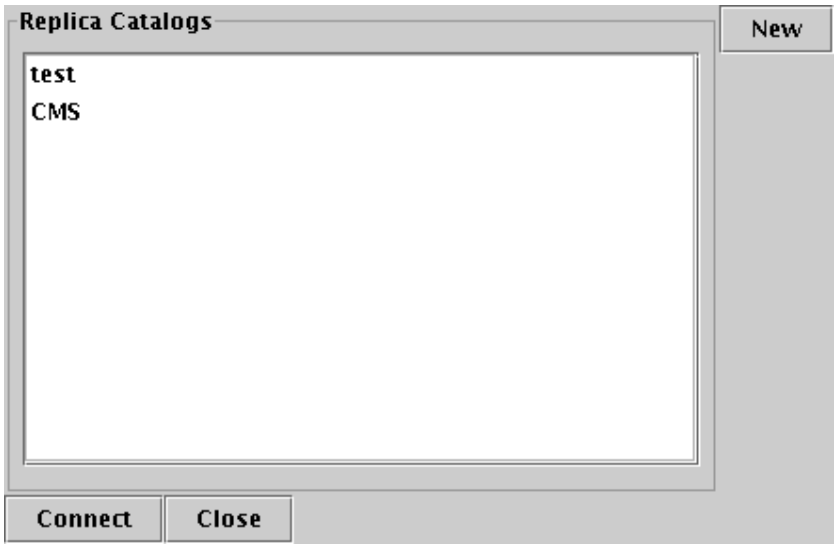

Figure 2: Managing stored contact information.

2.2.2 Displaying Logical and Physical File Names

After connection to a Replica Catalog is established, the available logical connections are listed by the User Interface (Fig. 3). The user can then select a collection and display the list of its Logical File Names (Fig. 4).

A user can select a Logical File Name and display the list of its associated Physical File Names (Fig. 5).

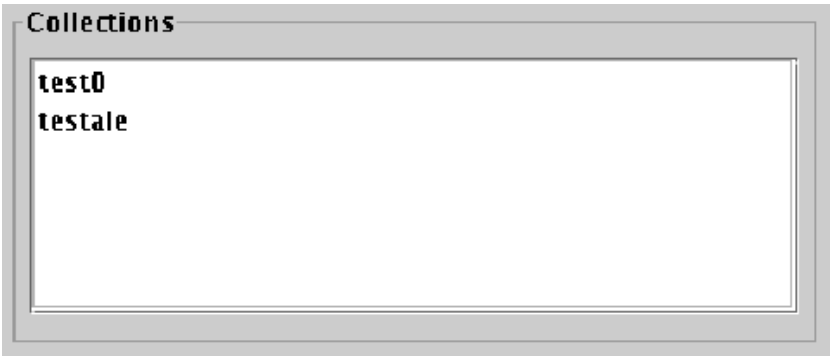

Figure 3: Listing logical collections.

2.2.3 Displaying Logical File attributes

A user can select a Logical File Name and display the attributes of its Logical File entry (Fig. 6).

2.2.4 Search operations

A user can enter a file name pattern and obtain a list of Collections or Storage Elements holding file names that match the pattern. A file name pattern is a string that may contain occurrences of the wildcard character '*'. Fig. 7 shows a search on Collections and Storage Elements.

2.3 Design of the User Interface

The User Interface for the EDG Replica Catalog has been designed under the following main constraints:

- The interface design should be flexible, so as to allow for new user requirements and new Replica Catalog schemas.
- The design should allow for easy replacement of the Replica Catalog implementation: for example, migrating from an LDAP directory to a relational database should need as little redesign of the User Interface as possible.
- The interface is to use the Java Swing graphical library.

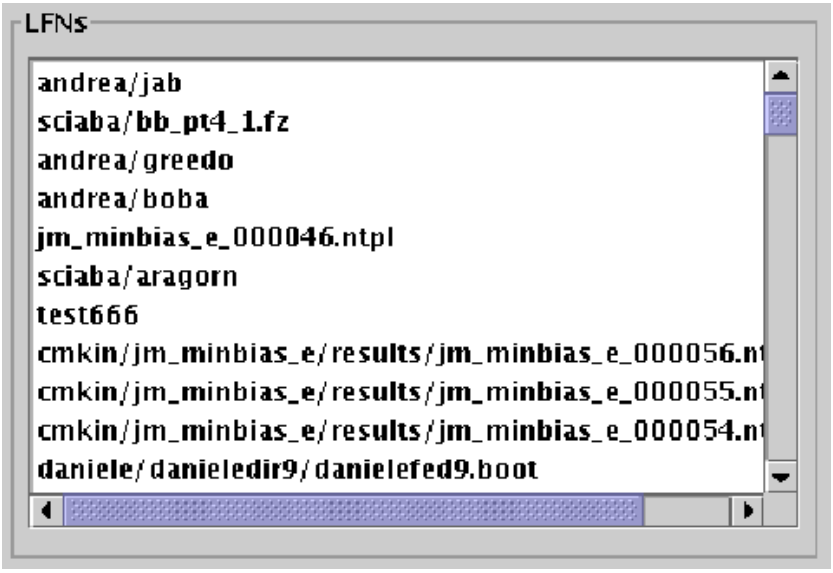

Figure 4: Listing Logical File Names.

Accordingly, the overall architecture can be summarized by the package diagram in Fig. 8, where Java and OpenLDAP stand for the Java libraries (including Swing) and the OpenLDAP client library, respectively.

Package EDGRepCat contains a single C++ class providing a set of operations tailored to the logical view of the EDG Replica Catalog. Package RCAdapter contains the Java class **RCAdapter**, which is similar to **EDGRepCat** but more oriented towards the functionality of the User Interface: for instance, instead of one query operation for each kind of Logical File attribute, as we have in **EDGRepCat**, **RCAdapter** offers a single operation that extracts the attributes from a Logical File entry and formats them so that they can be easily displayed by the User Interface. Finally, package RCGUI is the Graphical User Interface.

The coupling between the Java and the C++ components relies on the Java Native Interface (JNI). Package RCAdapter (Fig. 9) contains a JNI module where the native methods of the Java class **RCAdapter** are defined. The architecture therefore hinges on a simple variation of the well-known Adapter pattern [5], with the added constraint of allowing for classes implemented in different languages to be matched.

The highest-level components of the GUI package are shown in Fig. 10, together with their relationships with the Java Swing clases: most of them

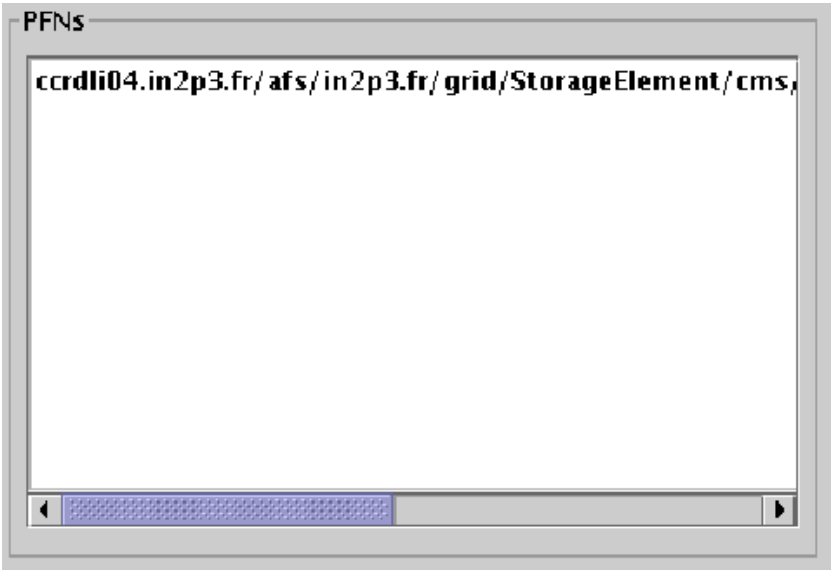

Figure 5: Listing Physical File Names.

are visible elements of the interface, such as the toolbar, the panes used to browse the catalog, and the dialog windows **SelnDialog** and **HostDialog**, used respectively to select a Replica Catalog from the database and to enter a new Replica Catalog. Further, **DefaultListModel** is a data structure shared among other components, **HostDB** manages the local database of known Replica catalogs, and **RCAdapter** is shown here since it is instantiated in the main application class, **RCInterface**.

A simplified view of the classes involved in browsing operations is shown in Fig. 11. Four instances of the **JPanel** class allow the user to display lists of Replica Catalog entries or attributes and to operate on their items, and the data displayed by each list is stored in an instance of the **DefaultListModel** class. User actions are processed by instances of classes derived from the abstract classes *ActionListenerImpl*, *ListSelectionListenerImpl*, and *MouseListenerImpl*, which use **RCAdapter** to access the Replica Catalog. We note that in the actual code the abstract classes have been optimized away in favor of a set of concrete classes that implement the respective interfaces directly.

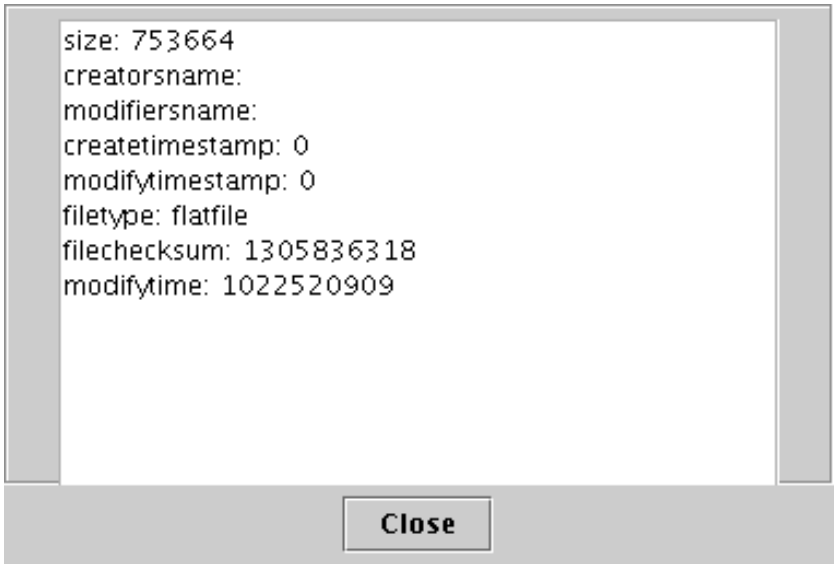

Figure 6: Listing Logical File attributes.

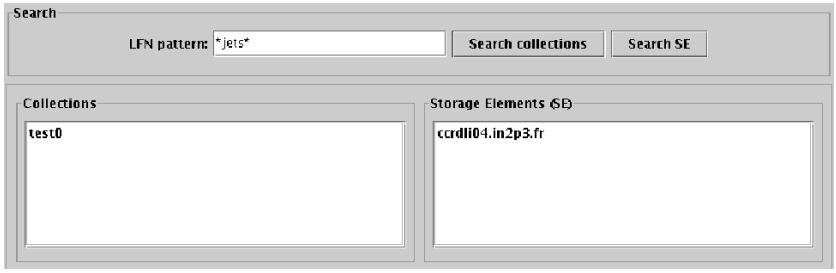

Figure 7: Searching for file names against a pattern.

3 The Replica Location Service Graphical User Interface

3.1 The EDG Replica Location Service

The Replica Location Service (RLS) [2] is a component of the Replica Management Service [4] that has the same purpose of the Replica Catalog, but has a flexible structure that better addresses the requirements of scalability and performance posed by a Grid environment, and is therefore expected to replace the Replica Catalog. Its main components are *Local Replica Catalogs*

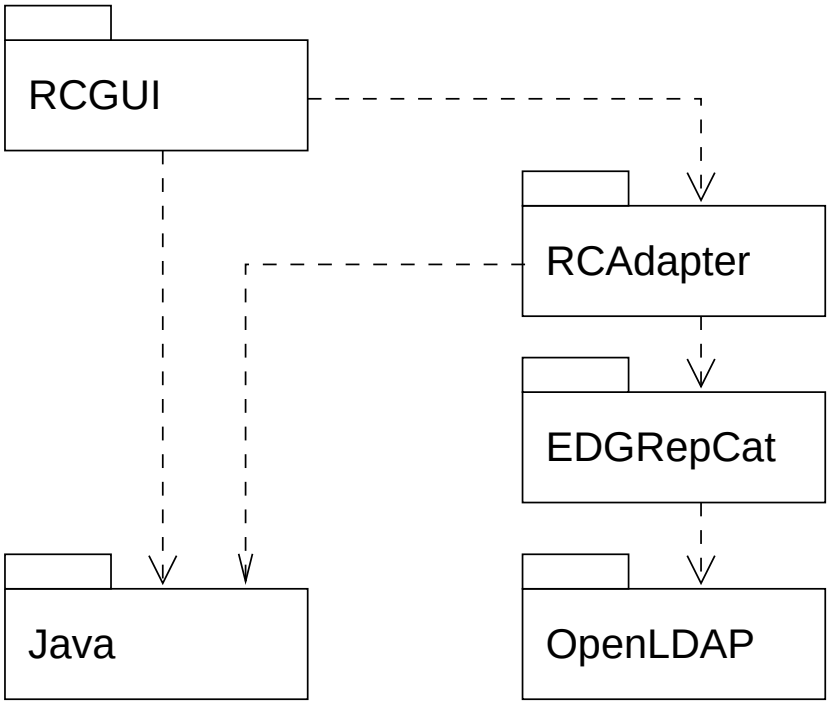

Figure 8: Architecture of the EDG Replica Catalog GUI.

(LRC) and *Replica Location Indices* (RLI): a LRC maps LFN's to the PFN's held at a single storage site, while a RLI maps a set of LFN's to a set of other RLS components, each being either a RLI or a LRC. Thus, a LRC has complete and up to date information on the replicas kept at the respective site, while a single RLI, in general, has information on a set of replica sites holding replicas for a set of the existing Logical File Names, perhaps indirectly, i.e., through other RLI's. Further, the information kept in a RLI may not be always up to date: a RLI receives periodical updates from the nodes it refers, and discards entries if they are not updated within a given timeout period. This mode of operation is referred to as *soft state*. As illustrated in [2], it is possible to obtain several RLS architectures by varying the number of RLI's, their mutual interconnections, and the partitioning criteria for LFN's and replica sites among indices and catalogs. Each architecture offers different tradeoffs between such requirements as performance, scalabilty, and reliability.

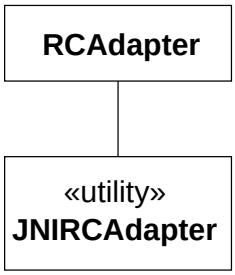

Figure 9: Package RCAdapter

3.2 Requirements of the User Interface

The Replica Location Service can be represented as a network of *nodes*, where each node is either Replica Location Index (RLI) or a Local Replica Catalogs (LRC), and the links between nodes represent references between indices and catalogs. A User Interface displays a hierarchical view of such a network. By selecting nodes of the network the user can browse the node's contents and limit the scope of search operations.

3.2.1 Selecting a Replica Location Service

A user selects a Replica Location Service by entering its URL in an input field. The URL has the form `rls://`*hostname*[`:`*port*].

3.2.2 Displaying service topology

A user can graphically display the topology of the network supporting a Replica Location service. The topology should be displayed as a set of tree structures.

Since the RLS architecture allows for several kinds of interconnection graphs to be established, in general a given topology can be represented by different sets of trees, according to which nodes are chosen as roots. The user can choose to display the topology from an LRC-centered view, where the LRC's are taken as tree roots, or from an RLI-centered view, where the RLI's are taken as roots.

The topology of a RLS can change dynamically, as nodes can be added or removed at any time. The user can therefore request the browser to update

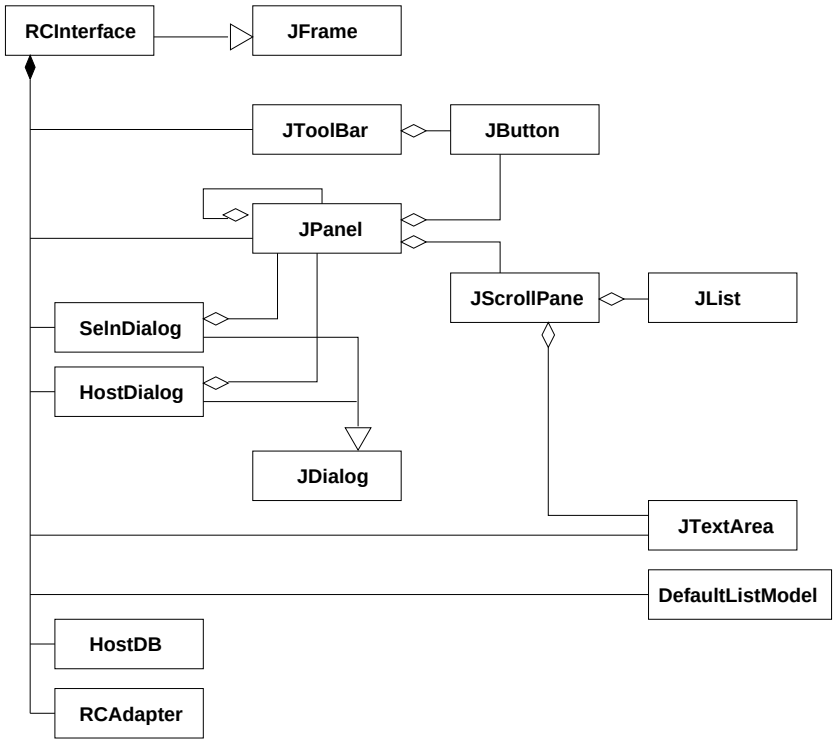

Figure 10: High-level classes of the GUI

the currently displayed topology.

3.2.3 Displaying node information

A user can display information about each node, including at least its *type*, i.e., whether it is a RLI or a LRC, and its URL.

3.2.4 Displaying node contents

A user can select a node and display the list of LFN's it maps.

3.2.5 Search operations

A user can enter a file name, specify whether it is a LFN or a PFN, and display the PFN's or LFN's, respectively, paired with that name in the mapping

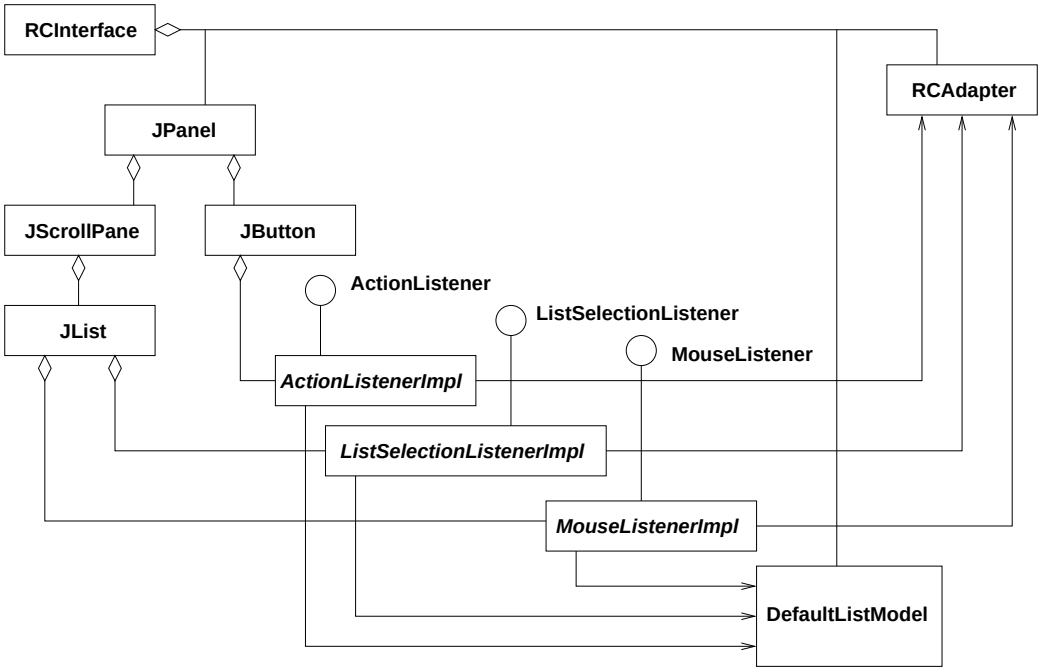

Figure 11: RC GUI classes for browsing operations.

maintained by the selected node.

3.3 Design of the User Interface

The main constraints on the design of User Interface for the Replica Location Service are similar to those on the Replica Catalog User Interface. Here, design flexibility and implemetation replaceability are particularly relevant, given that both the conceptual framework of the RLS and its programatic interface are in a state of constant evolution. The currently adopted interface to the service is the Globus RLS Client API, which is witten in C.

The overall architecture (Fig. 12) is therefore similar to the one adopted in the Replica Catalog User Interface: package RLSClient is a set of Java classes and interfaces that collectively act as an adapter towards the Globus RLS Client API.

Package RLSClient is more complex than RCAdapter (Sec. 2.3), given the more complex functionality and structure of the Replica Location Service. Figure 13 shows a simplified view of the package, omitting a few classes and

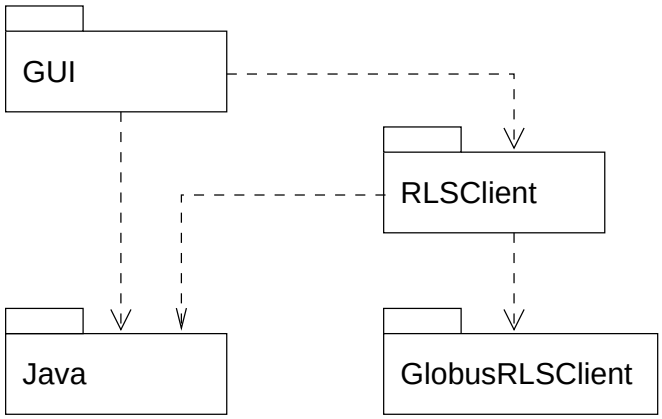

Figure 12: Architecture of the EDG Replica Location Service GUI.

listing only the operations currently used by the User Interface.

Interfaces **ReplicaLocationServer**, **ReplicaLocationIndex**, and **LocalReplicaCatalog** describe the corresponding entities in the Replica Location Service, and their implementations communicate with the service through a JNI module. An application instantiates a **ReplicaLocationServerFactory**, supplying it with the URL of the service, and creates a **ReplicaLocationServer** object through a factory operation. From this object it can obtain the service topology.

Interface **ServerTopology** defines operations that allow a client to explore the topology of a RLS. An implementation of this interface, such as **ServerTopologyImpl**, maintains a graph of nodes, each of them being a RLI or a LRC. The interface has operations that return **LocalReplicaCatalog** or **ReplicaLocationIndex** objects, or **ServerTopologyNode**'s: these are simplified representations of LRC and RLI nodes, holding summary information, such as node type and URL, that can be compactly displayed in a browser. The interface also has an operation that queries the service to update the topology.

Figure 14 shows the main classes from packages RLSGUI and RLSClient involved in browsing operations: classes **TopologyTreeBuilder** and **SearchTreeBuilder** access the RLSClient classes to obtain information that is then converted into the format required by the Java swing library for display.

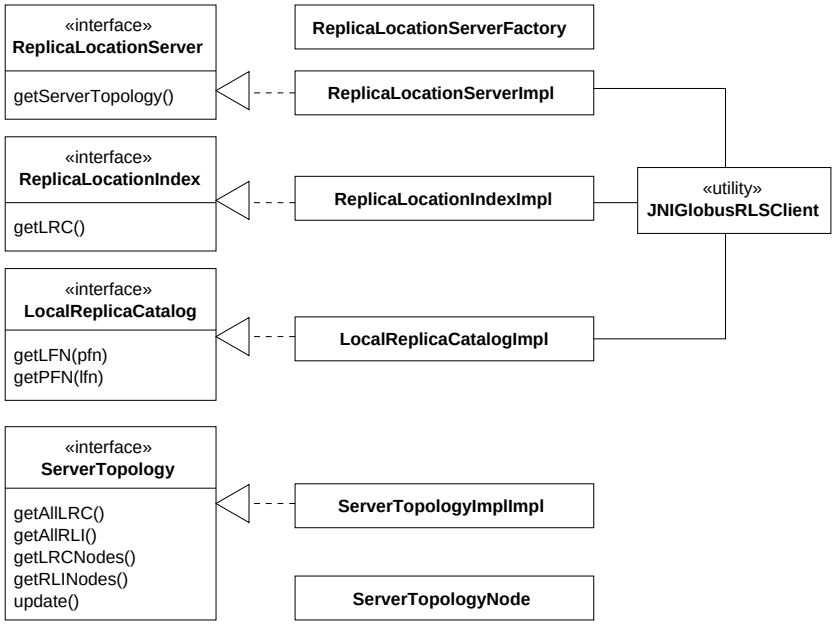

Figure 13: Package RLSClient.

4 Future directions

As mentioned before, the RLS browser is in a state of evolution, and tracks changes in the Globus RLS implementation. Three major additions, which will be added shortly are:

- An interface to the Metacomputing Directory Service (MDS) [3] in the System Topology object to find the set of RLS servers from an information server, as well as traversing the topology. The MDS provides Grid applications, as well other Grid services, with information on the structure and state of available resources.
- Detailed statistics information on each RLS, which can be published to the MDS, or obtained directly. This includes information such as number of mappings and attributes in the catalog, server statistics such as uptime and number of queries/inserts.
- A graphical browser, as well as the hierarchical browsers for Local Replica Catalog and Replica Location Indices.

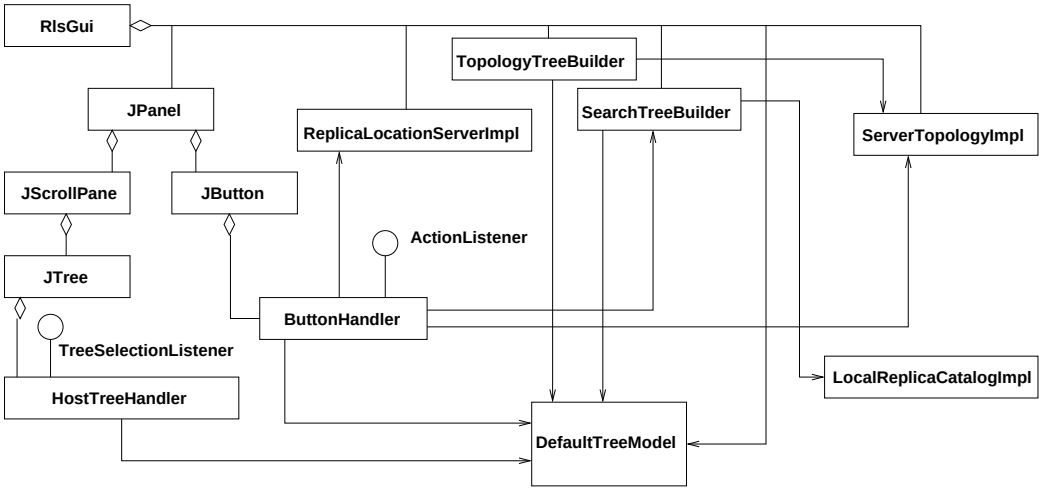

Figure 14: RLS GUI classes for browsing operations.

5 Acknowledgments

The work described in these notes is part of an international effort, the European Data Grid. The main developers of the browsers here described were Livio Salconi (currently at the European Gravitational Observatory) and the author. Marćin Kania and Arnaud Lacroix, contributed to the code for the RLS browser during their time at CERN, as a Technical Student and a Stageur, respectively.

References

[1] A. Domenici. Notes on the usage of an experimental Replica Catalog for the CERN DataGrid Testbed. Technical report, DataGrid WP2, 2001.

[2] A. Chervenak et al. Giggle: A framework for constructing scalable replica location services. In *Proceedings of the Int'l. IEEE/ACM Supercomputing Conference (SC 2002), Baltimore, USA*, 2002.

[3] K. Czajkowski et al. Grid information services for distributed resource sharing. In *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*. IEEE Press, 2001.

[4] L. Guy et al. Replica management in data grids. Working draft, Global Grid Forum (GGF4), Toronto, Canada, 2002.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[6] The Globus Project. Getting started with the Globus Replica Catalog. `http://www.globus.org/datagrid/deliverables/`.