Integrating RRS with Data Services in a Services Oriented Architecture

Functional Requirements Document

June 1, 2004

APEX DIGITAL SYSTEMS

Table of Contents

1. Introduction

The National Weather Service (NWS), through its Office of Hydrologic Development (OHD) has defined and documented a high-level software architecture design for upgrading and enhancing its operational forecasting systems. Now OHD intends to design the data services components that constitute a key part of the architecture, with the purpose of assessing the feasibility of such a design and establishing the scope of a proof-of-concept implementation of such components.

The proposed software architecture is focused on several key objectives, all of which revolve around the OHD's needs to modernize the NWSRFS (NWS River Forecasting System) scientific code, migrate to standard database systems, improve system performance with multi-threaded operation, implement more effective data locking capabilities, and enable greater configuration flexibility of the scientific systems.

The newly defined high-level software architecture focuses on services to address many of the modernization objectives discussed above. While previously the architecture referred to as services-oriented architecture (SOA) meant a services-oriented architecture built on XML standards and HTTP (XML, SOAP, WSDL, HTTP), we have expanded the term to include other architectures built on additional protocols and standards (JMS, JINI, CORBA, ODBC/JDBC).

In our document entitled "Software Architecture Engagement Summary" dated January 9th, 2004, we recommended the design of several core components that will drive the proposed SOA. During the course of writing the requirements for the components, we engaged in some design activities and defined the components and their interaction more clearly. As a result, we have renamed most of the components with more descriptive names, and have combined certain capabilities into one component where the more detailed design called for such a consolidation.

2. Task Objectives

According to the Statement of Objective (SOO), the purpose of this task is “to create a Data Services Design in accordance with the software architecture design described in the document *Software Architecture Engagement Summary* as delivered to OHD by Apex Digital Systems, Inc…. The Data Services Design resulting from this task must accurately describe the requirements and implementation specifications for development of a proof-of-concept Data Service using the NWS River Forecast System Rivers, Reservoirs and Snow (RRS) data pre-processor as an example application.” (OHD SOO for task No. 4-0009.)

Specifically, OHD requested requirements discovery focused around two capabilities:

A. Requirements related to upgrading RRS

Since OHD intends to create a working proof-of-concept data service that functions in the context of one NWSRFS application, OHD selected the RRS preprocessor (“Rivers, Reservoirs and Snow”) as an example integration point for a proof-of-concept data service. To achieve the example integration, OHD requested that Apex review the functionality of RRS’s interaction with its underlying data source. While the current task does not require requirements documentation related to implementing changes to RRS for integration with a future data service, we have outlined several relevant design issues below that will aid the understanding of the data services components requirements.

B. Requirements discovery related to future data services components

In our previous engagement, Apex defined a set of components deemed to be required for a flexible data service architecture that delivers the objectives described above. Under the current engagement, OHD requested that Apex define design requirements for the following components:

- Local Data Access Agent/Service: Local or remote application that routes read or write requests to the appropriate target repository.
- Data Location Agent/Service: Local or remote application that determines the location of a data provider registered for a specific data space identification code.
- Local Data Format Agent/Service: Local or remote application that formats data during read requests into a format specified by the calling application.
- Data Control Access Agent/Service: Local or remote application that accesses a specific data source for reading or writing and performs read or write requests.
- Directory Service: Local or remote application that provides data service identifiers based on a set of pre-defined parameters, such as a data space identification number.

We have conducted specific design activities to determine appropriate requirements for some of the components. We also discovered in the detailed analysis of RRS that the application is highly optimized to operate well within technical constraints present when RRS was created. These technical constraints, mainly related to the availability of memory, no longer exist in the same form, and in fact may inhibit an effective implementation of a services-oriented architecture surrounding the application. Before we discuss specific requirements for the listed components, it is important to address several general themes raised by the particular architecture underlying RRS.

3. Architectural Considerations for a Migration Strategy

The RRS program is an integral component to a critical national forecasting system. As a result, any efforts to modernize RRS, and other scientific applications like it, must occur while continuing to support production-level forecasting requirements. Any modernization efforts must produce a program that is at least as reliable and stable as the current RRS, while delivering key migration priorities. It is therefore imperative that OHD define a migration strategy that promotes stability and minimizes the risk of software failures while achieving significant architectural and technical improvements.

OHD will have a choice of several approaches to modifying RRS and similar applications to make use of the services-oriented architecture components proposed in this document, ranging from minimally invasive modifications to a complete reconstruction, in componentized fashion, of the code. Specifically, we have discussed with OHD the following migration options:

- Minimal impact: Rewrite functionality of read/write routines, and connect those revised routines directly to new data sources. While limited in scope, this approach will likely face major performance weaknesses as RRS accesses the current read and write routines very frequently. In the review by the Apex team, this option was discarded as not viable.

- Moderate impact: Define a standard API for each science application, and implement this API in an adapter-based code base. While this approach will likely require changes in the scientific code, those changes all serve to isolate the scientific code further from the database implementation, serving to stabilize the scientific code and enabling easier migration in the future.

- Significant impact: Rewrite science applications entirely in Java, C or C++. This approach has a major disadvantage in that it retains no existing code and requires a level of effort that is not viable for OHD at this time.

We recognize that the higher-impact solutions may be desirable, but conflict with the priorities stated above. Major Modifications present a high risk of introducing new errors in the code, and could degrade stability that has been achieved for key components. In addition, it is unlikely that NWS will have the resources available (people, time and funds) to achieve a major reconstruction of much of the NWSRFS code. Clearly, the selection of the appropriate migration option will remain a significant OHD policy discussion.

As a result, the Apex team proposes a program design that minimizes as much as possible the number of changes required to the scientific code and simplifies such changes as much as possible. The requirements presented in this document assume the moderate-impact approach, based on the adapter technology outlined below. This design approach reduces the risk of undermining production requirements and greatly increases the chances of migrating successfully and efficiently to more modern data sources. In addition, the design advocated in this document enables OHD to continue to modernize scientific code and data storage code safely and efficiently and prevents OHD from becoming locked into current database technology.

Key Migration Elements

Apex recommends a migration approach that will apply generally to NWSRFS applications that might be modernized in the future, and builds on several basic priorities:

- The design should allow incremental implementation.
- The design should enable performance optimizations for specific data sources.
- The design should enable future changes to RRS data sources with minimal code changes to the core interfaces.
- The design should promote stability and correctness while changing data sources.
- The design should enable data sources to be tested in isolation, prior to integration with scientific code.

Translating these priorities into a specific architectural structure is grounded in best software design practices on various platforms, we recommend that OHD consider implementing the following general components. Later in this document, we provide specific instances of these components, and provide use cases for their functionality.

- A defined **Data API** (Application Programming Interface), to provide scientific applications with a common method of interacting with their data.
- An extensible set of **Data Adapters** to implement the Data API in terms of specific databases. This enables almost any data source to be accessed from scientific code implemented in terms of the Data API.
- A set of **Data Adapter Factories** that are closely related to Data Adapters and are used to instantiate the specific driver needed by the scientific applications. This component loads a Data Adapter by name.

In this document, we outline in detail the requirements for the various components required to implement this architecture. We offer a piece of Java code to illustrate at a high level how these three components might interact, and how simple the interaction is intended to be:

```
RRSDataAdapter myAdapter;
int[] theObservations;

myAdapter=RRSAdapterFactory.GetAdapter("FS5","/users/…");

//remainder of scientific code…
theObservations = myAdapter.getObservations(stationID);

…
```

The Importance of Separating Science Code from Data Source Implementation

The main current concern regarding the RRS implementation is the fact that RRS very closely models the implementation of the database structures as part of the scientific code. Alternatively, the RRS Data API will enable the RRS scientific code to access RRS data in a database-neutral manner. The design of this API is critical to the success of the Data Adapter structure. Regardless of the level of representational specificity implemented in the API, the API must provide the following programmatic capabilities:

- Enable data access in a database-neutral manner; it must enable data queries to be made based on attributes of the data, rather than data storage attributes.

- Provide data in ways that is suited for effective use by the scientific code.

- Easily adapt to FORTRAN, C, C++, and Java; this means that data should be returned from data queries in types that are available in all these languages.

- Enable the Data Adapter to make performance decisions independently.

The API-based approach, coupled with the adapter technology described in further detailed below, enables the listed priorities. Figure 1 below provides a graphical representation of a possible implementation structure, showing that RRS scientific code uses a standard RRS Data API to interact with Data Adapters designed to interact with specific repositories. The diagram shows that the API remains consistent across adapters, while the implementation of the adapter's connection to the repository is specific to each adapter.

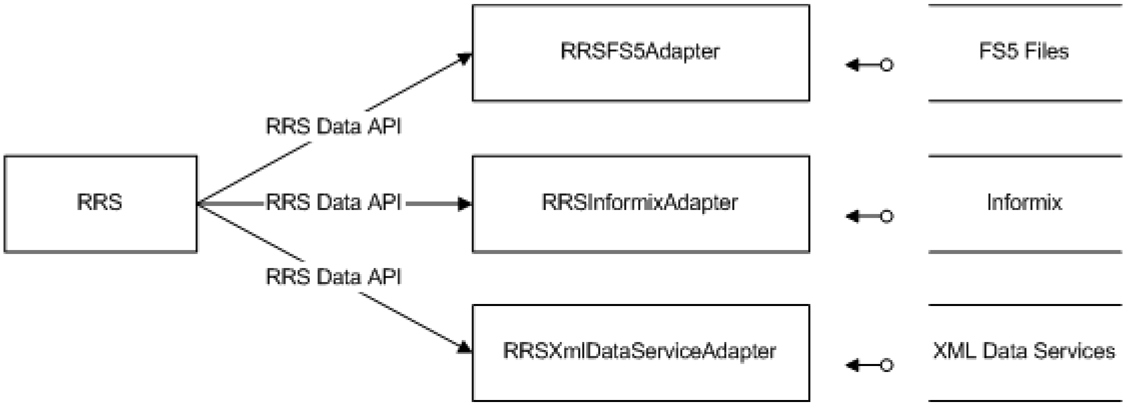

Figure 1: Conceptual RRS Data API

The RRS Data API could be implemented with varying degrees of specificity to the data representation, as follows:

- Full abstraction: Full data abstraction would require a general data representation language such as ANSI SQL. For accessing FS5 files, such a representation language would be onerous and would not provide much value. A sample call for data might have the following structure:

```
RRSDataAdapter myAdapter;
String mySQL;

mySQL="Select * from Observations where StationID='AB8JK73R0'";
myObservations = myAdapter.getData(mySQL);
…
```

- Full encapsulation: Full encapsulation would simplify the API and its implementation by providing specific-purpose interfaces. A sample call for data might have the following structure:

```
RRSDataAdapter myAdapter;
int[] myObservations;
```

```
String myStationID=”AB8JK73R0”;
myObservations= myAdapter.getAllObservations(myStationID);
…
```

At this point, the Apex team would recommend the latter approach, mostly due to its simplicity and stability. However, the approach should be tested further in the proof-of-concept phase of this project.

Integration with FORTRAN

The Data Adapter design cannot be implemented directly in FORTRAN. It can be implemented in C, C++, Java, and other languages. To integrate this design into RRS, a component is required to convert FORTRAN calls into calls native to the Data Adapter and Factory implementation. In this document, this component is described as the Science Application Interface (SAI). The SAI will expose all API methods with methods appropriate for FORTRAN. It will also expose methods that enable the calling application to identify the appropriate adapter based on application configuration settings.

4. Requirements Related to Upgrading RRS

In the course of analysis of RRS in its current implementation (version 72), we found RRS to be highly optimized for minimal availability of process memory. As a result, the scientific code is tightly integrated with the technical implementation of the underlying data repository, built in the proprietary FS5 technology. OHD will have a choice of several approaches to modifying RRS to make use of the data services architecture, ranging from minimally invasive modifications to a complete recreation, in componentized fashion, of the code. The moderate-impact approach, discussed above, establishes a challenging path to the solution. Any more significant modification to RRS is assumed to ease integration with newer technology such as the data services components presented in this document.

In order to begin to abstract RRS away from the specific database implementation, we are providing terminology and general use cases here. OHD will need to select a specific implementation approach for the proof-of-concept design effort.

Terminology/Domain Model for Data used in RRS

Station: A location of one or more meteorological or hydrologic data collection instruments. A station is represented by an 8-character identifier. A station typically contains more than on instrument, but only one instrument for each type of data.

Instrument: A device that collects hydrologic or meteorological data. An instrument is located at a station.

Data Type (Measurement Type): A categorization for measurements that have the same set of properties. Data types can refer to simulations as well as observations.

Observation: A single numeric measurement at a station for a specific point in time or time period. If the Observation applies to a time period, the observation has a start date and time as well as an end date and time. Each observation is of some Data Type.

Use Cases: Stations and Parameters

UC RRS-1.1: RRS will read Station parameters for several Stations and all parameter types.

RRS will retrieve from its repository a set of Stations, along with a list of all appropriate parameter types and parameter value arrays for each Station.

UC RRS-1.2: RRS will read Station parameters for one Station and all parameter types.

RRS will retrieve from its repository one Station, along with all the parameter types and parameter value arrays for the selected Station.

UC RRS-1.3: RRS will read Station parameters for one Station and selected parameter types.

RRS will retrieve from its repository one Station, along with the list of parameter values for a selected list of parameter types.

UC RRS-1.4: RRS will read Station parameters for several Stations and selected parameter types.

RRS will retrieve from its repository a set of Stations, along with the list of parameter values for a selected list of parameter types.

Use Cases: Observations

UC RRS-1.5: RRS will read Observations for one Station, all parameter types and all time periods.

RRS will retrieve from its repository a time-ordered list of Observations that were recorded during all available time periods for a specific Station.

UC RRS-1.6: RRS will read Observations for one Station, all parameter types and selected time periods.

RRS will retrieve from its repository a time-ordered list of Observations that were recorded during a selected time period for one Station for all Data Types.

UC RRS-1.7: RRS will read Observations for one Station, selected parameter and selected time periods.

RRS will retrieve from its repository a time-ordered list of Observations that were recorded during a selected time period for one Station for selected time Data Types.

UC RRS-1.8: RRS will read Observations for all Stations, all parameters and all time periods.

RRS will retrieve from its repository a time-ordered list of Observations that were recorded during all available time periods for all Stations for all Data Types.

UC RRS-1.9: RRS will read Observations for all Stations, all parameters and selected time periods.

RRS will retrieve from its repository a time-ordered list of Observations that were recorded during a selected time period for all Stations for all Data Types.

UC RRS-1.10: RRS will read Observations for all Stations, selected parameters and selected time periods.

RRS will retrieve from its repository a time-ordered list of Observations that were recorded during a selected time period for all Stations for selected Data Types.

Use Cases: Time Series

UC RRS-1.11: RRS will write time series data for a specific Station.

RRS will write to its repository time-ordered time series for a specific Station.

UC RRS-1.12: RRS will write time series data for various Stations.

RRS will write to its repository time-ordered time series for any number of Stations.

Use Cases: Lock Management

UC RRS-1.13: RRS will be able to lock a data source.

RRS will be able to exclusively lock a database. After RRS has requested a lock, the database is in the “Locked” state. While in the “Locked” state, all write operations to the database will return an exception, except when executed by the application that requested the lock. When locking a data source, RRS will indicate a timeout period after which the data source may unlock itself without an explicit unlock instruction.

UC RRS-1.14: RRS will be able to unlock a data source.

Only the RRS application instance that issued a specific lock will be able to unlock the data source, except when the unlock statement is issued after the timeout period expires.

UC RRS-1.15: A data source locked by RRS will unlock itself after the indicated time-out.

A database in the “Locked” state will return to “Unlocked” when the timeout period elapses. See section 12 for a further discussion of locking options.

Additional Considerations

The use cases listed above assume that only RRS-specific code will be addressed. For example, the new components will not process HCL in any way, nor will they provide the capability to manage system state for FORTRAN. Further, OHD will need to consider ways to manage RRS execution metadata, such as common block variable contents, as such metadata must be passed to the data services components explicitly. The appropriate approach for managing such metadata is yet unclear, and remains to be defined during the proof-of-concept phase.

Currently, user authentication in the context of NWSRFS is provided by operating system authentication – if a user has “execute” access to a directory, they have de-facto access to any application that may reside in that directory. With a services-oriented approach, authentication of callers to a service becomes an explicit requirement, as the user does not explicitly log on to the server or workstation that provides the services interface. Several options of providing authentication services should be considered, ranging from code-level authentication, where executing code must present a certificate to a service prior to exchanging data with it, to user-level authentication requiring application logons from the user executing an application.

5. Data Services Architecture Components

We have conducted specific design activities to determine appropriate requirements for some of the components, resulting in a slight reorganization and renaming of the components. As a result, this document discusses the following components:

Science Application Interface (SAI): The SAI enables existing FORTRAN code to access in a standardized fashion any new component to be integrated with the science algorithm, such as the new, services-oriented data services.

Data Location Interrogator (DLI), formerly called the Data Location Agent/Service: The DLI retrieves directory information regarding services-oriented data sources.

Data Service Adapter (DSA), formerly part of the Local Data Access Agent: The DSA connects to a data service of a specific type and specification, and exchanges data via a specific application-programming interface (API). The DSA implements a standard API which the SAI uses to make generalized calls to the data repository. Using this API, the SAI does not require any implementation-specific knowledge of the underlying repository. As a result, the SAI can use the same API for any data source implemented by a DSA.

Data Service Adapter Factory (DSAF), formerly part of the Local Data Access Agent: The DSAF ensures the correct setup and connection of data service adapters.

Data Access Controller (DAC), formerly called the Data Control Access Agent/Service: The DAC represents the data management code required to interact with a data source of a specific type, such as FS5, Informix, XML/HML, or others.

Directory Service (DS): The DS is a local or remote application that provides data service identifiers based on a set of pre-defined parameters, such as a data space identification number.

Lock Management Service (LMS): The LMS manages data source locking requests and distributes lock session tokens to requesting applications.

In addition, we have decided to integrate the Local Data Format Agent/Service into the Data Service Adapter, primarily because the Adapter will be designed specifically for each data type returned by the associated remote data source and must be able to return data fully formatted.

Further, the directory service definition remains unchanged, as is the overall concept of the data service interface, which is initially envisioned as a standard XML Web Services interface. The high-level diagram below shows the general relationships between the components. These relationships are described in more detail below. In the discussions below about each component, portions of this diagram will be repeated for clarity.

Terminology

Science Application: An application that contains scientific algorithms used to compute forecast data.

Data Space: An abstract group of data records or data tables in a repository that should be treated as one logical or transactional unit. In some cases, a data space identifier will point to an entire database, whereas in other cases such an identifier may only point to a specific set of records.

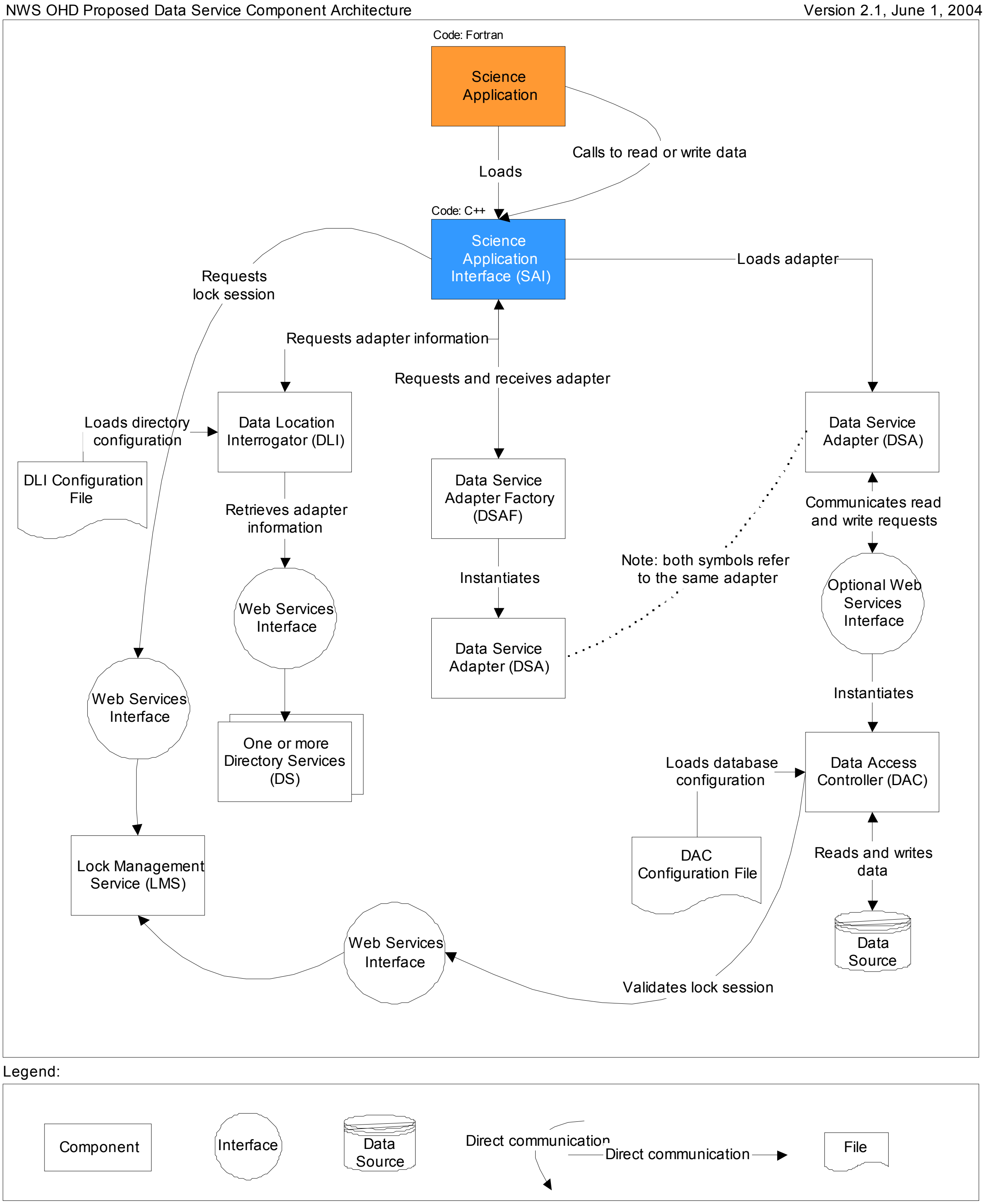
NWS OHD Proposed Data Service Component Architecture
Version 2.1, June 1, 2004
Code: Fortran
Science Application
Calls to read or write data
Loads
Code: C++
Science Application Interface (SAI)
Loads adapter
Requests lock session
Requests adapter information
Requests and receives adapter
Loads directory configuration
Data Location Interrogator (DLI)
Data Service Adapter (DSA)
DLI Configuration File
Data Service Adapter Factory (DSAF)
Communicates read and write requests
Retrieves adapter information
Note: both symbols refer to the same adapter
Instantiates
Web Services Interface
Optional Web Services Interface
Data Service Adapter (DSA)
Web Services Interface
Instantiates
One or more Directory Services (DS)
Loads database configuration
Data Access Controller (DAC)
Lock Management Service (LMS)
DAC Configuration File
Reads and writes data
Data Source
Web Services Interface
Validates lock session
Legend:
Component
Interface
Data Source
Direct communication
Direct communication
File

Figure 2: Data Services Components Overview

6. Component Requirements: Science Application Interface (SAI)

Overview

For existing applications, most of which are written in FORTRAN, to communicate with any new components, such as the envisioned data services components, they will require a small layer that serves as the conduit and integration point to code written in C or Java.

Use Cases

UC SAI-1.1: SAI provides a single integration point.

The SAI will serve as the single integration point for Science Applications with any data services to be used by the Science Application.

UC SAI-1.2: SAI invokes services components.

The SAI will invoke specific components to manage the flow of data between the Science Application into which the SAI is integrated, and the data services with which the Science Application must communicate.

UC SAI-1.3: SAI manages data outflow from the Science Application.

The SAI will receive data to be written to repositories from the Science Application and will present such data to the appropriate component for writing to the data source via the marshaled components.

UC SAI-1.4: SAI manages data inflow to the Science Application.

The SAI will receive data to be read from repositories by the Science Application and will present such data to the Science Application for algorithmic use.

UC SAI-1.5: SAI will manage DLI lookup failure.

The SAI will manage a failure by the DLI to resolve a data space identifier, and will communicate this failure in a standardized manner to the calling Science Application.

UC SAI-1.6: SAI manages lock session.

The SAI will exchange lock session data with the LMS required to ensure that connected data services only service requests from the SAI's Science Application.

The following diagram shows the SAI and its direct interactions with other components:

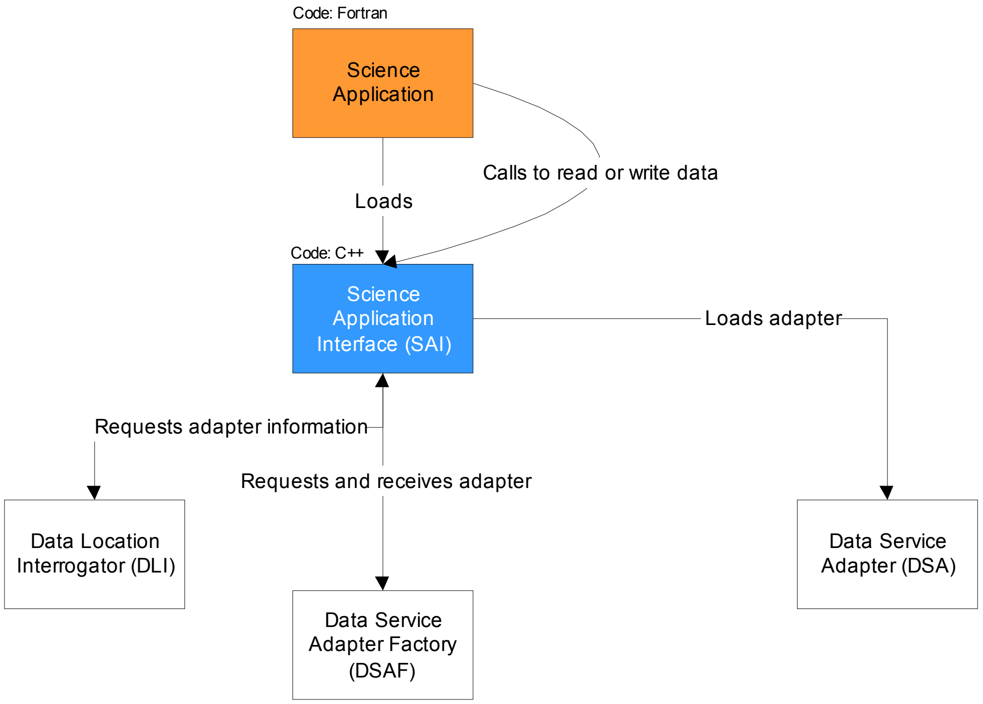
Code: Fortran
Science Application
Calls to read or write data
Loads
Code: C++
Science Application Interface (SAI)
Loads adapter
Requests adapter information
Requests and receives adapter
Data Location Interrogator (DLI)
Data Service Adapter Factory (DSAF)
Data Service Adapter (DSA)

Figure 3: Science Application Interface component interactions

7. Component Requirements: Data Location Interrogator (DLI)

Overview

The purpose of the DLI is to communicate with configured Directory Services to retrieve connection information for specific services via a data space identifier.

Use Cases

UC DLI-1.1: DLI is user-configurable.

The DLI will be locally configurable by a workstation user or administrator to determine all the Directory Services the DLI may contact to resolve a given data space identifier. Configuration will occur via a locally stored XML configuration file, which the DLI loads on startup.

UC DLI-1.2: DLI invoked by SAI.

The DLI will be invoked by the SAI, will receive from it the data space identifier, and will return to it a service address (IP address), Data Service Adapter type, and required connecting data.

UC DLI-1.3: DLI iterates over defined Directory Services.

The DLI will process data space identifier resolution in an iterative fashion, in the following manner:

- The DLI will contact the first configured Directory Service, requesting data space resolution.
- If the contacted Directory Service returns information that the Directory Service does not list the service, the DLI will contact the next configured Data Service and repeat the resolution request.
- The DLI will continue processing configured Directory Services until either the data space identifier is resolved, or the last configured Directory Service fails to resolve the identifier.

UC DLI-1.4: DLI manages lookup failure.

The DLI will manage a failure to determine a service address and connection data and communicate this failure in a standardized manner to the calling SAI.

The following diagram shows the DLI and its direct interactions with other components:

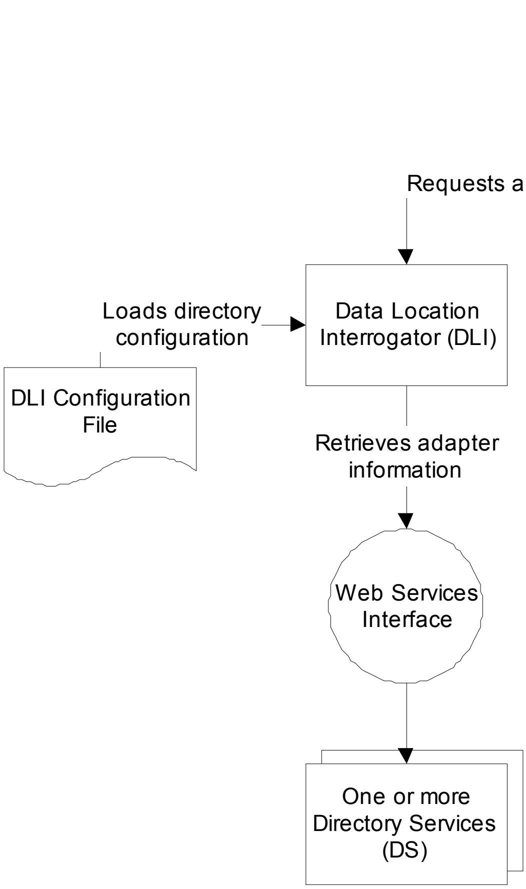
Science Application Interface (SAI)
Requests adapter information
Loads directory configuration
Data Location Interrogator (DLI)
DLI Configuration File
Retrieves adapter information
Web Services Interface
One or more Directory Services (DS)

Figure 4: Data Location Interrogator component interactions

8. Component Requirements: Data Service Adapter (DSA)

Overview

The purpose of the DSA is to create an abstraction layer between the calling SAI and the actual data service, regardless of the service type. As a result, the DSA implements a standard API that is specific to the Science Application's data, but which is also specific to a particular data service or source. For example, a data service implemented as an XML web service will require a separate adapter from a data service implemented as a JMS messaging-based data service. Further, adapters will also be used for access to locally stored files. This architecture ensures a single interface point for science algorithms to underlying data repositories, regardless of their location or type.

Use Cases

UC DSA-1.1: DSA will be instantiated by the DSAF.

The DSA is always instantiated by the DSAF, and receives specific connection information in order to connect to the correct data source. Once instantiated, the DSA is returned to the calling SAI.

UC DSA-1.2: One DSA will exist for each repository and access method.

Each combination of a repository and an access method requires a unique adapter. For example, for the SAI to connect to a FS5 database for RRS via XML web services, it requires access to a unique adapter. For the SAI to connect to the same database via JDBC, it requires a separate adapter. However, both adapters will implement a standard interface to RRS data.

UC DSA-1.3: All DSAs related to one particular application will implement the same generalized API.

Since the SAI must be able to access any data source regardless of its implementation, all adapters to be used by the SAI will implement the same standard API. In the case of RRS, the API will implement the functionality described by the RRS-related use cases above. The API will expose properties and methods to the SAI through which the SAI may read or write data from and to the data service.

UC DSA-1.4: A DSA will manage the connection to a DAC component.

A DSA, provided with the appropriate connection information, will monitor and manage the logical connection with the DAC via the required interface. The DSA will be able to detect connection failures, and will be able to communicate such a failure to the calling SAI.

UC DSA-1.5: A DSA will manage data exchange with a DAC component.

A DSA, provided with the appropriate connection information, will establish and manage a data exchange session with a DAC component via a specific interface (such as XML web services, JDBC, etc.) The DSA will execute method calls via the interface to the DAC and access properties, as needed, to read from the DAC or write to the DAC as required by calls from the SAI. The connection to the DAC may be a local connection, or it may connect through a remote interface or service.

UC DSA-1.6: A DSA will format data as required by the SAI and the DAC.

A DSA will format data passing through it appropriately for each consumer. If the SAI is reading data via the Adapter, the DSA will format data into structures required by the Science Application. If the SAI is writing to the Adapter and provides, e.g., array data, the DSA will transform the array data into the appropriate representation prior to calling the necessary methods and properties of the DAC.

UC DSA-1.7: A DSA will operate within a managed lock session.

A DSA will manage its connection to the DAC via the appropriate interface subject to the control of a managed lock session. If the SAI does not request an explicit lock management session, the DSA will request an implicit lock session, similar to implicit transactioning in relational database management systems. If a connection attempt occurs without valid explicit or implicit lock authentication, the DSA's connection attempt will fail. However, the DSA itself will not manage lock session information as such information will be managed by the SAI.

The following diagram shows the DSA and its direct interactions with other components:

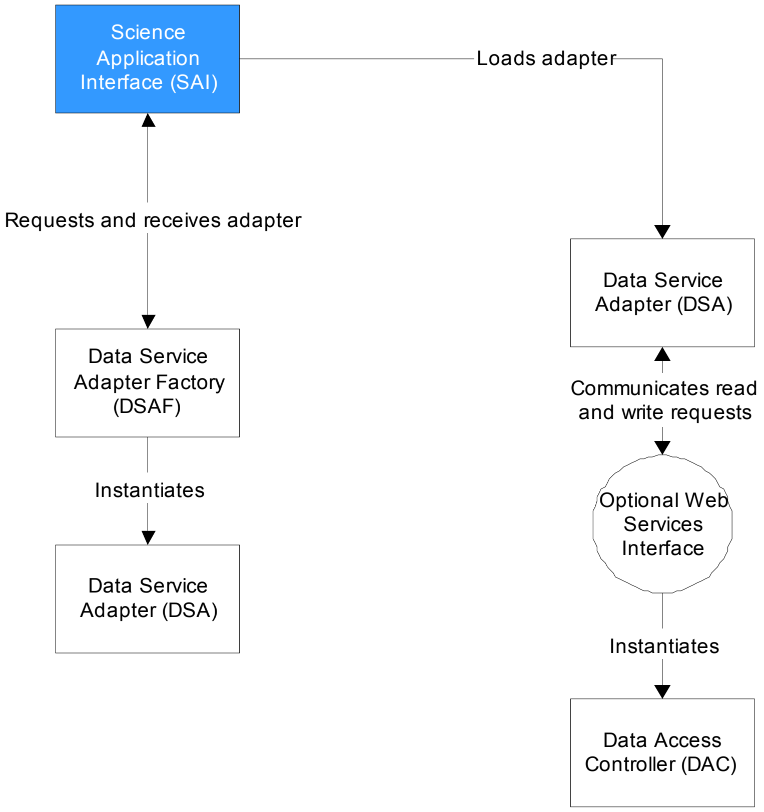
Science Application Interface (SAI)
Loads adapter
Requests and receives adapter
Data Service Adapter (DSA)
Data Service Adapter Factory (DSAF)
Communicates read and write requests
Optional Web Services Interface
Instantiates
Data Service Adapter (DSA)
Instantiates
Data Access Controller (DAC)

Figure 5: Data Service Adapter component interaction

9. Component Requirements: Data Service Adapter Factory (DSAF)

Overview

The purpose of the DSAF is to instantiate the appropriate Adapter with the connection settings provided by the SAI.

Use Cases

UC DSAF-1.1: DSAF will instantiate a specific Data Service Adapter to load.

From the information retrieved via the DLI from Directory Service, the DSAF will instantiate a Data Service Adapter of a specific type that is matched to the Data Service to be contacted. The DSAF will provide all necessary connection information to the Adapter.

UC DSAF-1.2: DSAF will return Data Service Adapter to the SAI.

If the Adapter instantiation succeeds, the DSAF will return the Adapter to the SAI, which in turn will use the adapter to contact the Data Service directly.

UC DSAF-1.3: DSAF will manage Adapter instantiation failure.

The DSAF will manage a failure to load a specified Data Service Adapter, and will communicate this failure in a standardized manner to the calling SAI.

The following diagram shows the DSAF and its direct interactions with other components:

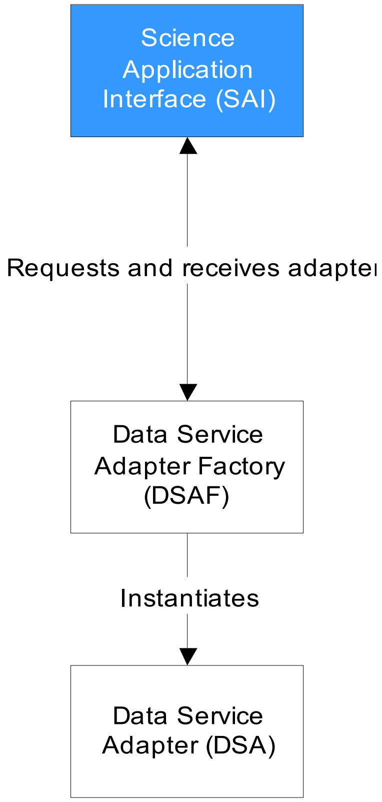
Science Application Interface (SAI)
Requests and receives adapte
Data Service Adapter Factory (DSAF)
Instantiates
Data Service Adapter (DSA)

Figure 6: Data Service Adapter Factory component interaction

10. Component Requirements: Data Access Controller (DAC)

Overview

The purpose of the DAC is to communicate directly with a specific data source and read data from the source or write data to the source. In many instances, the DAC will perform the routines now performed by the FORTRAN UREADT and UWRITT functions.

Use Cases

UC DAC-1.1: A DAC will be instantiated by a service or by a DSA.

If operating in a services-oriented mode, DACs will be instantiated by the service implementation (e.g., an XML web services running on a local or remote server). If an application is configured to use local data sources, a DAC will be instantiated directly by a specific adapter.

UC DAC-1.2: Each DAC will be associated with one repository.

Each DAC will be written to exchange data with one specific repository of a specific type and containing specific data. DACs will function as “custom” data exchange components closely fitted to the underlying data repository.

UC DAC-1.3: A DAC will directly access a repository using configuration data.

Since a DAC always operates directly on a repository, it will be configured to access such a repository as required. This may involve running local FORTRAN code to access FS5 data, using JDBC or ODBC to connect to Informix data, etc.

UC DAC-1.4: A DAC may receive read specifications from the DSA.

In order to read data from its associated repository, a DAC will receive a specific read specification, described in a generic language such as SQL, or defined by specific function calls with individual parameters. The DAC will be able to translate the request such that it can successfully retrieve the data described in the request.

UC DAC-1.5: A DAC may receive write specifications from the DSA.

In order to write data from its associated repository, a DAC will receive a specific write specification with the data to be written to the repository. This request will be described in a generic language such as SQL, or defined by specific function calls with individual parameters. The DAC will be able to translate the request such that it can successfully write the data described in the request.

UC DAC-1.6: A DAC will manage lock session information.

A DAC will be able to operate subject to the restrictions of lock sessions defined elsewhere. As a result, the DAC will only respond to requests that occur within a currently registered lock session.

UC DAC-1.7: A DAC will manage database errors.

A DAC will insulate calling Adapters from internal repository errors, and will expose standardized, structured exception information to the Adapter when exceptions occur.

UC DAC-1.8: A DAC is user-configurable.

The DAC will be locally configurable by a workstation user or administrator to determine the specific database connection required to access the database. Configuration will occur via a locally stored XML configuration file, which the DLI loads on startup.

The impact of this architecture on transactions in the science applications remains to be investigated in the future. While the data sources managed by DAC components will support transactions, the data space management capabilities will also have to provide such transaction management.

The following diagram shows the DAC and its direct interactions with other components:

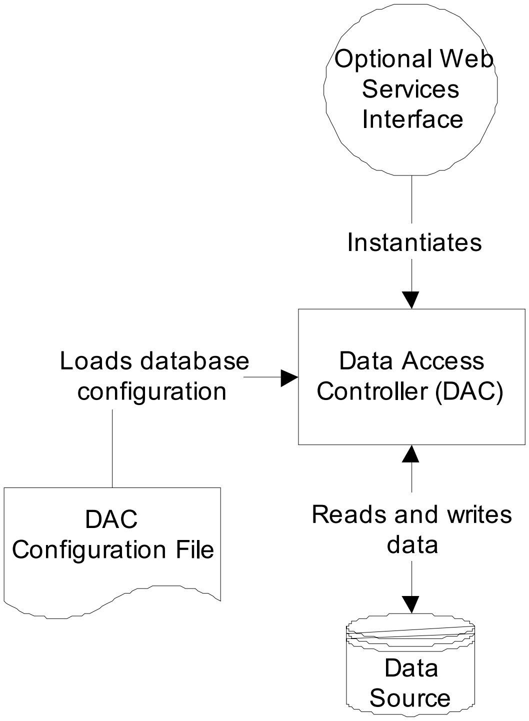
Optional Web Services Interface
Instantiates
Loads database configuration
Data Access Controller (DAC)
DAC Configuration File
Reads and writes data
Data Source

Figure 7: Data Access Controller component interaction

11. Component Requirements: Directory Service (DS)

Overview

The purpose of the Directory Service is to provide location and parametric information about specific services-based resources available to calling applications. A DS could be described as a global, specific-purpose data service in that it always serves resource location and connection parameter data via an XML web services protocol. In this way, it is similar to UDDI, but has a broader purpose.

Use Cases

UC DS-1.1: A DS will manage directory entries for data sources.

Directory entries will, at a minimum, consist of a data space identifier, an adapter type identifier, and connection parameters.

UC DS-1.2: A DS will receive requests to read directory entries from DLI instances.

A DS will listen, via an XML web services interface, to requests to read directory information from DLI instances. The DS will receive data space identifiers with each read request from a DLI.

UC DS-1.3: A DS will return directory information when possible.

When a DS matches a data space identifier with one entry in the directory repository, the DS will return the resource/service address, an adapter type identifier that indicates which adapter must be used to connect to the resource, and connection parameter information.

UC DS-1.4: A DS will return no-match-found information when appropriate.

If a DS cannot find a match for a data space identifier in its repository, the DS will return an exception to the calling DLI indicating that no match was found.

The following diagram shows the DS and its direct interactions with other components:

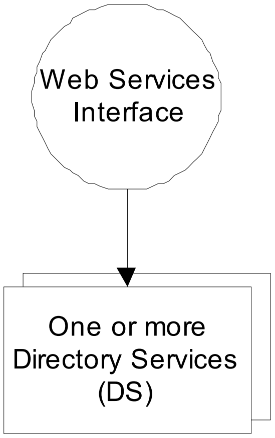

Figure 8: Data Service component interaction

12. Component Requirements: Lock Management Service (LMS)

Overview

Note: The Lock Management Service requirements and discussion are included for the purpose of comprehensiveness of the discussion in this document, but are not intended as Apex's final recommendation regarding locking. As stated in the proposal for the task for which this document is the final deliverable, lock management capabilities are complex and will require a separate discovery effort.

The purpose of the Lock Management Service is to manage data source locking requests effectively. We envision three options for managing locks, as follows. The specific implementation requirements remain to be determined once data space management capabilities are addressed in the future.

- Passive Lock Management: Locks are cleared based on pre-determined time-out periods
- Active Lock Management: Locks are cleared based on pre-determined time-out periods, but running processes may extend the pre-determined time-out periods at run-time. As a result, lock management achieves a higher degree of stability.
- Integrated Lock Management: This option requires deployment of a small software agent on all systems that execute applications requiring locks. When a lock approaches time-out, the LMS will be able to contact the agent on the application's host system and receive positive confirmation whether or not the application is still running. This option achieves the highest accuracy and stability, but also requires additional network traffic.

The use cases described here are based on Active Lock Management as described above.

Use Cases

UC LMS-1.1: The LMS will receive lock requests.

The LMS will receive locking requests that specify the application requesting the lock, the requested locking time, and a default lock timeout period.

UC LMS-1.2: The LMS will receive unlock requests.

The LMS will receive unlocking requests that specify the application requesting the unlock and the requested unlock time.

UC LMS-1.3: The LMS will receive lock period update requests.

The LMS will receive lock period update requests from applications that need to continue using a lock beyond the initially indicated timeout period.

UC LMS-1.4: The LMS will manage lock timeouts.

The LMS will automatically unlock data sources for which the LMS has not received an unlock request within the timeout period, and for which the LMS has not received a lock period update.

The following diagram shows the LMS and its direct interactions with other components:

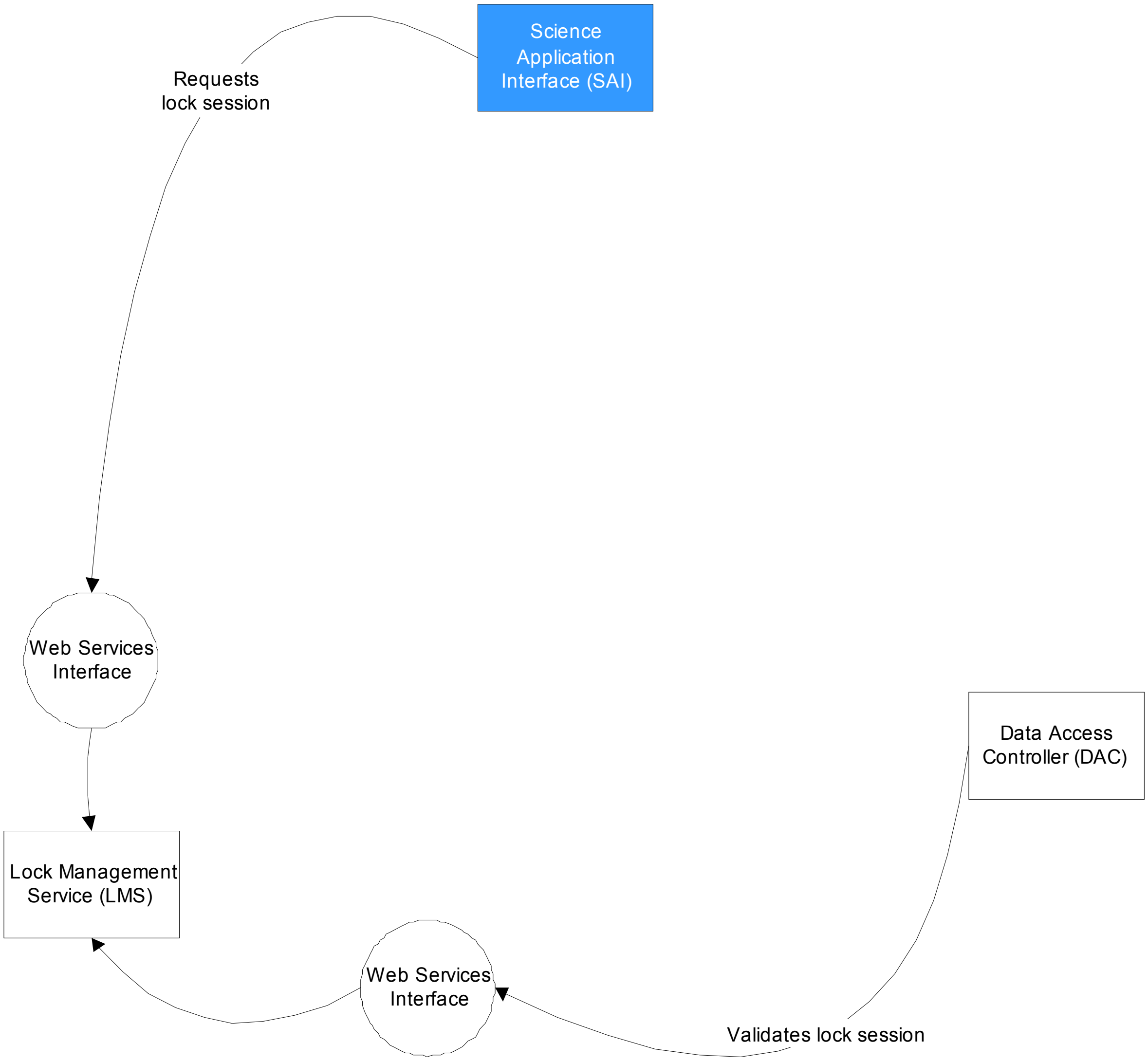
Science Application Interface (SAI)
Requests lock session
Web Services Interface
Lock Management Service (LMS)
Data Access Controller (DAC)
Web Services Interface
Validates lock session

Figure 9: Lock Management Service component interaction

13. Summary

The use cases and requirements identified in this document are intended to describe the capabilities required for a proof-of-concept data service implementation, using RRS as a test-case application for integrating existing, FORTRAN-based code with the new architecture. The requirements all point to a readily extensible implementation that is non-specific to RRS, but will support all of RRS's data exchange needs.

The requirements in this document serve multiple purposes. First and foremost, they are intended to articulate the need for a consistent technical migration strategy built around standard APIs. The purpose of the APIs is to abstract scientific operations clearly from the logical and physical implementation of underlying data repositories, enabling future enhancements or replacements of scientific code without a major impact on all of NWSRFS.

Second, the requirements are intended to outline a specific design pattern that is used widely in the software industry, as the best-practices approach to implementing the migration strategy. With the adapter-centric design pattern, OHD will be able to implement improvements, upgrades and replacements to parts of NWSRFS while maintaining system operations, working in small increments and iterations, and in easily testable code segments. The adapter pattern will be valuable especially in migrating applications from older data repositories to more modern ones without losing functionality.

Finally, the design advocated here is simple in a positive way: the components described here require careful and knowledgeable design, but have a relatively small footprint, which means that they can be adapted and used widely with moderate effort once the base components have been created. Component simplicity and stability may, in the final analysis, drive the success of this approach. The requirements outlined here, with the implicit design described, meet key requirements, ranging from organizational to technical, and should be implemented in a proof-of-concept fashion for validation.