T A R T U U N I V E R S I T Y
FACULTY OF MATHEMATICS AND INFORMATICS
Institute of Computer Science
Chair of Distributed Systems

Aleksei Nazarov

KnowARC Information System

Master thesis

Supervisor: prof. Eero Vainikko

Author: …… "….." September 2007
Supervisor: ……. "….." September 2007

TARTU 2007

Table of contents

Introduction

Scientists and researchers around the world constantly face problems that require more and more computational resources to solve them.

Leading corporations persistently compete with each other trying to develop the most powerful system. To achieve that aim companies increase the number of processors and their operating speed. But the creation of such multi-processor computers is very expensive. Cheaper solution is cluster systems that combine a number of less productive interconnected computers into a single computational resource. However, the capacity of multi-processor and cluster systems is still limited.

The alternative is grid systems. Theoretically, grid system can aggregate unlimited number of multiprocessor computers, cluster, ordinary personal computers etc.

So, by the grid we mean computational, storage or any other resources that are connected via network forming single resource that provides different type of service for its customers. Resources are usually widely geographically distributed, but to users it seems as a single system. Users should not be worried if he or she uses the resource that is placed in the next room or on another side of the globe.

Another rapidly developing field is Peer-to-peer (P2P) technology. Peer-to-peer networking model defines its members as equivalent nodes which don't rely on a centralized server. That is the main difference from the server-based networks, when part of the parties act as servers providing some services and others act as clients that use available services. In P2P networks members act as both client and server providing services to each other.

Recently there is a growing tendency to merge two technologies: grid systems and P2P networks. It means that grid developers are trying to create distributed grid system with the decentralized architecture that will be able to self-organize and would not have a single point of failure.

So we are also making an attempt to employ P2P approach to build grid.

Aim of the thesis

The aim of this work is to create new Information System for KnowARC [1] – Next Generation Grid middleware based on NorduGRID/ARC [2].

Acknowledgment

I would like to express my gratitude to my supervisor Eero Vainikko for his support. I want to thank people from the Lund University and especially Balázs Kónya for the productive collaboration.

Structure of the document

The first chapter describes the role and the mission of the Information System. Also we provide the description of old implementation.

The next chapters contains short introduction into JXTA platform. Here you can find the theoretical basics and the description of framework resources.

Third chapter describes the external interface of the Information Indexing Service.

In the fourth chapter we define the design of the new Information System.

Last chapter provides the implementation details.

Appendixes A and B contain Information Indexing Service schema file and WSDL definition file.

Appendix C provides the configuration file for eXist database.

Appendix D enumerates dependency libraries that are required to compile and run the system.

Besides, there is a CD with the source codes attached to the work.

1 Grid Information System

1.1 Role of Information System

The main tasks of an Information System of a grid are to collect and hold the valid information about available grid resource, and to provide that information to the clients during resource discovery.

An Information System must be scalable, robust, dynamic and fault resistant while serving these tasks.

Basically, Information System must be able to perform following activities:

- *Collecting and maintaining resource information* – the system must accept the information that describes the resource, process it, organize in some manner and store for a time while information is valid. Information system can be implemented relying on push or pull model.
- *Serving client queries* – client can execute different types of queries while doing resource discovery.

1.2 Old solution

Current implementation of the Information System (ARC version 0.6) is a LDAP-based system [3]. It uses customized OpenLDAP database that is a part of the Globus Monitoring and Discovery Services framework [4] provided by the Globus Alliance.

The system consists of three components:

- **Local Information Tree** – component is responsible for management and generation of the dynamic state information. The LIT provides the information that it holds for the requesting clients.

  The information in the LIT database is stored as attribute-value pairs grouped in entries which are organized into a hierarchical tree. The information stored in the database includes information about computing resources, grid jobs, users, storage elements and metadata catalogs.
- **Index Service** – the purpose of Index Service instances is to form coherent information system. Index Services are connected to each other organizing tree-like topology. The bottom level of the tree is represented by the Local Information Trees.
- **Registration Process Component** – during the execution, registration process links Local Information Trees and Index Service together. The registrations are performed

repeatedly: the information from the Local Information Trees and Index Services is sent to the parent Index Service.

In particular, activities taking place in the information system are:

- **Collection of the information into the Local Information Tree** – special programs, called information providers, collect local state information and push the data into the LIT.
- **Registration process** – as described above, this activity is implemented in the separate component that is responsible for registration.
- **Resource discovery** – during resource discovery the client walks through the tree of Index Services moving from an upper level to level above and collects the contacting information. Finally, the client reaches the Local Information Trees retrieving the actual resource describing information.

1.3 The need of a new Information System

Despite the fact that current LDAP-based implementation of the Information System has proven its reliability (the system is in use since May 2002 and the amount of aggregated resources is approaching 50 sites with the total number of CPU about 5000) the need of new Information System has become clear due to a number of reasons.

First of all, current Globus-patched OpenLDAP library is not supported by the vendor anymore. Hence, Information System should be migrated to a new platform implemented using technologies such as XML, Web Services etc.

Second, the external interface of the Information System needs to be standardized to ensure interoperability between different grid systems. Thus users of one grid system, for example gLite [5], could work with another grid, for example ARC, using the same client program without installing additional software.

2 JXTA

2.1 What is JXTA

JXTA is an open source project under the patronage of Sun Microsystems [6]. The term "JXTA" comes from the word "juxtapose" and means that peer-to-peer technology is a co-existing alternative to the client-service computing.

JXTA defines the common set of open protocols that define different aspects of P2P computing. Furthermore, JXTA also provides a number of language bindings – the implementations of protocol specifications that allow writing JXTA applications. The primary language binding is Java SE based API, but Java ME, C/C++ and other bindings are also available.

So, the main advantages of the JXTA are that the technology is independent from network addressing and physical protocols, and also independent from programming languages and platforms.

2.2 JXTA Components

JXTA defines a number of components and resources that may be used to build full-functional P2P systems. The core part of JXTA architecture includes basic elements that are common to P2P networking. The JXTA service layer allows creating flexible and advanced environment with the ability to implement and host various client applications (application layer) [7-10].

2.2.1 Peers

A peer is logical unit that is able to perform network operations and implements at least one JXTA protocol. Peers can be represented by various devices or applications.

To enable the communication peers must initiate one or more network addresses that can be used by JXTA protocols and advertise them as peer endpoints. Using endpoints peers can establish direct point-to-point connections to transmit data.

Peers in the JXTA network are divided in following categories:

- **Edge peers** – Simple peers that serve a single end user. Edge peer may implement minimal required set of JXTA protocols or may implement all of them.
- **Super peers** – Peers plays special role in building and deploying JXTA network. The super peer may implement one or combine several of following functions:

- *Relay* – allows to store and forward messages between peers when it is not possible to create direct connection because of firewalls or NAT.
- *Rendezvous* – manages global advertisements indexes that augment the efficiency of advertisements search. Also participates in propagation of messages.
- *Proxy* – is responsible for supporting minimal-edge peers that are not able to access to all JXTA network functionalities on their own.

2.2.2 Peer groups

A peer group is a set of peers that imply the common interest that grouped peers are agreed to serve.

- **Service based grouping** – group provides specific services that are available only to the members of the peer group.
- **Security based grouping** – group can establish specific security policies to limit access to the group's resources. Joining the group may require member authentication.
- **Information based grouping** – group allows its members to monitor the peer group's resources. The state and activities information is available to all members of the group.

Peers, basically, self-organize into peer groups. Peer group can define membership policies, it can be open or protected, and thus the peer interested to join the group must provide valid credentials. Furthermore, single peer can belong to more than one peer group simultaneously. Groups themselves organize hierarchical tree-like structure where each group has a single parent, except Net Peer Group that is on top of hierarchy. The Net Peer Group is default peer group for all peers in JXTA network.

2.2.3 Pipes

Peers utilize pipes to exchange messages. Pipe can be viewed as a virtual communication channel that connects one peer as a sender on one side and one or more receivers on the other side.

To use the pipe peer endpoint must be bound to pipe endpoints: input pipe for receiving and output pipe for sending. The pipe binding occurs dynamically. It is a requirement that pipe endpoints belong to the same peer group.

JXTA provides three pipe types:

- **Point-to-point** – connection is established between exactly two pipe endpoints, an input pipe on one peer to receive data and an output pipe on another pipe to send data.
- **Propagate** – propagate pipes have multiple input pipes. Messages are sent from the output pipe to every connected receiver.
- **Secure unicast** – the same as point-to-point pipe but also guarantees secure and reliable transmission.

2.2.4 Services

JXTA peers offer various network services to other P2P network members and use services of other peers. Service can be status monitoring, file transferring or any other "useful" functionality resided in the particular JXTA application.

JXTA defines two types of network services:

- **Peer Services** – service is accessible only at the peer that has published this service. If home peer of the service is dead or not reachable then the service cannot be invoked as well.
- **Peer Group Services** – service is present in the peer group with multiple instances running on different members of the peer group. Peer group service is available till there is at least one peer that provides the service in the group.

JXTA developers are able to write own customized peer and peer group services while JXTA provides the set of built-in peer group services. These include core obligatory services and optional services.

Following services must be implemented by every JXTA peer:

- **Endpoint Service** – service is responsible for message transmission between peers, it also implements basic part of the Endpoint Routing Protocol to enable message routing between JXTA peers.
- **Resolver Service** – service allows sending generic queries and handling responses. Services that require information exchange are usually built on top of Resolver Service.

The remaining services are optional; peer group implementation may or may not include these services:

- **Discovery Service** – service is used to obtain peer group resources, such as peers, peer groups, pipes and services.

- **Membership Service** – service allows a peer to provide its identity within a peer group. Peer identity may be used by services and applications to determine the requestor and check it against access rules.
- **Access Service** – service is used to validate peer requests. Requested peer can check the requesting peer credentials to determine if access is allowed.
- **Pipe Service** – service is used to create and manage JXTA pipes.
- **Monitoring Service** – service provides means to monitor state of other members of the peer group.

2.2.5 Modules

JXTA modules are general abstraction used to represent functional entity and define corresponding interface. Modules are used to implement services, network transports or any other loadable code compatible with JXTA.

Each module defines the way how peers can instantiate it. Module is represented by the interface specification while many different implementations of module specification can exist in the network. Modules are language and platform independent, the implementation of the same module specification can be written in Java, C/C++, .NET etc.

The module abstraction consists of three interconnected elements:

- **Module class** – is used to ensure the presence of the module in the network.
- **Module specification** – defines the way module can be accessed. Provides information necessary to invoke the module.
- **Module implementation** – the implementation of associated module specification. Multiple implementations may exist for the particular specification.

2.3 Advertisements

Every component or resource listed above is described with the help of advertisement – XML document that has particular structure defined by JXTA specification. When peer discovers JXTA resource it basically searches for the advertisement representing that resource. Found or discovered advertisements may be stored in local cache and used later without any networking.

Created and published advertisement contains presumable lifetime that defines when information becomes obsolete and should be discarded. An advertisement can be republished; in this case the lifetime is updated/extended.

Following advertisement types are defined in the JXTA specification:

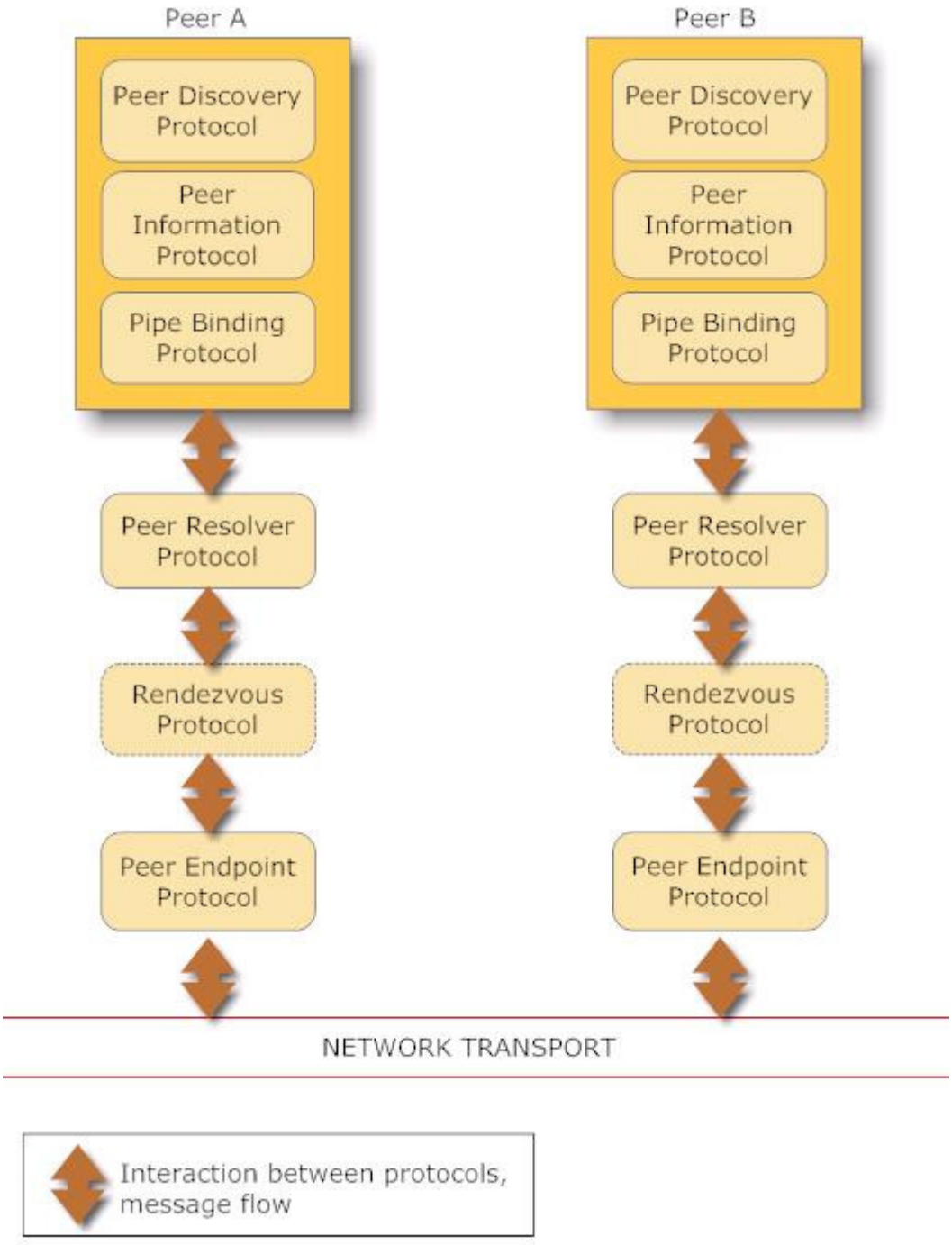
- **Peer Advertisement** – the peer's resources description. Contains peer specific information: peer name, peer ID, available endpoints etc.
- **Peer Group Advertisement** – contains peer-group attributes: name, peer group ID, description, specification, and service parameters.
- **Pipe Advertisement** – represents a pipe and is used to create input and output pipe endpoints. Pipe advertisement contains a unique pipe ID, type, and symbolic ID (optional).
- **Rendezvous Advertisement** – represents a rendezvous peer of a peer group.
- **Peer Info Advertisement** – provides peer state information: uptime, inbound and outbound message count, time last message received, and time last message sent.
- **Module Class Advertisement** – describes a module class. Consists of a name, description, and a unique ID.
- **Module Specification Advertisement** – defines a module specification. Advertisement may be used as a reference in case the implementation of specification need to be created or may provide information how to run implementation instance remotely with the help of a pipe. Advertisement includes name, description, unique ID (ModuleSpecID), pipe advertisement, and parameter field.
- **Module Implementation Advertisement** – an implementation definition of a corresponding module specification. Advertisement provides name, associated ModuleSpecID, code, package, and parameters necessary for execution.

2.4 Protocols

As it is described above the primary aim of JXTA is to define a set of protocols that enable to build P2P applications. The protocols allow performing dynamic discovery, data exchange, monitoring activities of parties, grouping peers and so on – the basic operations of every full-featured P2P application.

Basically, JXTA protocols are XML messages transmitted between peers. The sender is responsible for forming the message and sending it to remote peer. Remote peer receives message, processes it and sends response if necessary. The communication is asynchronous, zero or more responses can be given to a query in different times.

JXTA specification defines six protocols; each of them is responsible for one fundamental aspect of P2P networking. Besides, the protocol suite has layered structure: each protocol uses underlying protocol to pass messages.



Schema 1 JXTA protocols

- **Peer Discovery Protocol** – using this protocol peers can advertise their resources and discover resources from remote peers. Description of available resource is provided in the peer advertisement. A resource can be a peer, peer group, pipe, service etc. Thus, resource discovery means finding advertisement(s) holding information about corresponding resource.

Peer composes discovery query message to obtain advertisements within a peer group. This message is wrapped in resolver query message that is sent through peer resolver service. Query may not return any responses or return multiple results limited by the threshold value.

- **Peer Information Protocol** – defines a number of messages that can be used to obtain peer specific information such as uptime, state, traffic statistics etc.

  The ping message should be used to check presence of a peer and to get information about peer state. As a response to a ping message destination peer sends PeerInfo message that contains full description of responding peer. Peer Information Protocol as well as PDP uses Peer Resolver Protocol for message transmission.
- **Pipe Binding Protocol** – protocol can be used to create virtual communication channel, i.e. pipe, between peers. To use pipe, it must be bound to a peer endpoint. It allows creating an input pipe used to receive data or an output pipe to send data. Pipes are mainly used to transmit streamed data.

  The PBP query message is used to discover if pipe with the given identifier is bound. If receiver of the query has matching pipe in the cache it composes the PBP answer message that contains Peer Advertisement with the endpoint information included.
- **Peer Resolver Protocol** – provide means to send query to one or more remote peers and receive responses to the queries. One query can result in zero, one or multiple responses. The protocol may be used for any generic query that an application implements.

  The sending of the query requires of peer to create resolver query message that may be send to one or more remote peers. To process resolver query messages destination peer must register a handler. Handler is identified by name, the same name should be given in the message. Peer may compose and send resolver response message to a received query. Responses are also processed with the help of query handlers.
- **Rendezvous Protocol** – primary task of the protocol is to provide efficient methods to enable message propagation within a peer group.

  Peers utilize the protocol in different manner. On one hand, rendezvous peers with the help of PeerView protocol, which is part of RVP, organize themselves and share the distributed hash table address space, that allows to increase message propagation efficiency. On the other hand, client peers use the protocol to propagate messages and to subscribe themselves for receiving propagation messages.

- **Endpoint Routing Protocol** – protocol is responsible for routing messages to the destination peer(s). The "connection" between peers is not necessary direct, the message may pass through a number of intermediate peers (relay peers) before it is delivered to a recipient.

  While the message is conveyed it is supplied with the routing information that defines the path of the message from its source to the destination.

  To obtain route information to a destination peer requesting peer must create and send route query message. The resulting route response contains identifiers of source and destination peers as well as semi-ordered sequence of peer IDs that represent complete or partial route. The sequence may include several alternative routes.

3 Definition of Information Indexing Service (IIS) interface

The aim of this chapter is to define the WS-based interfaces of the Information Indexing Service.

A general ARC service interface consists of Service Specific Interface (SSI), the mandatory Local Information Description Interface (LIDI) and other optional interfaces such as the Delegation Interface. The SSI is responsible for offering the main service operations while the LIDI is used to present local information description. The optional Delegation Interface of services can be used to pass proxy credentials to the service. IIS implements only mandatory interfaces: SSI and LIDI.

***

The Information Indexing Service provides means to manage registrations of services and to discover services registered in the IIS. Registrations are managed via the service specific main interface of the IIS while the service discovery (queries about registered services) is served via the Local Information Description Interface of IIS. The registrations represent the activities of the Information Indexing Service. The collection of interacting Information Indexing Services, referred as the Information System, can be considered as a distributed store holding Registration Entries.

The service specific interface of the IIS provides the following operations to manage registrations of grid:

- *Register* – request registration by submitting registration message
- *RemoveRegistrations* – request to remove registration entries
- *GetRegistrationStatuses* – request the statuses of a set of registration entries

Registration entry is the main logical unit of information managed by the Information Indexing Services. A registration entry is an activity of the IIS. The Registration entry is a composition of Service Advertisement and its metadata. Service Advertisement is a subset of attributes from Full Service Description. Service Advertisement metadata is information about Service advertisements, e.g. recommended lifetime, origin. Each Registration Entry is uniquely identified by the Endpoint Reference of the service the entry corresponds to. Endpoint reference is mandatory element of Service Advertisement.

Registration message is a message accepted by the IIS during the registration process. Registration message consists of registration header element and the registration entries. The

header contains information about registration process. See the draft schema in Appendix A and WSLD in Appendix B for details.

3.1 Local Information Description Interface (LIDI)

Each ARC service interface contains a subset interface which is used to publish the complete local information, the *Full Service Description (FSD)*. The FSD describes the service, the resource behind the service, and activities managed by the service. This interface is called the Local Information Description Interface (LIDI). Initially we will use a subset of WS-ResourceProperties standard interface [11].

The FSD consists of the following information content, and will be made available through LIDI:

1. *Core service description*: common for all services and includes rather basic information about the name and access end-point of the service.
2. *Extended service description*: service specific information which includes uncommon special parameters about service itself.
3. *Service Status*: description of the service state like running, not running, overloaded etc. It can be part of both the core and extended.
4. *Resource description*: since only the services can represent resources in the grid environment the resource description contains the detailed description of the resource behind the service if there are any.
5. *Activities description*: in general, the services manage activities (in case of IIS the activity is resource registration information). This activity description should include states and attributes of activities managed by certain service.

The exact information model will be defined by service specific schemas following the emerging Glue-2.0 specification [12].

The access to the information content is based on a subset of OASIS Web Services Resource Framework (WSRF) [11]. The following operations of WS-Resource Properties specification were chosen to provide the necessary interface:

- *GetResourcePropertyDocument* – for obtaining all information at once
- *GetResourceProperty* – for obtaining specific sets of information
- *QueryResourceProperties* – for sophisticated selection of information. For query language ARC services should support `XPath`.

For specification details of the above three operations see WS-Resource Properties 1.2 document by OASIS [13].

3.2 IIS Service Specific Interface

The service specific interface of the Information Indexing Service provides means to manage the Registration Entries, to operate on the activities of IIS.

3.2.1 Common fault(s)

NotAuthorizedFault: This fault indicates that the client is not authorized to perform the operation.

InvalidInputFault: This fault indicates that given input has invalid format, i.e. it doesn't meet the schema definition. It is also possible that the input cannot be parsed and hence cannot be processed.

ServiceUnavailableFault: This fault indicates that the service is too busy or unavailable at the moment. Service cannot perform the operation.

ProcessingFault: This fault indicates that the operation failed because of an internal error. The fault provides an exact description of the occurred problem

3.2.2 Register operation

This operation is used to push registration information, i.e. Registration Entries, to the Information Indexing Service.

Input

RegistrationMessageType RegistrationMessage: An XML document containing registration information requested to be stored by the IIS.

A *RegistrationMessage* element consists of a single *Header* element that provides secondary information, like the identifier of the registration requester (the registrant), and a collection of *RegistrationEntry* elements. Each *RegistrationEntry* describes a single Service registration including Service Advertisement and the related metadata [see schema in Appendix A].

Output

None.

Fault(s)

No operation specific faults.

3.2.3 RemoveRegistrations operation

This operation is used to request removal of a collection of zero or more Registration Entries stored in the Information System.

Input

wsa:EndpointReferenceType[] RegistrationEntryID: A collection of zero or more Endpoint References, identifying Registration Entries the actor of the operation wants to remove.

Output

None.

Fault(s)

No operation specific faults.

3.2.4 GetRegistrationStatuses operation

This operation is used to request current statuses of zero or more Registration Entries previously pushed into system by the Register operation. The operation receives a collection of Endpoint References as its input and returns a collection of <*EndpointReference*, *RegistrationStatus*> and/or <*EndpointReference*, *Fault*> tuples.

Input

wsa:EndpointReferenceType[] RegistrationEntryID: A collection of zero or more Endpoint References, identifying Registration Entries whose status the requester wishes to obtain.

Output

GetRegistrationStatusesResponseType[] GetRegistrationStatusesResponse: An XML document containing a collection of *RegistrationStatus* elements for each Endpoint Reference given in the input.

The *RegistrationStatus* element represents the status of the *RegistrationEntry* identified by the Endpoint Reference provided in the input. The *RegistrationStatus* element consist of two child elements either the *EndpointReference* and the *Status* element or the *EndpointReference* and the *SOAP-1.1:fault* element. Thus the *RegistrationStatus* element consists of the pair of identification of the Registration entry and its status or fault in case of error.

Fault(s)

No operation specific faults.

4 System design description

In designing the Information System our goal was to distribute business logic among independent units. This goal was achieved and as a result we have created three independent but interacting with each other components. Each of them is intended to accomplish own portion of tasks.

The components are Information Indexing Service HED-Module, Information Indexing Service Main Component and Registration Store Component.

Also we consider some functional requirements that new Information System must satisfy. First of all, the implementation must be based on the Hosting Environment Daemon (HED) framework of KnowARC [14].

Additionally, the system must consist of collection of distributed independent instances that are forming the Information System Network. The topology of the network must ensure the absence of single point of failure which can break system's stability and efficiency. To achieve that goal we employ the P2P approach to build the network infrastructure.

4.1 Overview of components

New Information System consists of the following components:

- **Information Indexing Service HED-Module (IIS-HED)** – HED service that implements Service Specific Interface and Local Information Description Interface according to the specification [15].
- **Information Indexing Service Main Component (IIS-MAIN)** – component that implements business logic of the Information System. It is responsible for the formation of the network of IIS instances, sending and receiving messages within that network etc.
- **Registrations Store Component (RS)** – component is responsible for organization of the distributed store of registration information (Registration Entries). Single Registration Store Component instance represents Local Registration Store.

There are also components representing Information System consumers:

- **Registrant** – HED module that is responsible for registration of grid services to the Information System, which means pushing Registration Entries into Information System Store through IIS-SSI.
- **Query Component** – component that is used to query Registration Entries stored in the Information System. Query component uses IIS-LIDI to submit queries.

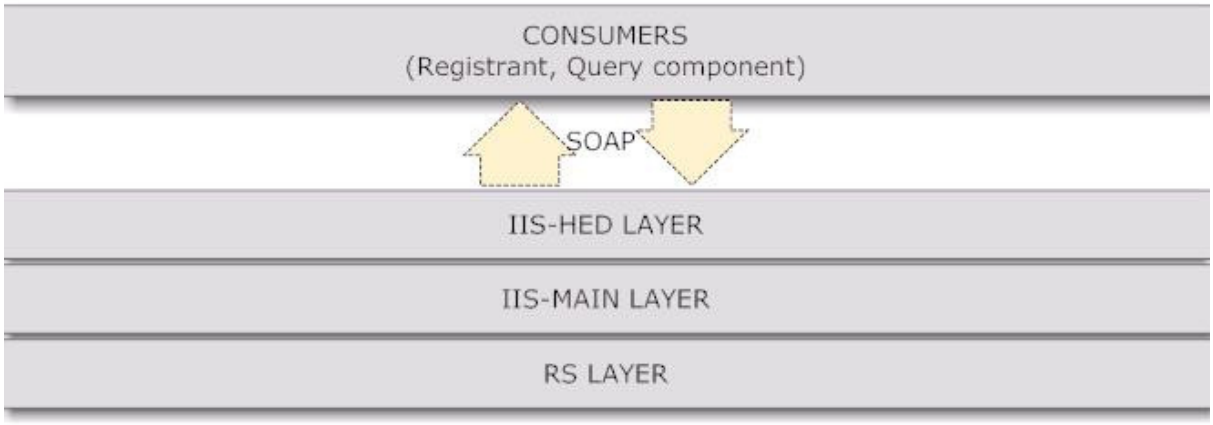
Detailed description of the components is given further in this document.

Registrant and Query component implementations are out of the scope of this work but we give description and some recommendation apropos of their implementation.

4.2 Structure and relationships

4.2.1 Information System

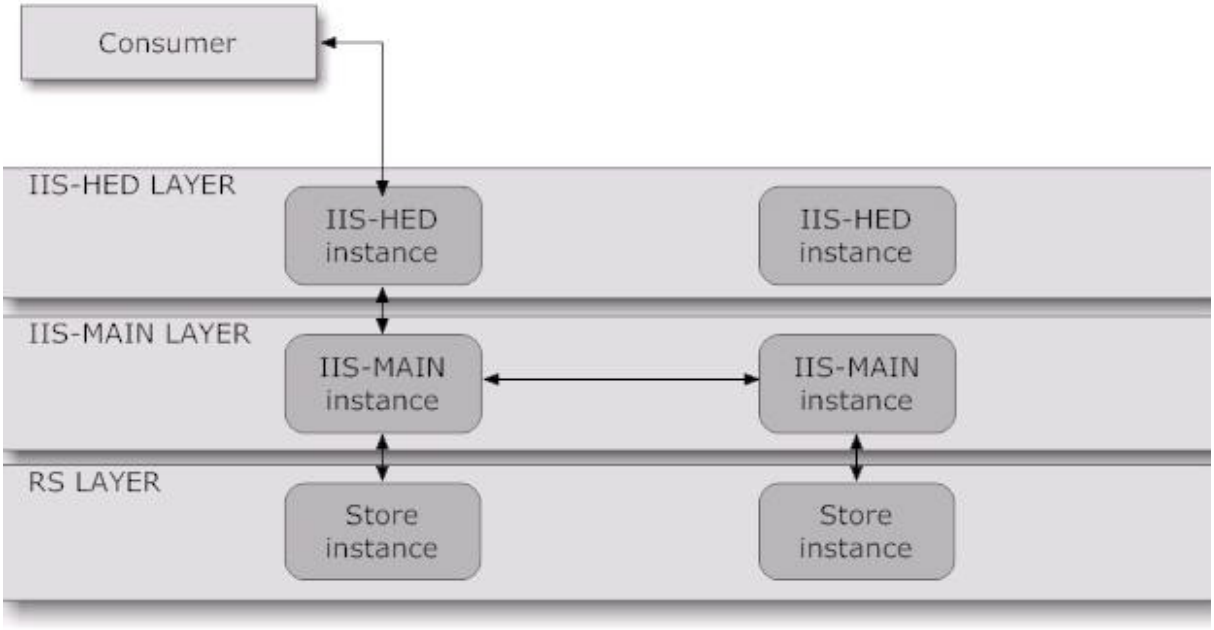
Information system has layered structure. Communication between different components or their different instances occurs within particular layer or involves adjoining layers.



Schema 2 Information System layers

- **IIS-HED Layer** – basically this is a external gateway for the Information System clients.
- **IIS-MAIN Layer** – this layer provides main business logic of the system, it is responsible for setup of the network of Information Indexing Service instances and implements the communication between them.
- **RS Layer** – layer provides the access to the data held by the system.

Diagram bellow illustrates the distribution of the components between different system layers and shows how the components interact with each other.



Schema 3 Interactions between components and component instances

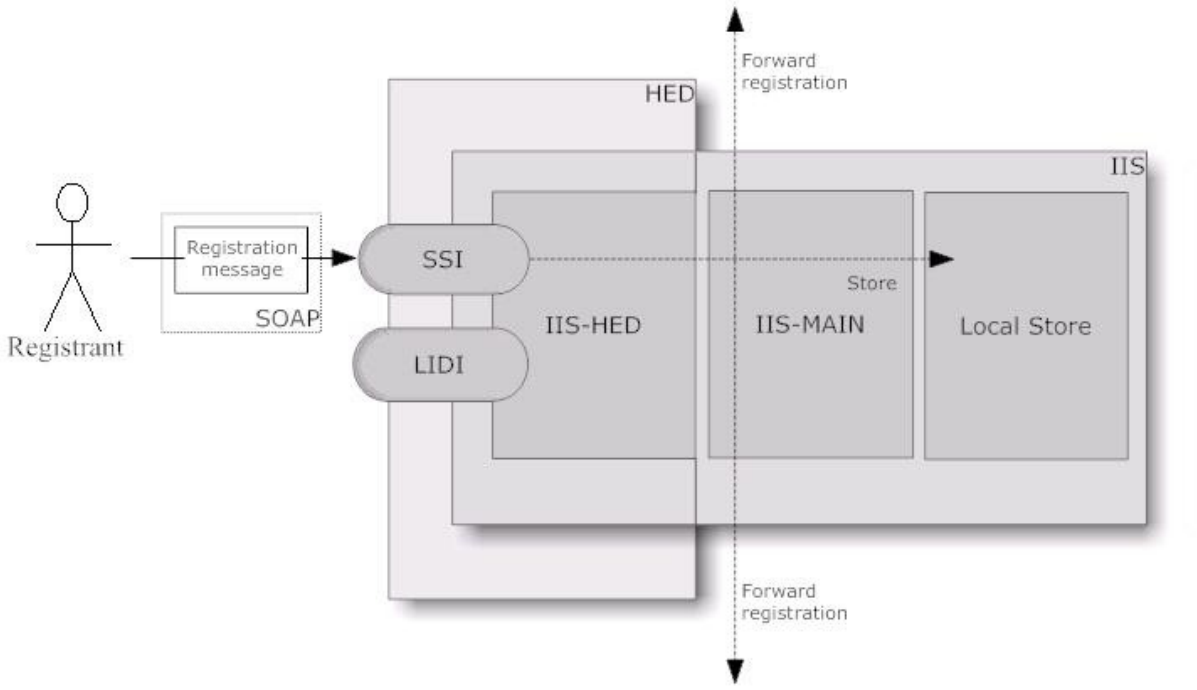
IIS-HED instance implements IIS-SSI and IIS-LIDI interfaces. It accepts requests from clients; each request is a SOAP message. Received message is parsed on that level obtaining method and input parameters. After parsing, method call is dispatched to the underlying IIS-MAIN layer. IIS-MAIN instance does further processing performing necessary actions. If processing requires data reading/writing then IIS-MAIN addresses to the RS instance (Registration Store Layer).

As it is also shown on the diagram different IIS-MAIN instances communicate with each other within IIS-MAIN layer. This communication takes place if request processing requires its propagation within IIS network.

4.2.2 Registration process

Registration of the grid services to the Information System means sending information describing grid services to the Information System. Received information is held by the Information System for a limited amount of time, expired information is discarded.

Registration process involves two parties, Registrant on one hand and single instance of IIS-HED component on the other hand. Following illustration enlightens the registration process.



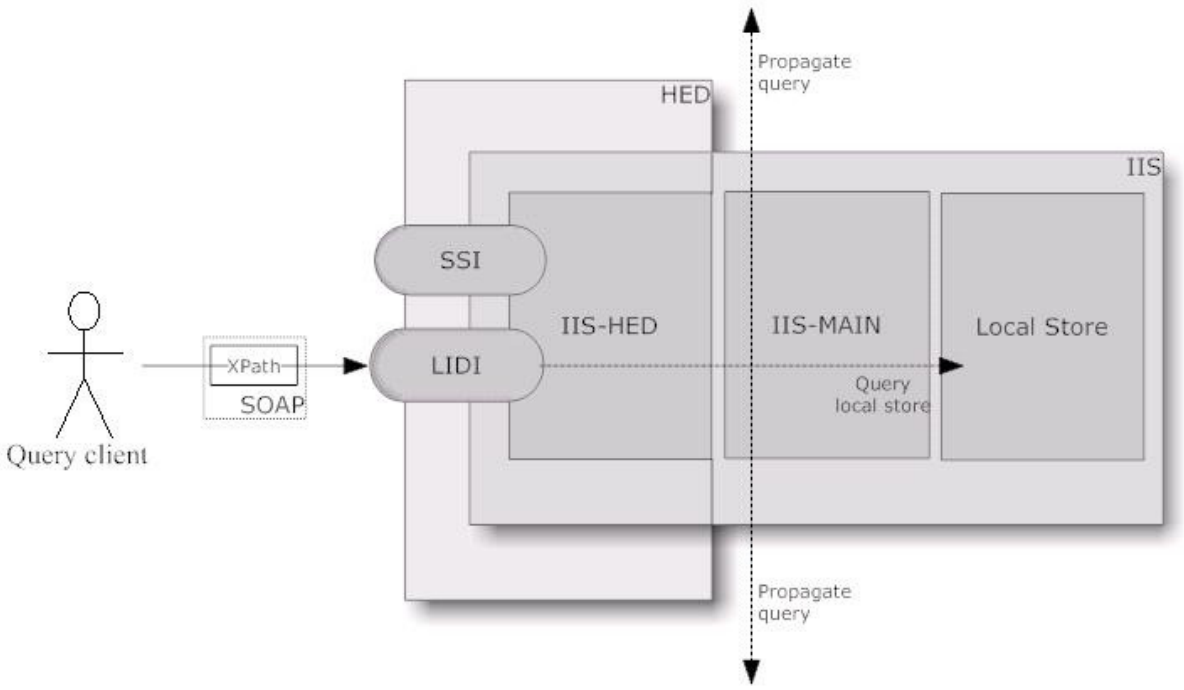
Schema 4 Registration process

Registrant must form Registration Message that contains registration information about grid services. Created Registration Message is delivered to the Information System over `SOAP`, particularly to a single IIS instance through IIS-SSI interface. Format of the Registration Message is described in the specification document.

Received information is stored in the Registration Store layer, in particular in the Local Registration Store.

4.2.3 Query process

Querying information from the Information System occurs through IIS-LIDI interface. Query client must compose proper `XPath` query and send it to the Information System, particularly to a single IIS instance, over `SOAP`. When clients create queries they must consider IIS XSD schema.



Schema 5 Query process

Received through IIS-LIDI interface request is processed by the IIS-HED. IIS-HED extracts `XPath` query from the received `SOAP` message and forwards method call to the IIS-MAIN. IIS-MAIN component executes the query and returns results that satisfy given query.

5 Description of the components

5.1 IIS-HED Component

5.1.1 Purpose and description

Information Indexing Service HED component is a HED-loadable module. It means IIS-HED is hosted by the HED environment and acts as every other HED service/module implemented according to the HED specification. For that purpose class representing the component implements special methods which format is defined by the specification. These methods are responsible for handling `SOAP` messages addressed to Information Indexing Service. For further information see the HED specification.

IIS-HED implements interfaces defined in the Chapter 3: Local Information Description Interface and Service Specific Interface, by that (implementation) we mean that processing methods handle `SOAP` messages that are specific for the interfaces referred above.

5.1.2 Function

As it has been described before the primary task of the component is accepting and parsing `SOAP` messages addressed to the IIS. Thus the functionality of the component consists of the following steps:

1. **accept** `SOAP` message given as an input.
2. **parse** input determining method to call and obtaining input values for the method.
3. **call** corresponding method of the underlying IIS main component for further processing.
4. compose **output** `SOAP` message by putting result(s) returned by the method into in.

In order to make all these steps possible IIS-HED also instantiates IIS-MAIN component.

5.1.3 Interfaces

`MCC_Status process(SOAPMessage inmsg, SOAPMessage outmsg)` – processes input `SOAP` message `inmsg` and creates output `SOAP` message `outmsg`. Function returns status of the processing.

See HED specification for further information [15].

5.2 IIS-MAIN Component

5.2.1 Purpose and description

Information Indexing Service Main component encapsulates business logic of the Information System while IIS-HED is only "gateway" to it.

IIS-MAIN component is responsible for building "Information System Network" that combines Information Indexing Service instances into single system so that they can communicate to each other within IIS-MAIN layer. In different cases this communication involves either all available IIS instances forming network or part of them. It depends on the operation that must be performed by the Information System.

To build such network infrastructure we employ one of the P2P solutions that is very promising and has a great potential. It is JXTA [10].

5.2.2 Function

When one of functions is called (except `getQueryResults`) IIS-MAIN creates message that is propagated over the network. Most of the operations also require re-propagation of message.

The query function deserves special attention. This operation is non-blocking and at the same time it returns collection of results. For that purpose function accepts listener as an input parameter. The listener is notified when a portion of new results is arrived. Then `getQueryResults` function is used to obtain query results. Each query has unique identifier.

5.2.3 Interface

`void store(String regMsg)` – extracts registration entries from the given registration message and stores them into the system.

`int query(String queryString, QueryStateListener queryStateListener)` – queries the Information System. The `queryString` is a `XPath` query against which the search is performed. The status of the search operation can be observed by the `queryStateListener` that is also accepted by the method.

`Collection getQueryResults(int queryID)` – this method can be used to obtain results of the query which has specified `queryID`.

`void remove(Collection eprs)` – this operation forces the removal of registration entries store in the Information System. The registration entries with identifiers provided in the `idCol` should be removed.

5.3 Registrations Store Component

5.3.1 Description and purpose

Registration Store Component's main purpose is to provide access to the persistence.

As long as primary data representation mean in Information System is XML and query language used by clients is `XPath` we select one of the XML native databases that is compliant with `XML:DB API` (developed by the XML:DB Initiative) [16].

5.3.2 Function

Component is responsible for creating new or opening existing Registration Store Database.

It also provides three functions to manipulate the data: save, remove and search underlying XML database. The save function saves given XML document into the database, remove function accepts identifier of registration entry that should be removed; search function executes given `XPath` query and returns corresponding results.

5.3.3 Interface

`void save(String id, String regEntry)` – saves given registration entry document into the database with the given `id`.

`Collection search(String queryString)` – searches the database for the registration entries that satisfy the given `queryString`. The `queryString` contains `XPath` query.

`void remove(String id)` – method removes registration entry with the given `id` from the registrations database.

`void shutdown()` – shutdowns the database.

5.4 Registrant

5.4.1 Purpose and description

Registrant is a HED loadable module that is responsible for sending registrations of services which are running in the same local HED. Registrant must collect information about locally running services. Then collected information is used to compile Registration Message. When the Registration Message is formed it is sent to the Information System using Service Specific Interface of the one of IIS instances

Registrant performs registration periodically depending on the lifetime of registration which is defined by the registering service itself.

5.5 Query component

5.5.1 Purpose and description

Query component is used by the Information System clients to query registration information stored in the system. Component must use Local Information Description Interface. It must create proper `XPath` query, compose request according to OASIS Web Service Resource Framework and send it over `SOAP`. For further information see the WSRF specification [13].

6 Implementation details

6.1 IIS-HED Component. Handling SOAP message

Component is responsible for parsing and processing `SOAP` messages. Message handing is done with the help of KnowARC API [17].

First of all we parse input `SOAP` message extracting the name of method and input parameters then we call corresponding method of the `IIService`. See register operation dispatching below.

```
XMLNode op = null;
if ((op = in_payload.Get("iis:Register")) != null) {
    String regMsg = op.Get("iis:RegistrationMessage").toString();
    try {
        service.store(regMsg);
    } catch (IOException ioe) {
        return new MCC_Status(StatusKind.UNKNOWN_SERVICE_ERROR);
    }
    ...
} else ...
```

6.2 IIS-MAIN Component

6.2.1 JXTA platform configuration

As long as Information System is JXTA application we need to initialize the JXTA platform and instantiate the Net Peer Group.

The Net Peer Group is the peer starting point. First thing the peer must do is to obtain Net Peer Group, only then peer can join other peer group which are basically the subgroups of the Net Peer Group. Thus the Net Peer Group is the top-level peer group that provides the connectivity to the network.

In addition, JXTA platform defines the World Peer Group. The difference of the World Peer Group is that it is just an abstract concept that serves the configuration means and is generally used as a template for the Net Peer Group.

It is not possible to create Net Peer Group instance directly. Developers must use `NetPeerGroupFactory` class instead:

```
NetPeerGroupFactory factory = new NetPeerGroupFactory();
netPeerGroup = factory.getInterface();
```

Code below creates `NetPeerGroupFactory` with the default configuration that is hard-coded in the JXTA framework. To create our private Net Peer Group we can configure the platform using `NetworkConfigurator` tool class.

Using this class JXTA application developer can specify peer identification information, security attributes, network description, peer transports etc.

When creating a private Net Peer Group the most important aspect is to specify the name and the ID of the group that are different from the default configuration.

```
NetPeerGroupFactory factory = new NetPeerGroupFactory(
                      (ConfigParams)configurator.getPlatformConfig(),
                      new File(jxtaHome).toURI(),
                      IDFactory.fromURI(new URI(NetPeerGroupID)),
                      NetPeerGroupName,
                      (XMLElement) StructuredDocumentFactory
                          .newStructuredDocument(MimeMediaType.XMLUTF8,
                                                 "desc",
                                                 NetPeerGroupDesc));

netPeerGroup = factory.getInterface();
```

In the application we configure peers to act in two different modes: as Edge peer and as Rendezvous peer.

The `NetworkConfiguration` also specifies the list of rendezvous seeds that is necessary for system "bootstrap". Moreover Edge peers are run in the auto-rendezvous mode, it means Edge peers can become Rendezvous peer if peer cannot find Rendezvous Service to connect to or there is a lack of Rendezvous peers in the network.

6.2.2 Network setup

Once we have configured and started JXTA platform, we obtain the instance of the Net Peer Group.

Next step we want to accomplish is to create custom peer group service - `IIService`. That requires us to create a number of advertisements and publish them. Finally, we also must change our peer group advertisement by putting service implementation advertisement into it.

For the `IIService` we must create three advertisements `ModuleClassAdvertisement`, `ModuleSpecAdvertisement` and `ModuleImplAdvertisement`. For that purpose we have three methods in `Utils` class:

- `public static ModuleClassAdvertisement createModuleClassAdv(String moduleClassID, String name, String description)` – method creates new

Module Class Advertisement with given attributes: the ID, the name and the description of new advertisement. Before setting these attributes we create the advertisement using the following method of AdvertisementFactory class:

```
ModuleClassAdvertisement moduleClassAdv =
  (ModuleClassAdvertisement) AdvertisementFactory.newAdvertisement(
    ModuleClassAdvertisement.getAdvertisementType());
```

Two remaining methods are very similar to the `createModuleClassAdv` method. The main difference is that we use different types when creating the object.

- **`public static`** `ModuleSpecAdvertisement createModuleSpecAdv(String moduleSpecID, String name, String description)` – method creates new Module Specification Advertisement with given attributes. The type of the advertisement must be `ModuleSpecAdvertisement.getAdvertisementType()`.
- **`public static`** `ModuleImplAdvertisement createModuleImplAdv(ModuleImplAdvertisement groupImpl, ModuleSpecID specID, String description, String code)` – method creates new Module Implementation Advertisement with given attributes. The type of the advertisement must be `ModuleImplAdvertisement.getAdvertisementType()`.

In addition to `IIService` advertisements we also must fix implementation advertisement of the service providing peer group. First, we create all purpose advertisement:

```
ModuleImplAdvertisement implAdv =
            netPeerGroup.getAllPurposePeerGroupImplAdvertisement();
```

Then we add module implementation advertisement of Registrations Store Service to the created peer group advertisement:

```
Utils.importService(implAdv, iisClassAdv.getModuleClassID(),
iisImplAdv);
```

After peer group advertisement is ready we publish it locally and remotely.

```
discovery.publish(adv);
discovery.remotePublish(adv);
```

Finally we create PeerGroup object:

```
arcGroup = netPeerGroup.newGroup(groupID, implAdv, GROUP_NAME,
GROUP_DESC);
```

When group is create it will have the ID that we have provided, and we also publish `PeerGroupAdvertisement`. Now local peer and remote peer can discover and use `IIService`.

All created JXTA resources need to have unique identifier. The IDs are generated during the development using `Utils.generateIDs()` method and after that we hold generated values as constants.

6.2.3 Building Peer Group Service

As a peer group service `IIServiceImpl` class implements `Service` interface. JXTA `Service` interface defines a number of methods:

- **`public`** `Advertisement getImplAdvertisement()` - returns implementation advertisement of this service
- **`public`** `Service getInterface()` - returns the service itself, we return this object
- **`public void`** `init(PeerGroup group, ID assignedID, Advertisement implAdv)` - initiates this service. In this method we just cache input parameters, peer group of the service, identifier of the service and its implementation advertisement, also we obtain the ID of local peer.
- **`public int`** `startApp(String[] args)` - starts the service. First we obtain JXTA service that our service uses – Resolver Service, we can say that `IIService` is built on top of the JXTA Resolver Service. Then we create and register query handlers. The role of the query handlers is discussed below in the document.
- **`public void`** `stopApp()` - method is used in order to stop the service. We perform clean-up: remove references to objects obtained in `startApp` method and unregister query handler.

`IIServiceImpl` class also implements external interface of the IIS-MAIN component – `IIService` interface. The implementation of each `IIService` interface operation is similar – the operation is executed locally and remotely.

Local execution of each operation implies the call of corresponding method of `RegStoreDAO` – Registration Store Component:

- local store operation executes `save` method of `RegStoreDAO`
- local remove operation executes `remove` method of `RegStoreDAO`
- local query operation executes `search` method of `RegStoreDAO`

By the remote execution of the operation we mean that we compose operation specific message and send it to remote peers. Peer that receives the message executes corresponding operation locally.

For example, to execute `store` operation remotely we create `StoreQuery` message where we put the collection of Registration Entries that need to be stored by remote peers. Created `StoreQuery` message is wrapped then by `ResolverQuery` message which is then sent to remote peers using Resolver Service:

```
StoreQuery msg = new StoreQuery();
msg.setRegEntryCol(regEntryCol);
ResolverQuery resolverMsg = new ResolverQuery();
resolverMsg.setHandlerName(StoreQueryHander.getHandlerName(assgndID));
resolverMsg.setCredential(null);
resolverMsg.setSrc(peerID);
resolverMsg.setQueryId(StoreQueryHander.ID());
resolverMsg.setQuery(msg.toString());

resolver.sendQuery(null, resolverMsg);
```

While sending `ResolverQuery` message we don't specify destination peer. That means the message will be sent to all discovered remote peers, i.e. so-called overlay network.

When `ResolverQuery` message is received by the remote peer, JXTA platform checks if there is a Query Handler that is responsible for handling received message.

Query Handler processes received message defined by the name that is provided in the massage itself. After Query Handler has determined message it is passed to the following method for processing:

```
public int processQuery(ResolverQueryMsg query)
```

Let us demonstrate query processing with an example of `store` operation. First of all, we need to extract Store Query message from wrapping it Resolver Query message.

```
XMLDocument asDoc = (XMLDocument)StructuredDocumentFactory
    .newStructuredDocument(MimeMediaType.XMLUTF8,
                           new StringReader(query.getQuery()));
storeQuery = new StoreQuery(asDoc);
```

`StoreQuery` is basically a bean representing the message. For all Registration Entries held by the `StoreQuery` object we obtain the ID of the entry and then store Registration Entry in the database.

```
XMLDocument asDoc = (XMLDocument)StructuredDocumentFactory
    .newStructuredDocument(MimeMediaType.XMLUTF8,
                           new StringReader(entry));
String id = (String)((Element)asDoc.getChildren("ID").nextElement())
    .getValue();

regStoreDAO.save(id, entry);
```

For store operation we stop propagation of the message, but for remove and query operations we re-propagate the message thus all the peers in the network receive it. To indicate that we want re-propagate the message method must return `ResolverService.Repropagate`.

You may have noticed that store and remove "remote" operations don't return any answer while in case of search operation we create Resolver Response message and send it to the source peer of the request. This resulting response contains the results on the query execution.

To handle search operation responses `SearchQueryHandler` class also implements the following method:

```
public void processResponse(ResolverResponseMsg response)
```

Processing of the response also starts with extraction of the content of the Resolver Response message. After we obtain the results of the search operation from the message we notify the listeners about arrived operation results:

```
SearchEvent event = new SearchEvent(response.getQueryId(),
                                    res.getResultCol());
```

To handle search operation events `IIServiceImpl` class implements `SearchListener`. On `SearchEvent` we cache results of the operation, the results are distinguished by the Resolver Query ID. Results are held in the cache until

```
public Collection getQueryResults(int queryID)
```

method is called, where `queryID` parameter identifies the query.

6.2.4 Cleaning expired entries

As we have mentioned above each Registration Entry stored in the Information System has two attributes which define the status and lifetime of the entry - the generation date-time and the expiration in milliseconds. To clean obsolete information from the database IIS-MAIN starts separate process that is responsible for cleaning up expired information.

Clean-up process is implemented using `Timer` and `TimerTask` classes provided in the Sun Java API.

First, we search for expired Registration Entries executing following `XPath` query:

```
String query =
"data(/iis:RegEntry/iis:ID[xs:dateTime(../iis:MetaSrcAdv/iis:GenTime)
+ xdt:dayTimeDuration(\"PT24H\") < current-dateTime()])";
```

Then we remove all found Registration Entries:

```
Collection col = DAO.search(query);
for (Iterator it = col.iterator(); it.hasNext();) {
    regStoreDAO.remove((String)it.next());
}
```

The clean-up task is run repeatedly, once in a predefined period.

6.3 Registrations Store Component

We use eXist database [18] in embedded access mode [19] that allows us to reduce efforts needed to maintain and administrate the database. eXist instance is set up and run in the same JVM as the IIS.

6.3.1 Database configuration

Configuration of the eXist database [20] used in Information System doesn't require any special setup or administration. We provide the configuration file with the source code of the system. You can also fine the `conf.xml` in the Appendix C. For the information about configuring eXist see the documentation.

On the first run we need to create the collection where we will hold registration information.

```
CollectionManagementService mgtService = (CollectionManagementService)
root.getService("CollectionManagementService", "1.0");
regStoreCollection = mgtService.createCollection(COLLECTION_NAME);
```

In addition we configure the indexes [21] for the created collection. We store indexes configuration file in the system collection:

```
<collection xmlns="http://exist-db.org/collection-config/1.0">
  <index xmlns:x="http://www.nordugrid.org">
    <fulltext default="none" attributes="false" alphanum="false">
      <include path="/RegMsg/RegEntry/SrcAdv/AuthzInfo"/>
      <include path="/RegMsg/RegEntry/SrcAdv/Type"/>
    </fulltext>
  </index>
</collection>
```

In case the collection already exists we just obtain the reference to the collection of registration entries – the information that we store in database, and use that reference to perform operations with the data.

```
regStoreCollection = DatabaseManager.getCollection(uri,
                                                    "admin",
                                                    null);
```

6.3.2 Saving XML documents

According to the `XML:DB API` in order to store a piece of XML data we must create `XMLResource` object that will wrap the data we want to store in the database. When creating `XMLResource` object we also must specify unique identifier for XML that we are trying to store.

The code that is responsible for storing XML data into the database is provided below.

```
Collection col = getRegStoreCollection();
XMLResource document = (XMLResource)col.createResource(id,
                                                       "XMLResource");
document.setContent(regEntry);
col.storeResource(document);
```

6.3.3 Executing XPath queries

Before executing `XPath` query we need to obtain special service that is responsible for `XPath` execution. The service is named `XPathQueryService`.

After `XPathQueryService` is acquired we use it to query the database. The result of this execution is `ResourceSet` object which contains `Resources` that satisfy given query.

```
Collection col = getRegStoreCollection();
XPathQueryService service = (XPathQueryService) col
    .getService("XPathQueryService", "1.0");
ResourceSet resultSet = service.query(queryString);
```

Abstract

Grids are collections of computational, storage or any other resources that are connected via network forming single resource that provides different type of service for its customers. Resources are usually widely geographically distributed, but to users it seems as a single system. Users should not be worried if he or she uses the resource that is placed in the next room or on another side of the globe.

One of the main components of each grid system is Information System. The task of Information System is to collect and hold the valid information about available grid resource, and to provide that information to the clients during resource discovery. An Information System must be scalable, robust, dynamic and fault resistant while serving these tasks.

The current implementation of the Information System of ARC middleware is LDAP-based. While it has proven its reliability a new Information System must be developed because of a number of reasons. Globus-patched OpenLDAP library, on top of which current system is built, is not supported by the vendor anymore. Additionaly, the external interface of the Information System needs to be standardized to ensure interoperability between different grid systems.

The Information Indexing Service is logic unit of the Information System. We define the external interface of the Information Indexing Service. The interface is defined according to the Web Service specification. The interface of IIS consists of Service Specific Interface (SSI) and Local Information Description Interface (LIDI). The SSI is responsible for offering the main service operations while the LIDI is used to provide access to local information description. We define the IIS-SSI providing the list of operation that service must support and also define data structures.

Designing new Information System we distinguish three component layers among which responsibilities are distributed: Information Indexing Service HED-Module (IIS-HED), Information Indexing Service Main Component (IIS-MAIN) and Registrations Store Component (RS). IIS-HED implements Service Specific Interface and Local Information Description Interface according to the specification. IIS-MAIN implements business logic of the Information System. RS is responsible for organization of the distributed store of registration information. We also define main system's consumers: Registrant and Query Component.

The Information System components are implemented in Java. The Information Indexing Service HED-Module is written according the the HED specification with the help of

KnowARC API. The Registration Store component serves as a gateway to the underlying database. We have choosen eXist XML native database as a data storage. The XML native database allows to store XML documents and execute XPath queries. eXist database is run in embedded access mode. While IIS-HED and Registration Store component are relatively simple the IIS-MAIN encapsulates all the business logic of the system, i.e. formation of the network of IIS instances, sending and receiving messages over this network.

The networking is the chief task that the component must accomplish. Implementation of this part of the component is done with the help of P2P technology. We build the system on top of JXTA platform. JXTA is an open source project under the patronage of Sun Microsystems. JXTA defines the common set of open protocols that define different aspects of P2P computing, also providing Java API allowing development of P2P applications.

As a result of the work new KnowARC Information System is developed. The system will be deployed and used in the next version of KnowARC middleware.

KnowARC'i Infosüsteem

Magistritöö

Aleksei Nazarov

Kokkuvõte

Grid on arvutus-, andmehoidmis- ja/või muu ressursside kogu. Ressurssid on ühendatud omavahel võrgu kaudu moodustades ühtse süsteemi, mis pakub klientidele mitmesuguseid teenuseid. Ressurssid on tavaliselt geograafiliselt hajutatud, kuid kasutajale pastavad need ühe süsteemina. Kasutajad ei pruugi teada kas kasutatav ressurss asub kõrvalruumis või maakera teisel poolel.

Üks tähtsamatest iga grid-süsteemi komponentidest on infosüsteem. Infosüsteem kogub ja hoiab kehtivat informatsiooni, mis kirjeldab kättesadavaid gridi-ressursse ning annab kogutud informatsiooni klientidele ressursside avastamisel (resource discovery). Infosüsteem peab olema skaleeritav, dünaamiline ja tõrkekindel oma ülesannete täitmisel.

Käesolev Infosüsteemi versioon on realiseeritud LDAP'i põhjal. Kuigi süstem on tõestanud oma efektiivsust, leidub mitu põhjust miks tuleb arendada uus infosüsteemi versioon. Esiteks, Globus ei toeta enam enda parandatud OpenLDAP teeki, mida oli kasutatud vana süsteemi loomisel. Teiseks, on vaja defeneerida standardne välisliides, et tagada erinevate grid-süsteemide omavaheline koostöö.

Informatsiooni Indekseerimisteenus (Information Indexing Service - IIS) on Infosüsteemi loogiline üksus. Meie defineerime IIS välisliidese, definitsioon vastab Veebitenuste spetsifikatsioonile. Liides koosneb kahest osast: Service Specific Interface (SSI) ja Local Information Description Interface (LIDI). SSI vastutab teenusespetsiifiliste meetodite täitmise eest ning LIDI annab ligipääsu andmetele. Selles töös toome ära IIS-SSI meetodite nimekirja ning defineerime kasutatavad andmestruktuurid.

Infosüsteemi kavandamisel eristame kolm komponendikihti, mille vahel vastutus on jagatud: IIS HED-Moodul (IIS-HED), IIS peakomponent (IIS-MAIN), Registratioonide Ladu (Registrations Store - RS). IIS-HED realiseerib SSI ja LIDI liideseid vastavalt spetsifikatsioonile. IIS-MAIN realiserib Infosüsteemi äriloogikat. RS vastutab registreerimisinfo "hajusladu" organiseerimise eest. Lisaks, me defineerime Infosüsteemi kliente: Registreerija (Registrant) ja Päringu komponent (Query component).

Süsteemi komponendid on programmeeritud Java keeles. IIS-HED moodul on kirjutatud vastavalt HED spetsifikatsioonile ning KnowARC API abil. Registrastioonide Lao

komponent tagab juurdepääsu andmebaasile. Kasutusele võetud andmebaas on eXist. eXist on XML natiivne andmebaas, mis võimaldab salvestada XML dokumekte ja täita XPath päringuid. Andmebaas on juurutatud sisseehitatud režiimis (embedded mode). IIS-HED ja RS komponendid on suhteliselt lihtsad, kuid IIS-MAIN kapseldab kogu süsteemi äriloogika, s.h. IIS eksemplaride võrgu moodustamine ja sõnumite vahetus selle võrgu kaudu.

Võrgu operatsioonide võimaldamine on IIS-MAIN komponendi peamine ülesanne. Võrgu operatsioonide eest vastutav komponendi osa on realiseeritud kasutades P2P lähenemisviisi. Me loome süsteemi JXTA raamistiku põhjal. JXTA on Sun Microsystems juhendamisel arendatav avatud koodiga projekt. JXTA defineerib protokollide komplekti, mis määravad erinevaid P2P tehnoloogia aspektid. Lisaks võimaldab JXTA P2P rakenduste arendamist Java ja C/C++ keeltes.

Töö tulemuseks on uus KnowARC infosüsteem. Süsteem juurutatakse ja kasutusele võetakse järgmises KnowARC grid-tarkvara versioonis.

Bibliography

[1] KnowARC, *Homepage*
http://www.knowarc.org/, last time viewed on 15.09.2007

[2] NorduGrid, *Homepage*
http://www.nordugrid.org/, last time viewed on 15.09.2007

[3] B.Kónya, *The NorduGrid/ARC Information System*
http://www.nordugrid.org/documents/arc_infosys.pdf, last time viewed on 15.09.2007

[4] Globus, *GT Information Services: Monitoring & Discovery System (MDS)*
http://www.globus.org/toolkit/mds/, last time viewed on 15.09.2007

[5] gLite, *Homepage*
http://glite.web.cern.ch/glite/, last time viewed on 14.09.2007

[6] JXTA, *Homepage*
https://jxta.dev.java.net/, last time viewed on 01.09.2007

[7] Brendon J. Wilson*, JXTA*, New Riders Publishing, 2002

[8] By Li Gong, Scot t Oaks, Bernard Traversat, *JXTA in a Nutshell*, O'Reilly, 2002

[9] *JXTA Java Standard Edition v2.5: Programmers GuideJune*, 2007
https://jxta-guide.dev.java.net/source/browse/*checkout*/jxta-guide/trunk/src/guide_v2.5/JXSE_ProgGuide_v2.5.pdf, last time viewed on 15.09.2007

[10] *JXTA v2.0 Protocols Specification*, 2006
https://jxta-spec.dev.java.net/nonav/JXTAProtocols.html, last time viewed on 10.09.2007

[11] OASIS, *WS-ResourceProperties (WSRF-RP) WSDL*
http://docs.oasis-open.org/wsrf/rpw-2.wsdl

[12] OGF, *GLUE 2.0 Specification*
http://forge.ogf.org/sf/go/doc14639?nav=1, last time viewed on 10.09.2007

[13] OASIS, *Web Services Resource Properties 1.2 (WS-ResourceProperties)*, 2006
http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf, last time viewed on 20.08.2007

[14] KnowARC, Design Document, 2007
http://www.knowarc.org/documents/Knowarc_D1.1-1_07.pdf, last time viewed on 15.08.2007

[15] KnowARC, *The ARC Container*, 2007
http://www.knowarc.org/documents/Knowarc_D1.2-2_07.pdf, last time viewed on 01.09.2007

[16] XML:DB Initiative for XML Databases, *Homepage*
http://xmldb-org.sourceforge.net/, last time viewed on 10.09.2007

[17] *KnowARC Reference Manual*
http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/doc/KnowARC-API.pdf, last time viewed 15.09.2007

[18] eXist, *Homepage*
http://exist.sourceforge.net/, last time viewed on 10.09.2007

[19] eXist, *Database Deployment*
http://exist.sourceforge.net/deployment.html, last time viewed on 10.09.2007

[20] eXist, *Service configuration*
http://exist.sourceforge.net/configuration.html, last time viewed on 10.09.2007

[21] eXist, Configuring Database Indexes
http://exist.sourceforge.net/indexing.html, last time viewed on 10.09.2007

Appendix A: Information Indexing Service XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Data types of Information Indexing Service
-->

<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:iis="http://www.knowarc.org/iis"
  xmlns:wsa="http://www.w3.org/2005/08/addressing"
  targetNamespace="http://www.knowarc.org/iis"
  elementFormDefault="qualified" attributeFormDefault="unqualified">

    <!-- xsd:import namespace="http://www.w3.org/2005/08/addressing" schemaLocation="./ws-
addr.xsd"/ -->
    <xsd:import namespace="http://www.w3.org/2005/08/addressing"
schemaLocation="http://www.w3.org/2006/03/addressing/ws-addr.xsd"/>


<!-- This is an initial and incomplete DRAFT which mainly concentrates on the structure but
not on the actual names. Final version will use GLUE-2.0 ternminology. -->

    <!-- ==== Input and output types for IIS operations ==== -->


    <!-- Input type for Register operation -->
    <xsd:complexType name="RegistrationMessageType">
        <xsd:sequence>
            <xsd:element name="Header" type="iis:HeaderType" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="RegEntry" type="iis:RegistrationEntryType" minOccurs="1"
maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="RegisterRequest">
        <xsd:sequence>
             <xsd:element name="RegistrationMessage" type="iis:RegistrationMessageType"
minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="Register" type="iis:RegisterRequest"/>

    <!-- Output type for Register operation -->
    <xsd:complexType name="RegisterResponseType">
        <xsd:sequence>
            <xsd:element name="Fault" type="iis:FaultType" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
    <element name="RegisterResponse" type="iis:RegisterResponseType"/>

    <!-- Input type for RemoveRegistrations and GetRegistrationStatuses operations -->
    <xsd:complexType name="RegistrationEntryIDListType">
        <xsd:sequence>
             <xsd:element name="RegEntryID" type="xsd:string" minOccurs="1"
maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="RemoveRegistrations" type="iis:RegistrationEntryIDListType"/>
    <xsd:element name="GetRegistrationStatuses" type="iis:RegistrationEntryIDListType"/>

    <!-- Output type for RemoveRegistrations operation -->
    <xsd:complexType name="RemoveRegistrationsResponseType">
        <xsd:sequence>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="RemoveRegistrationsResponse"
type="iis:RemoveRegistrationsResponseType"/>

    <!-- Output type for GetRegistrationStatuses operation -->
    <xsd:complexType name="GetRegistrationStatusesResponseType">
        <xsd:sequence>
            <xsd:element name="RegEntryStatus" type="iis:RegistrationEntryStatus"
minOccurs="1" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
```

```
    <element name="GetRegistrationStatusesResponse"
type="iis:GetRegistrationStatusesResponseType"/>

    <!-- === Helper type definitions === -->
    <xsd:complexType name="RegistrationEntryStatus">
        <xsd:sequence>
            <xsd:element name="RegEntryID" type="wsa:EndpointReferenceType" minOccurs="1"
maxOccurs="1"/>
            <xsd:element name="Status" type="iis:StatusType" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="Fault" type="iis:FaultType" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:simpleType name="StatusType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="1"/>
            <xsd:enumeration value="2"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:complexType name="HeaderType">
        <xsd:sequence>
            <xsd:element name="RequesterID" type="xsd:string"/> <!-- Identifier of the source
HED -->
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="RegistrationEntryType">
        <xsd:sequence>
            <xsd:element name="ID" type="xsd:string" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="SrcAdv" type="iis:ServiceAdvertisementType" minOccurs="1"
maxOccurs="1"/>
            <xsd:element name="MetaSrcAdv" type="iis:ServiceAdvertisementMetadataType"
minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>


    <xsd:complexType name="ServiceAdvertisementType">
        <xsd:sequence>
            <xsd:element name="Type" type="iis:ServiceTypeType"/>
            <xsd:element name="EPR" type="wsa:EndpointReferenceType"/>
            <xsd:element name="Impl" type="xsd:string"/>
            <xsd:element name="Version" type="xsd:string"/>
            <xsd:element name="URI" type="xsd:anyURI"/>
            <xsd:element name="AuthzInfo" type="iis:AuthorizationInfoType"/> <!--
Authorization information -->
            <xsd:element name="SSPair" type="iis:NameValuePairType" minOccurs="0"
maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="ServiceAdvertisementMetadataType">
        <xsd:sequence>
            <xsd:element name="Expiration" type="xsd:duration"/>
            <xsd:element name="GenTime" type="xsd:dateTime"/>
            <xsd:element name="Source" type="wsa:EndpointReferenceType"/>
            <xsd:element name="Status" type="xsd:string"/> <!-- Registration status: NEW,
ACCEPTED ... -->
        </xsd:sequence>
    </xsd:complexType>

    <xsd:simpleType name="FaultTypeType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="1"/>
            <xsd:enumeration value="2"/>
        </xsd:restriction>
    </xsd:simpleType>

    <!-- description of fault -->
    <xsd:complexType name="FaultType">
        <xsd:sequence>
            <xsd:element name="Name"        type="xsd:string"/>
            <xsd:element name="Type"        type="iis:FaultTypeType"/>
            <xsd:element name="Description" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
```

```
    <xsd:complexType name="FullServiceDescriptionType">
        <xsd:sequence>
            <xsd:element name="CoreSD" type="iis:CoreServiceDescriptionType" minOccurs="1"
maxOccurs="1"/>
            <xsd:element name="ExtSD" type="iis:ExtendedServiceDescriptionType" minOccurs="1"
maxOccurs="1"/>
            <xsd:element name="Resource" type="iis:ResourceDescriptionType" minOccurs="0"
maxOccurs="1"/>
             <xsd:element name="Activity" type="iis:ActivityDescriptionType" minOccurs="0"
maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="CoreServiceDescriptionType">
        <xsd:sequence>
            <xsd:element name="Status" type="xsd:string"/>
            <xsd:element name="Type" type="iis:ServiceTypeType"/>
            <xsd:element name="EPR" type="wsa:EndpointReferenceType"/> <!-- Endpoint reference
-->
            <xsd:element name="URI" type="xsd:anyURI"/>
            <xsd:element name="Name" type="xsd:string"/>
            <xsd:element name="Impl" type="xsd:string"/> <!-- Name of the implementation
(ComputingService: A-REX, ...) -->
            <xsd:element name="Version" type="xsd:string"/> <!-- Version of the service
implementation -->
            <xsd:element name="Profile" type="iis:ProfileType"/> <!-- Supported profiles -->
            <xsd:element name="AuthzInfo" type="iis:AuthorizationInfoType"/> <!--
Authorization information -->
            <xsd:element name="AuthInfo" type="iis:AuthenticationInfoType"/> <!--
Authentication information -->
        </xsd:sequence>
    </xsd:complexType>

    <!-- List of the service types will be provided by the GLUE-2.0 -->
    <xsd:simpleType name="ServiceTypeType">
        <xsd:restriction base="xsd:string">
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:complexType name="ProfileType">
        <xsd:sequence>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="AuthorizationInfoType">
        <xsd:sequence>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="AuthenticationInfoType">
        <xsd:sequence>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="NameValuePairType">
        <xsd:sequence>
            <xsd:element name="Name" type="xsd:string"/>
            <xsd:element name="Value" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>

    <!-- Service Specific attributes will be defined later following GLUE-2.0 -->
    <xsd:complexType name="ExtendedServiceDescriptionType">
        <xsd:sequence>
            <xsd:element name="Pair" type="iis:NameValuePairType" minOccurs="0"
maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <!-- Service Specific attributes describing resources will be defined later following
GLUE-2.0 -->
    <xsd:complexType name="ResourceDescriptionType">
        <xsd:sequence>
            <xsd:element name="Pair" type="iis:NameValuePairType" minOccurs="0"
maxOccurs="unbounded"/>
```

```
        </xsd:sequence>
    </xsd:complexType>

    <!-- Service Specific attributes describing activity will be defined later following GLUE-
2.0 -->
    <xsd:complexType name="ActivityDescriptionType">
        <xsd:sequence>
            <xsd:element name="Pair" type="iis:NameValuePairType" minOccurs="0"
maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

</xsd:schema>
```

Appendix B: Information Indexing Service WSDL

```
<?xml version="1.0" encoding="utf-8"?>
<!--
Interface of Information Indexing Service
-->
<wsdl:definitions
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsa="http://www.w3.org/2005/08/addressing"
  xmlns:iis="http://www.knowarc.org/iis"
  xmlns:wsrf-bf="http://docs.oasis-open.org/wsrf/bf-2"
  xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"
  xmlns:wsrf-rpw="http://docs.oasis-open.org/wsrf/rpw-2"
  xmlns:wsrf-rw="http://docs.oasis-open.org/wsrf/rw-2"
  targetNamespace="http://www.knowarc.org/iis"
>

    <wsdl:import namespace="http://docs.oasis-open.org/wsrf/rpw-2"
location="http://docs.oasis-open.org/wsrf/rpw-2.wsdl"/>
    <!-- wsdl:import namespace="http://docs.oasis-open.org/wsrf/rpw-2" location="./rpw-
2.wsdl"/ -->


    <!-- ======= Type definitions ====== -->
    <wsdl:types>
        <xsd:schema targetNamespace="http://www.knowarc.org/iis"
xmlns="http://www.w3.org/2001/XMLSchema" xmlns:iis="http://www.knowarc.org/iis">
            <xsd:include schemaLocation="./iis.xsd"/>

        </xsd:schema>
    </wsdl:types>

<!-- ====== Messages definitions ====== -->

<!-- ====== Register ====== -->
<wsdl:message name="RegisterRequest">
    <wsdl:part name="RegisterRequest" element="iis:Register"/>
</wsdl:message>

<wsdl:message name="RegisterResponse">
    <wsdl:part name="RegisterResponse" element="iis:RegisterResponse"/>
</wsdl:message>

<!-- ====== RemoveRegistrations ====== -->

<wsdl:message name="RemoveRegistrationsRequest">
    <wsdl:part name="RemoveRegistrationsRequest" element="iis:RemoveRegistrations"/>
</wsdl:message>

<wsdl:message name="RemoveRegistrationsResponse">
    <wsdl:part name="RemoveRegistrationsResponse" element="iis:RemoveRegistrationsResponse"/>
</wsdl:message>

<!-- ====== GetRegistrationStatuses ====== -->
<wsdl:message name="GetRegistrationStatusesRequest">
    <wsdl:part name="GetRegistrationStatusesRequest" element="iis:GetRegistrationStatuses"/>
</wsdl:message>

<wsdl:message name="GetRegistrationStatusesResponse">
    <wsdl:part name="GetRegistrationStatusesResponse"
element="iis:GetRegistrationStatusesResponse"/>
</wsdl:message>

<!-- ====== PortType definitions ====== -->
<wsdl:portType name="IISPortType">
    <wsdl:operation name="Register">
        <wsdl:input name="RegisterRequest" message="iis:RegisterRequest"/>
        <wsdl:output name="RegisterResponse" message="iis:RegisterResponse"/>
    </wsdl:operation>
    <wsdl:operation name="RemoveRegistrations">
```

```
        <wsdl:input name="RemoveRegistrationsRequest"
message="iis:RemoveRegistrationsRequest"/>
        <wsdl:output name="RemoveRegistrationsResponse"
message="iis:RemoveRegistrationsResponse"/>
    </wsdl:operation>
    <wsdl:operation name="GetRegistrationStatuses">
        <wsdl:input name="GetRegistrationStatusesRequest"
message="iis:GetRegistrationStatusesRequest"/>
        <wsdl:output name="GetRegistrationStatusesResponse"
message="iis:GetRegistrationStatusesResponse"/>
    </wsdl:operation>
</wsdl:portType>

<!-- ====== Bindings ====== -->

<wsdl:binding name="iis" type="iis:IISPortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="Register">
        <soap:operation soapAction="Register"/>
        <wsdl:input name="RegisterRequest">
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output name="RegisterResponse">
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="RemoveRegistrations">
        <soap:operation soapAction="RemoveRegistrations"/>
        <wsdl:input name="RemoveRegistrationsRequest">
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output name="RemoveRegistrationsResponse">
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetRegistrationStatuses">
        <soap:operation soapAction="GetRegistrationStatuses"/>
        <wsdl:input name="GetRegistrationStatusesRequest">
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output name="GetRegistrationStatusesResponse">
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>

<wsdl:binding name="GetResourcePropertyDocument" type="wsrf-rpw:GetResourcePropertyDocument">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="GetResourcePropertyDocument">
       <soap:operation soapAction="GetResourcePropertyDocument"/>
       <wsdl:input name="GetResourcePropertyDocumentRequest">
         <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output name="GetResourcePropertyDocumentResponse">
            <soap:body use="literal"/>
        </wsdl:output>
        <wsdl:fault name="ResourceUnknownFault">
            <soap:fault name="ResourceUnknownFault" use="literal"/>
        </wsdl:fault>
        <wsdl:fault name="ResourceUnavailableFault">
            <soap:fault name="ResourceUnavailableFault" use="literal"/>
        </wsdl:fault>
    </wsdl:operation>
</wsdl:binding>

<wsdl:binding name="GetResourceProperty" type="wsrf-rpw:GetResourceProperty">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="GetResourceProperty">
        <soap:operation soapAction="GetResourceProperty"/>
        <wsdl:input name="GetResourcePropertyRequest">
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output name="GetResourcePropertyResponse">
            <soap:body use="literal"/>
        </wsdl:output>
        <wsdl:fault name="ResourceUnknownFault">
            <soap:fault name="ResourceUnknownFault" use="literal"/>
```

```
        </wsdl:fault>
        <wsdl:fault name="ResourceUnavailableFault">
            <soap:fault name="ResourceUnavailableFault" use="literal"/>
        </wsdl:fault>
        <wsdl:fault name="InvalidResourcePropertyQNameFault">
            <soap:fault name="InvalidResourcePropertyQNameFault" use="literal"/>
        </wsdl:fault>
    </wsdl:operation>
</wsdl:binding>

<wsdl:binding name="QueryResourceProperties" type="wsrf-rpw:QueryResourceProperties">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="QueryResourceProperties">
        <soap:operation soapAction="QueryResourceProperties"/>
        <wsdl:input name="QueryResourcePropertiesRequest">
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output name="QueryResourcePropertiesResponse">
            <soap:body use="literal"/>
        </wsdl:output>
        <wsdl:fault name="ResourceUnknownFault">
            <soap:fault name="ResourceUnknownFault" use="literal"/>
        </wsdl:fault>
        <wsdl:fault name="ResourceUnavailableFault">
            <soap:fault name="ResourceUnavailableFault" use="literal"/>
        </wsdl:fault>
        <wsdl:fault name="InvalidResourcePropertyQNameFault">
            <soap:fault name="InvalidResourcePropertyQNameFault" use="literal"/>
        </wsdl:fault>
        <wsdl:fault name="UnknownQueryExpressionDialectFault">
            <soap:fault name="UnknownQueryExpressionDialectFault" use="literal"/>
        </wsdl:fault>
        <wsdl:fault name="InvalidQueryExpressionFault">
            <soap:fault name="InvalidQueryExpressionFault" use="literal"/>
        </wsdl:fault>
        <wsdl:fault name="QueryEvaluationErrorFault">
            <soap:fault name="QueryEvaluationErrorFault" use="literal"/>
        </wsdl:fault>
    </wsdl:operation>
</wsdl:binding>

<!-- ====== Service definition ====== -->
<wsdl:service name="iis">
    <wsdl:port name="GetResourcePropertyDocument" binding="iis:GetResourcePropertyDocument">
       <soap:address location="mailto:test@test.com"/>
    </wsdl:port>
    <wsdl:port name="GetResourceProperty" binding="iis:GetResourceProperty">
       <soap:address location="mailto:test@test.com"/>
    </wsdl:port>
    <wsdl:port name="QueryResourceProperties" binding="iis:QueryResourceProperties">
       <soap:address location="mailto:test@test.com"/>
    </wsdl:port>
    <wsdl:port name="iis" binding="iis:iis">
       <soap:address location="mailto:test@test.com"/>
    </wsdl:port>
</wsdl:service>

</wsdl:definitions>
```

Appendix C: eXist database configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<exist>
    <db-connection cacheSize="48M" collectionCacheSize="128" database="native" files="data"
free_mem_min="5" pageSize="4096">
        <pool max="5" min="1" sync-period="120000" wait-before-shutdown="120000" />
        <recovery enabled="yes" group-commit="no" journal-dir="data" size="100M" sync-on-
commit="no" />
        <watchdog output-size-limit="10000" query-timeout="-1" />
    </db-connection>

    <indexer caseSensitive="no" index-depth="5" preserve-whitespace-mixed-content="no"
stemming="no" suppress-whitespace="both"
tokenizer="org.exist.storage.analysis.SimpleTokenizer" track-term-freq="yes" validation="no"/>

    <serializer add-exist-id="none" compress-output="no" enable-xinclude="yes"  enable-
xsl="no" indent="yes" match-tagging-attributes="no" match-tagging-elements="yes" />

    <xquery enable-java-binding="no">
        <builtin-modules>
            <module class="org.exist.xquery.functions.util.UtilModule" uri="http://exist-
db.org/xquery/util" />
            <module class="org.exist.xquery.functions.transform.TransformModule"
uri="http://exist-db.org/xquery/transform" />
            <module class="org.exist.xquery.functions.xmldb.XMLDBModule" uri="http://exist-
db.org/xquery/xmldb" />
            <module class="org.exist.xquery.functions.request.RequestModule"
uri="http://exist-db.org/xquery/request" />
            <module class="org.exist.xquery.functions.response.ResponseModule"
uri="http://exist-db.org/xquery/response" />
            <module class="org.exist.xquery.functions.session.SessionModule"
uri="http://exist-db.org/xquery/session" />
            <module class="org.exist.xquery.functions.text.TextModule" uri="http://exist-
db.org/xquery/text" />
            <module class="org.exist.xquery.modules.example.ExampleModule" uri="http://exist-
db.org/xquery/examples" />
            <module class="org.exist.xquery.functions.validation.ValidationModule"
uri="http://exist-db.org/xquery/validation" />
            <module class="org.exist.xquery.functions.system.SystemModule" uri="http://exist-
db.org/xquery/system" />
        </builtin-modules>
    </xquery>

    <xupdate allowed-fragmentation="5" enable-consistency-checks="no" growth-factor="20" />
</exist>
```

Appendix D: Dependency libraries

Common

- log4j-X.X.X.jar

JXTA dependencies

- bcprov-jdk14.jar
- javax.servlet.jar
- jdom.jar
- jxta.jar
- jxtaext.jar
- org.mortbay.jetty.jar
- swixml.jar

eXist dependencies

- exist.jar
- exist-modules.jar
- antlr-X.X.X.jar
- commons-pool-X.X.jar
- jgroups-all.jar
- resolver.jar
- sunxacml.jar
- xmldb.jar
- xmlrpc-1.2-patched.jar

Appendix E: CD, Source code