A distributed JXTA-based architecture for searching and retrieving solar data

A. Sanna, C. Zunino, L. Ciminiera

Dip. di Automatica e Informatica - Politecnico di Torino,
C.so Duca degli Abruzzi 24, I-10129, Torino (Italy)

Abstract

This paper describes a hybrid distributed architecture, based on the JXTA technology, for searching and discovering data in a federation of solar observation archives. This work has been developed within the European project EGSO (European Grid of Solar Observations) with the intent of studying and analysing a set of technologies and showing the brokerage capabilities of a highly controlled distributed network topology.

Three roles have been identified: providers that contain data and metadata, brokers that manage providers and user queries, and consumers/clients. JXTA is used to develop all network system functionalities, while a native XML database is used to store resource descriptions.

Key words: archives federation, highly controlled distributed topologies, JXTA, resource sharing.

1 Introduction

Storing, organizing and searching data are key operations for a large spectrum of disciplines. In particular, they are challenging tasks when large and distributed data repositories have to be federated in order to provide the user an unified view of all archives. Distributed architectures such as peer-to-peer (P2P) networks and grids are natural candidates to tackle this problem.

Email addresses: andrea.sanna@polito.it (A. Sanna), claudio.zunino@polito.it (C. Zunino), luigi.ciminiera@polito.it (L. Ciminiera).

The Sun exhibits complex phenomena that pose important challenges to the understanding of the universe; moreover, understanding solar structure, and ultimately predicting solar behaviour, requires the correlation of many types of observations. A distributed architecture can allow to federate distributed archives providing researchers an unified view of solar data. Two main projects are currently under development to provide a "virtual solar observatory" where users could find the largest data repository concerning solar observations: VSO and EGSO. The Virtual Solar Observatory (VSO) (http://umbra.nascom.nasa.gov/vso/) is based on a conceptual model of a single (but mirrored) web service from which a user can carry out a search of all available VSO component archives locate data that fulfils a set of criteria. The European Grid of Solar Observation (EGSO) (http://www.egso.org/) is based on a more complex architecture that identifies three roles: providers, brokers, and consumers. Providers provide data and metadata, while brokers have to able to manage consumers queries. With the intent of studying and analysing a set of technologies potentially useful for EGSO, a highly controlled JXTA-based P2P architecture has been developed. Although EGSO will not be based on JXTA, the proposed architecture allows: to search for data in a federation of solar archives, to simulate (at least partially) the behaviour of EGSO, and to investigate issues concerning the design and the implementation of a distributed system on a large scale.

In the proposed solution system resources are described by XML documents and stored in native XML databases within provider peers. A provider shares data (files) and metadata (catalogues); a provider has to be connected to (at least) a broker to be part of system. The network of brokers provides the brokerage capability of system by managing providers and the consumer queries. Each broker is in charge to cope with a certain number of providers and it knows a limited number of neighbouring brokers. Clients/consumers access to the system by means of special access point peers named gateways that are directly connected to a broker. A broker forwards the consumer queries both to all its providers and its neighbouring brokers; the consumer receives results (metadata in XML format) in an incremental way, and it can choose the best provider from which downloading the resource.

Brokers and providers are implemented by a Java application that contains: the graphics user interface (GUI), a storage module (brokers do not have this module), a network module and two elements to manage both modules and the events. In particular, the storage module stores resource descriptors in the XML database Xindice, while the network module is built on JXTA services. This project is tailored to manage resources describing solar observations, therefore, the user can perform queries by specifying parameters such as: instrument of the observation, temporal range, wavelength range, and so on. Consumers can access to the system by web-peers (gateway) that allow the connection by the HTTP protocol.

The paper is organized as follows: goals of this work are presented in Section 2, while peer-to-peer technologies, XML databases, and JXTA are briefly analysed in Section 3. Architecture modules are described in the detail in Section 4 and the phases of a search operation are outlined in Section 5. Finally, an evaluation both of performance and reliability of the system is provided in Section 6.

2 Goals

The proposed solution represents a framework for the storage, search and retrieving of information within a distributed architecture. A set of goals have been considered in the design phase:

- Scalability: the adding of new nodes (peers) to the architecture has to be an easy task and system performance must not be affected.
- Versatility: the system has to be able to cope with information of any size and type.
- Platform independence: the access to the system has to be possible using almost any platform (Windows, Unix/Linux, Mac, and so on).
- Usage of standard technologies: the technologies used both for data manipulation and for the interaction among peers must be standard and possibly available under open source license.
- Modularity: the whole system is designed as a module collection and each module can be accessed exclusively by its own interface.

Moreover, despite of traditional P2P architectures, the topology of the system is a key issue in this project. Project specifications clearly identify three roles: providers, brokers and consumers. Providers are in charge to supply data (files) and metadata (catalogues), brokers provide users a "unified view" of the archives by managing queries and accesses to the system by means of gateway peers. On one hand, the system has to be easily expandable (new providers/brokers can be added to the architecture) but, on the other hand, the system topology has to be controlled in order to provide the user an efficient query management mechanism.

3 Background

This section aims to provide an overview of peer-to-peer networks and briefly presents two technologies largely used in this project: XML databases and JXTA.

3.1 P2P technologies

A distributed network architecture named peer-to-peer (P2P) is characterized by a direct access between peer computers, rather than through a centralized server. In particular a P2P system is different from the traditional client/server model because the applications involved act as both clients and servers: while they are able to request information from other servers, they also have the ability to act as a server and respond to requests for information from other clients at the same time. It is possible to classify peer-to-peer systems using two general aspects: the degree of centralization and the network structure.
A pure peer-to-peer application has no central server. Some examples of this kind of system are the original Gnutella[1] and Freenet[2].
In partially centralized P2Ps there are some nodes that assume a more "important" role than the rest of the nodes, acting as local central indexes for files shared by local peers. Some architectures of this kind are Morpheus, (http://www.morpheussoftware.net/), the last version of Gnutella, and Kazaa (http://www.kazaa.com/).
Finally, there are P2Ps with a centralized server (e.g. Napster). In unstructured networks (such as Gnutella), the placement of data (files) is completely unrelated to the overlay topology. On the opposite in structured networks, (such as Chord[3], CAN[4], Pastry[5], and so on) the overlay network topology is tightly controlled and files (or pointers to them) are placed at precisely specified locations. For a deeper description on peer-to-peer topologies see [6].

3.2 XML databases

XML[7] is becoming a data standard for exchanging information because using XML the data are portable. Also in the proposed architecture there is a great exchange of messages in XML. In particular, queries, query results, and resource descriptions are formatted using XML. For this purpose there are three possibilities: a traditional RDBMS, a XML-enabled database or a native XML database (for more details see also [8]). Using a traditional RDBMS needs normalizing XML structures into relations: this can be complex for designers, time-consuming for programmers and operationally inefficient for users. A XML enabled database is, typically, a traditional database with a middleware to manage XML data. It is also possible to store data in XML documents in a native XML database. A reason to store data in a native XML database is the retrieval speed. Depending on how the native XML database physically stores data, it might be able to retrieve data much faster than a relational database. The reason for this is that some storage strategies used by native XML databases store entire documents together physically. Native XML databases support features like transactions, security, multi-user access,

programmatic APIs, query languages, and so on. The only difference from other databases is that their internal model is based on XML.

There are two models that XML databases can use to store data. A text-based native XML database stores XML as text. Common to all text-based native XML databases are indexes, which allow the query engine to easily jump to any point in any XML document. This gives such databases a speed advantage when retrieving entire documents or document fragments. The second category of native XML databases is model-based native XML databases. Rather than storing the XML document as text, they build an internal object model from the document and store this model. How the model is stored depends on the database. Model-based native XML databases built on other databases are likely to have performance similar to those databases when retrieving documents because they rely on those systems to retrieve data. On the other hand, most such databases optimize their storage models and retrieval software. Many native XML databases supports the following features:

- Query languages. The most popular of these are XPath (with extensions for queries over multiple documents) and XQL, although numerous proprietary query languages are supported as well. In the future, most native XML databases will probably support XQuery from the W3C.
- Native XML databases have a variety of strategies for updating and deleting documents. As a general rule, each product that can modify fragments of a document has its own language, although a number of products support the XUpdate language from the XML:DB Initiative.
- Transactions. However, locking is often at the level of entire documents, rather than at the level of document fragments, so multi-user concurrency can be relatively low.

3.3 A brief overview of JXTA

Project JXTA [9,10] (http://www.jxta.org/) is a set of open, generalised peer-to-peer protocols that allow any connected device (cell phone, PDA, PC, and so on) on the network to communicate and collaborate. JXTA protocols establish a virtual network overlay on top of the Internet and non-IP networks, allowing peers to directly interact and organize independently of their network location. The Project JXTA software architecture is divided into three layers:

- Platform Layer (JXTA Core). The platform layer, also known as the JXTA core, encapsulates minimal and essential primitives that are common to P2P networking. It includes building blocks to enable key mechanisms for P2P applications, including discovery, transport (including firewall handling), the creation of peers and peer groups, and associated security primitives.

- Services Layer. The services layer includes network services that may not be absolutely necessary for a P2P network to operate, but are common or desirable in a P2P environment. Examples of network services include searching and indexing, directory, storage systems, file sharing, distributed file systems, resource aggregation and renting, protocol translation, authentication, and PKI (Public Key Infrastructure) services.
- Applications Layer. The applications layer includes implementation of integrated applications, such as P2P instant messaging, document and resource sharing, entertainment content management and delivery, P2P Email systems, distributed auction systems, and many others.

The JXTA network consists of a series of interconnected nodes, or peers. Peers can self-organize into peer groups, which provide a common set of services. In particular, JXTA denotes two particular peers called: rendezvous and edge. Rendezvous peers can be used to interconnect different groups logically separated. A rendezvous peer acts as a router; all (edge) peers connected to a rendezvous can interact with peers not belonging to the group only sending and receiving messages via a rendezvous peer. Rendezvous peers naturally allow to build a controlled topology (when a peer is set as rendezvous a list of other rendezvous peers to be used for setting up direct connections has to be specified) able to manage all communications needed for the proposed system.

Although the popularity of JXTA is growing and several JXTA-based applications are designed and developed both for research and commercial purposes, performance and scalability of JXTA are not still well understood. A deep analysis of JXTA performance can be found in [13,14]; in particular, latency times due to:

- start the JXTA platform;
- join a peer into a peer group;
- publish advertisements of a peer;
- open a pipe;
- learn about other peers; obtain pipe advertisements;
- the round-trip time (that is the time elapsed between the send request and the receipt of the acknowledgement); the transport protocol (HTTP, UDP, and TCP).

are analyzed and measured. On the other hand, the search efficiency of JXTA is a tradeoff between P2P networks that use a distributed hash-table (DHT) schema (i.e., CAN, Chord, Pastry) and P2P networks that use a flooding approach (i.e., Gnutella). A loosely-consistent DHT walker[11] approach for searching and discovering is implemented in JXTA. This approach has two main advantages: it does not require a strong-consistency DHT maintenance and it is well adapted to ad-hoc unstructured P2P networks.

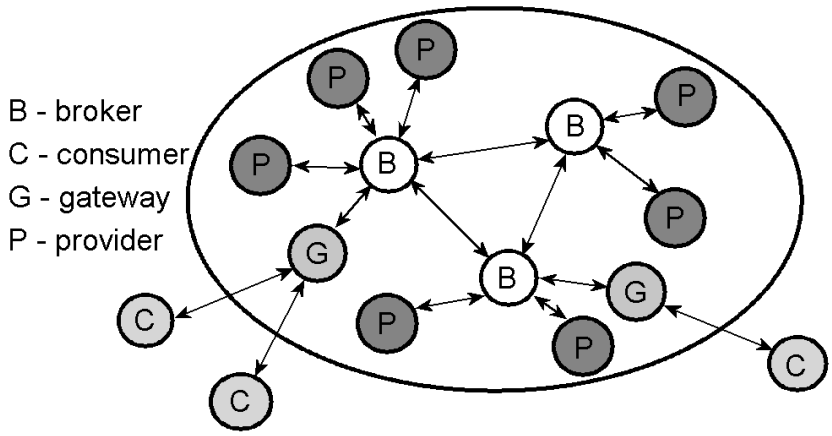

Fig. 1. The system architecture.

The JXTA community started a dedicated project to collect performance and scalability measurements; results and benchmark programs are available on the project we site (http://bench.jxta.org).

4 The proposed framework

The system architecture is shown in Figure 1. The system is built by a set of broker peers (rendezvous peers of JXTA) that allow the interconnection of all the other nodes. The network of brokers can be seen as the kernel of the system. Each broker is inserted in the architecture listing in the JXTA configuration table a set of *neighbouring* brokers. A data provider (an edge peer of JXTA) is added to the system by connecting it to a broker (for fault-tolerance purposes more than one broker/rendezvous peer should be indicated in the configuration table). This solution allows to build a balanced system where each broker is in charge to manage only a certain number of providers; when the number of providers connected to a broker grows beyond a threshold, the system administrator can manually insert a new broker.

The access to the system is performed by gateway peers (edge peers of JXTA). Each gateway is an access point to the system and allows to the consumers to "see" the resources. The gateway interface does depend on its implementation, for instance HTTP, SOAP (www.w3.org/TR/SOAP/), and so on.

Consumers/clients are not really a part of the architecture, but they can use services and get data from the system by means of gateways. A consumer cannot communicate with other consumers neither indirectly nor directly.

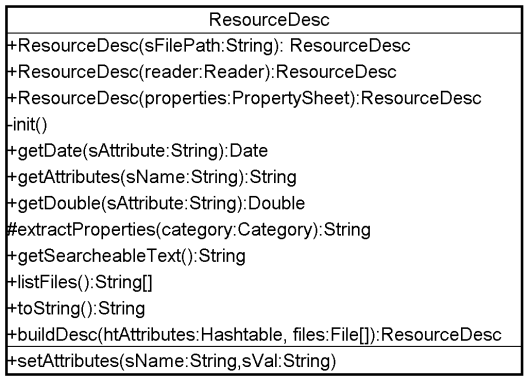

Fig. 2. The UML diagram of the Java class implementing the resource descriptor.

All information stored, indexed and searched within the system are represented as resources. A resource is an abstract entity denoting a set of files. A descriptor is associated with each resource; the descriptor has to specify structure and attributes of the resource. Attributes (metadata) contain information on the resource described as pairs *name-value* (for instance title of the resource); it does not exist any constraint on the name, value or order of the attributes. The descriptor is denoted by a XML document and the UML diagram of the Java class implementing it is shown in Figure 2.

4.1 The peer architecture

Each peer/node has been implemented as a Java application (the Java language has been chosen to fulfil portability requirements). Applications are organized as a collection of modules, where every module is a set of classes and interfaces Java. Every application can be configured by means of a XML file (for instance to choose the language according to the standard ISO-639 or the level of logs generated).

An application can be considered as composed by three different elements:

(1) the manager;
(2) the modules;
(3) the graphics user interface (GUI).

The manager has to manage both the modules and the events generated by the modules. In particular, the part of management is in charge of loading, executing and stopping the modules, while the event manager captures the events and routes them to the other elements. UML diagrams of module and event managers are shown in Figures 3 and 4, respectively. A set of modules has been designed in order to provide peer functionalities. Figure 5 shows how the elements of the architecture are interconnected. The module manager

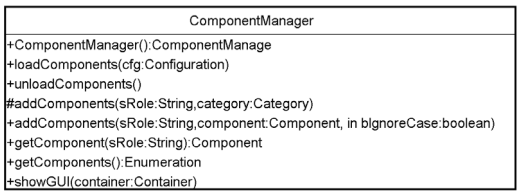

Fig. 3. The UML diagram of the Java class implementing the module manager.

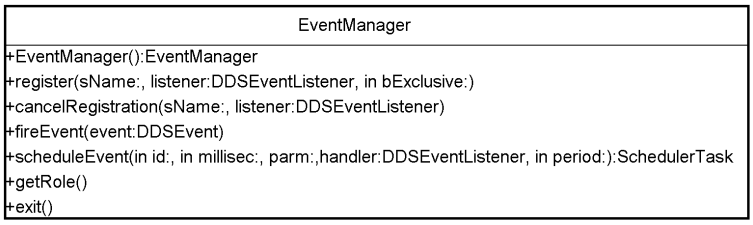

Fig. 4. The UML diagram of the Java class implementing the event manager.

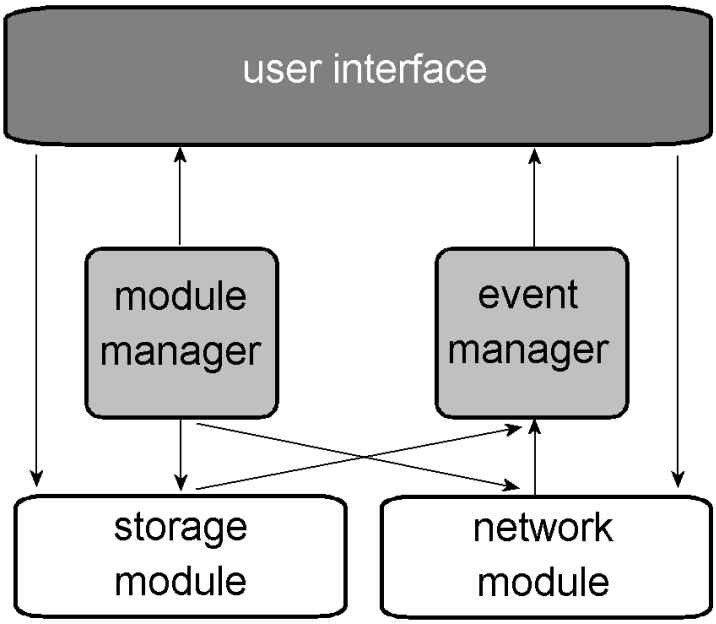

Fig. 5. Interconnection of the architecture elements.

provides information for the other modules and the user interface, while the event manager routes the information from modules to the interface. The interaction between the interface and a module is, in general, bi-directional and it is always started from the interface. In Figure 5 two modules are depicted: the storage and the network module. The storage module stores and indexes the resources (in general brokers do not have this module). Technological and implementation details are hidden to the other modules by means of a Java interface named StorageAcces that provides a generic way to store, search and retrieve data. As the only requirement of the storage module is to offer the

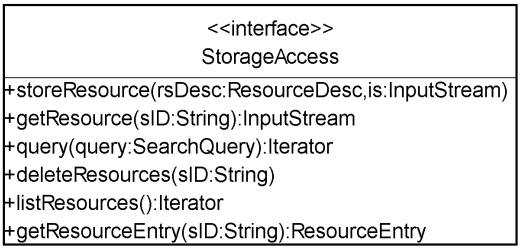

Fig. 6. The UML diagram of the storage module interface.

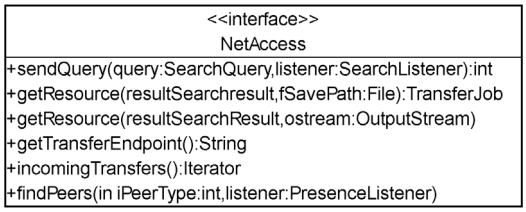

Fig. 7. The UML diagram of the network module interface.

interface of Figure 6, every application could implement a different storage strategy (for instance, relational DB, XML DB, LDAP, and so on).

As mentioned above, all system resources are described by means of a (potentially variable) XML document/schema. For this reason, the storage module has been developed by a native XML database. XML databases can be more "flexible" than a traditional RDBMS; moreover, Xindice, chosen for this project, can be embedded within the application, is distributed under the open source licence, and provides an XML:DB interface.

The storage module accesses to the database only by means of XML:DB APIs, in this way, it is compliant with all databases providing a XML:DB interface. A resource is considered as composed by two parts: a set of files and the metadata describing the resource itself. Files are compressed and stored in a directory on the disk (Xindice does not well manage large dimension binary files) while the metadata are stored in the database. The correspondence between data and metadata is collected in the resource descriptor. The network module contains all the code needed to build and maintain the network infrastructure. It acts as the interface between the network and the other elements of the application and it is in charge to autonomously maintain the (logical) network configuration. The network module is the only element of the application that knows the underlying network technology (JXTA in this case). The other modules communicate with the network one by means of the NetAccess Java interface (see Figure 7). The network module uses JXTA to build a hybrid P2P (see Figure 1), where broker functionalities are accomplished by rendezvous peers. In particular, the access to JXTA is implemented by the JXTAAccess class (Figure 8 shows the UML diagram of the JXTAAccess interface). The GUI allows the user to insert new resources in the database (functionality

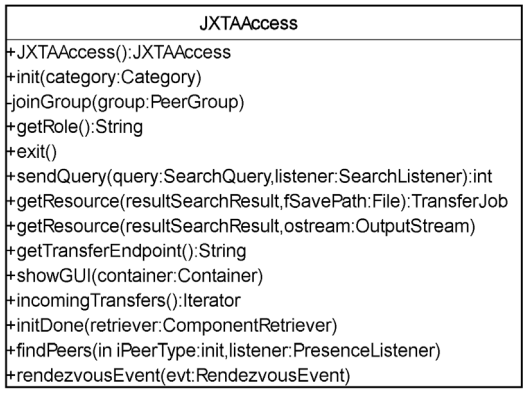

Fig. 8. The UML diagram of the JXTAAccess interface.

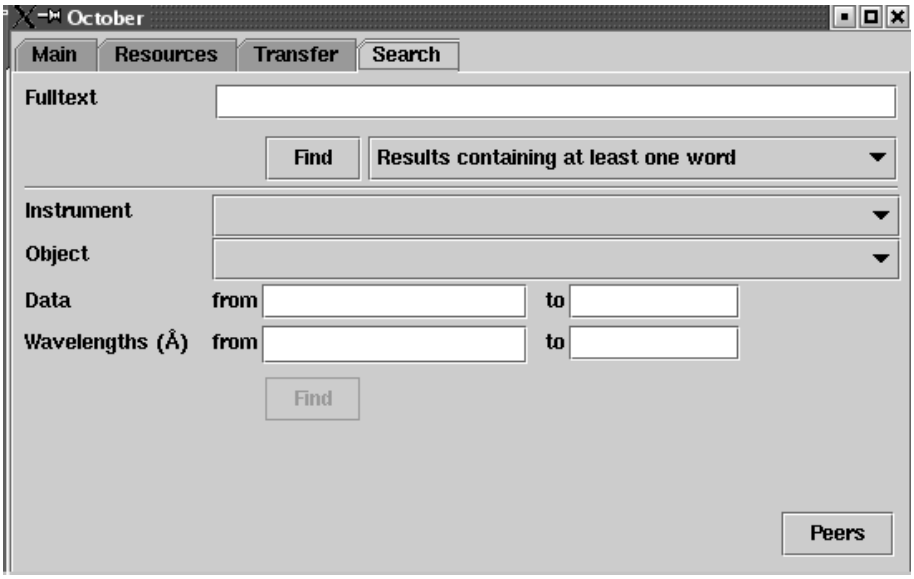

Fig. 9. The application GUI.

useful for providers) as well as to perform search operations. Searches can be also performed by consumer peers that, as already mentioned above, are not strictly belong to the system. Consumers connect to the system by gateway peers; in particular, it has been developed a web-peer allowing the user to connect the system by HTTP. The web-peer contains an embedded version of the web server Apache Tomcat.

As this project aims to provide a federation of solar archives, some typical parameters of solar observation have been considered to build providers catalogues. In particular, instrument, object, observation time (an interval denoted by the starting and the end dates) and the observation wavelength range can be specified in search operations. Queries can be also performed by specifying full text searches. The application GUI is shown in Figure 9; the web-peer interface is almost the same and it has been implemented by an applet.

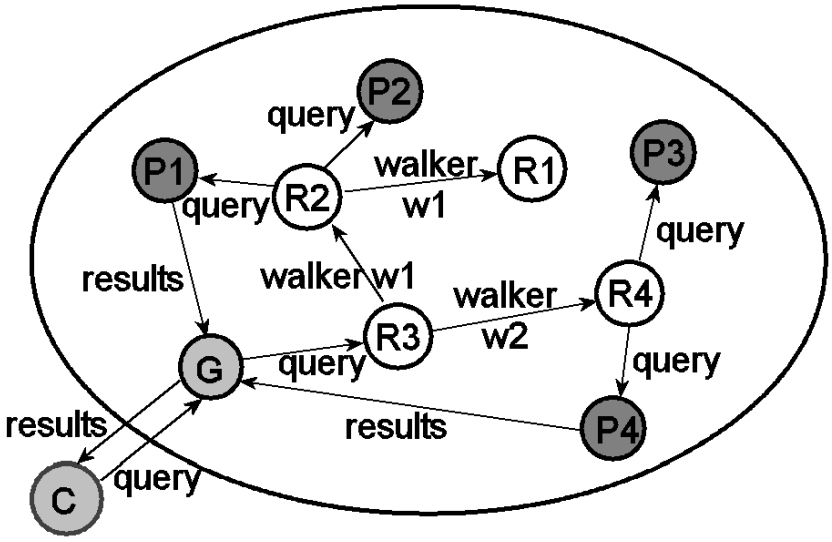

Fig. 10. An example of search operation.

5 Search operations

The search operation strategy is based on the walkers method provided by JXTA. When a consumer/client, connected to a broker by a gateway peer, want to perform a search, the query is sent to the broker that performs the following operations:

- Creates two walkers. The first one will visit neighbouring broker/rendezvous peers denoted by JXTA identifiers less than the one of the current broker, while the second one will visit neighbouring brokers denoted by greater JXTA identifiers; in this way, the walkers cannot visit the same peer.
- Sends the query to all the edge peers (providers) directly connected.

When a broker receives a walker from another broker:

- Routes the walker to neighbouring brokers if they exists. The walker mechanism is designed to avoid loops, that is, a walker cannot visit two times the same broker.
- Sends the query to all the edge peers (providers) directly connected.

Providers directly return results to the peer that started the search operation (i.e., the gateway peer). Figure 10 shows a search operation started from a consumer. The query, by the gateway, reaches the broker R3; R3 generates two walkers w1 and w2. The walker w1 visits the brokers R2 and R1, while w2 reaches R4. In this example only the providers contain the storage module and, in particular, P1 and P4 have data matching the consumer's query.

A set of providers has been configured to store a subset of the SOHO archive (http://sohowww.nascom.nasa.gov/) in order to test the proposed architecture. Several brokers have been geographically distributed over the north west of Italy in all sites of the Politecnico di Torino: Alessandria, Aosta, Ivrea, Mondovì, Torino, and Vercelli; each broker manages a certain number of providers (two-three providers for each broker).

A consumer can query the system or by a full text search or specifying a set of parameters (see Section 4.1). Both the application GUI and the web-peer applet provide the consumer the results (metadata) in an incremental way; results are updated and presented as they arrive in a tabular form. Each line of the resulting table contains the name of the provider, the resource name and, above all, the response time. The response time can be used as a sort of quality of service index and the consumer can choose to download the resource from the "best" provider. The resource is locally downloaded and decompressed (the web-peer applet must have the permissions to write on the disk); at this time, files can be further analysed and processed by means of ad-hoc software (solar physicists usually process solar observations by means of routines written in IDL language - http://www.rsinc.com/idl/).

6 Performance and reliability evaluation

The evaluation of a distributed architecture such as a grid or a P2P network is indeed a very challenging task. Performances strongly depend on a set of parameters: speed of network connections, system load, architecture topology, and so on. In this section we present a set of tests aimed to evaluate:

- the response time of the system almost independently of network delays.
- The latency of a broker to process and propagate a query to a provider.
- The reliability of the system in the event of crashes.

The configuration used in the first test involves a network of three brokers and three providers interconnected within a LAN (see Figure 11). Using the configuration of Figure 11, the list of peers visible from the provider1 is shown in Table 1. Moreover, a query was submitted from the provider1 (brokers and providers have a GUI that also allows to perform query operations; see Section 4.1) and response times are shown in Table 2. The three providers have the same replicated database; the name of the resource is reported in the first column, the size in the second column, while the last two columns list provider names and response times, respectively. In particular, it can be noticed that the response times of the brokers not directly connected to the provider1 (broker2 and broker3) are very similar to the response times of the connected edges peers (provider3 and provider2). These results are due to the

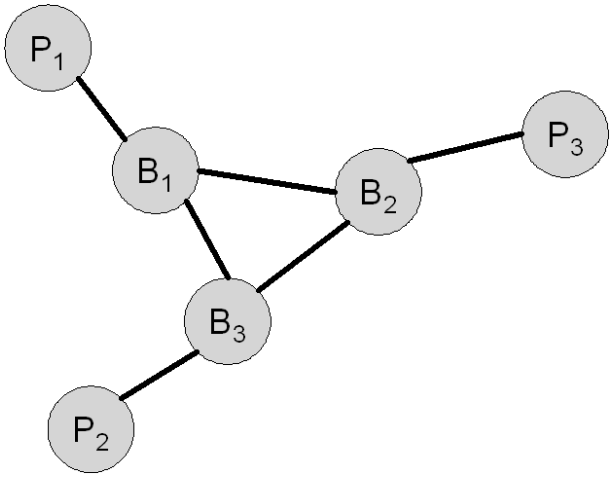

Fig. 11. The first configuration used for performance analysis tests.

Peer	**Topology**	**Response time**
provider1	Edge	0.019s
broker1	Rendezvous	0.317s
broker3	Rendezvous	0.508s
broker2	Rendezvous	0.509s
provider3	Edge	0.511s
provider2	Edge	0.512s

Table 1
Peers list received from the provider1.

logical position of the broker1: when a query is submitted using the broker1, the broker1 creates two walkers and the queries are submitted at the same time to broker2 and broker3.

The broker1 is shutdown in the second test configuration; the network topology becomes as shown in Figure 12. As it can be noticed, the provider1 is now connected to the broker3. In this new configuration the same two requests are submitted. Following the first one the provider1 gets the list of the connected peers (see Table 3). It can be noticed that in this new configuration the provider2 is faster than the provider3. The provider2 is also faster in the search process as shown in Table 4. These results are different from the ones obtained in the first test, in fact, while during the first test the provider2 and the provider3 had the same logical distance, now the provider2 is nearer than the provider3 to the provider1. The third test involves two brokers (broker1 and broker2) placed in Torino and in Vercelli, respectively. The two brokers are connected by a slow (2 Mb/s) and not dedicated network connection. A provider is connected to the broker1 within the same switched LAN. Response times of the provider1 to a query coming from the broker2 varied in the range

Title	Size	Peer	Response time
UVCS_CME_01.FITS	49.75KB	provider1	1.281s
UVCS_CME_02.FITS	43.89KB	provider1	1.291s
UVCS_CME_31.FITS	191.14KB	provider1	1.299s
UVCS_CME_41.FITS	33.42KB	provider1	1.311s
UVCS_CME_68.FITS	59.45KB	provider1	1.317s
UVCS_CME_121.FITS	176.12KB	provider1	1.324s
UVCS_CME_01.FITS	49.75KB	provider3	1.691s
UVCS_CME_02.FITS	43.89KB	provider3	1.701s
UVCS_CME_31.FITS	191.14KB	provider3	1.707s
UVCS_CME_41.FITS	33.42KB	provider3	1.731s
UVCS_CME_68.FITS	59.45KB	provider3	1.737s
UVCS_CME_121.FITS	176.12KB	provider3	1.749s
UVCS_CME_01.FITS	49.75KB	provider2	1.876s
UVCS_CME_02.FITS	43.89KB	provider2	1.975s
UVCS_CME_31.FITS	191.14KB	provider2	1.981s
UVCS_CME_41.FITS	33.42KB	provider2	1.988s
UVCS_CME_68.FITS	59.45KB	provider2	1.994s
UVCS_CME_121.FITS	176.12KB	provider2	2.0s

Table 2
Test1: response times for the submitted query.

Peer	Topology	Response time
provider1	Edge	0.019s
broker3	Rendezvous	0.164s
provider2	Edge	0.226s
broker2	Rendezvous	0.281s
provider3	Edge	0.334s

Table 3
Peers list received from the provider1.

from 2 s to 3.5 s. These times slightly change when the provider1 is directly connected to the broker2: broker1 introduces an additional delay due to the query management of about 0.15 s.

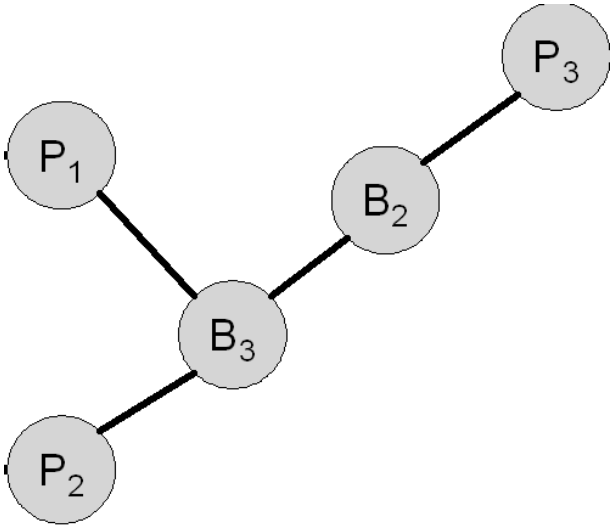

Fig. 12. The second test configuration.

A set of tests have been also performed to evaluate robustness and reliability. A small network of three brokers B_1, B_2, and B_3 managing several providers has been built to simulate crashes. JXTA identifiers (IDs) are in the following relationship: $ID(B_1) < ID(B_2) < ID(B_3)$. These cases have been tested:

- B_2 does not manage any provider and crashes: sub-networks leaded by B_1 and B_3 have been isolated for some minutes, afterward the two brokers have automatically built a direct link. This can be explained by considering the IDs. B_2 has an intermediate ID between B_1 and B_3, therefore, both B_1 and B_3 sent walkers to B_2; a direct link between them was established when they discovered the crash of B_2.
- B_1 does manage some providers and crashes: providers are automatically "inherited" from another broker (B_3) and all peers are achievable.
- A provider crashes: it is immediately "discarded" from the list of available peers.

In a network with just one broker and a set of providers, the crash of the broker leads to the election of a new broker chosen among the providers.

7 Conclusions and remarks

The proposed framework allows to manage an archives federation by a hybrid P2P architecture. A network of brokers provides the user all search functionalities tailored to query solar observation databases. All resources in the system are described by XML documents; at the moment only data and metadata are managed, but computing services could be also integrated (for instance some providers could be specialized in executing IDL routines over the data

Title	Size	Peer	Response time
UVCS_CME_01.FITS	49.75KB	provider1	0.593s
UVCS_CME_02.FITS	43.89KB	provider1	0.602s
UVCS_CME_31.FITS	191.14KB	provider1	0.61s
UVCS_CME_41.FITS	33.42KB	provider1	0.616s
UVCS_CME_68.FITS	59.45KB	provider1	0.706s
UVCS_CME_121.FITS	176.12KB	provider1	0.713s
UVCS_CME_01.FITS	49.75KB	provider2	1.3s
UVCS_CME_02.FITS	43.89KB	provider2	1.309s
UVCS_CME_31.FITS	191.14KB	provider2	1.315s
UVCS_CME_41.FITS	33.42KB	provider2	1.341s
UVCS_CME_68.FITS	59.45KB	provider2	1.367s
UVCS_CME_121.FITS	176.12KB	provider2	1.373s
UVCS_CME_01.FITS	49.75KB	provider3	1.523s
UVCS_CME_02.FITS	43.89KB	provider3	1.529s
UVCS_CME_31.FITS	191.14KB	provider3	1.535s
UVCS_CME_41.FITS	33.42KB	provider3	1.574s
UVCS_CME_68.FITS	59.45KB	provider3	1.58s
UVCS_CME_121.FITS	176.12KB	provider3	2.0s

Table 4
Test2: response times for the submitted query.

downloaded from other peers).

The topology of the architecture is strictly controlled in order to provide an efficient query management. Each broker copes with a limited number of providers and knows a limited number of neighbouring brokers used to forward user queries. The JXTA underlying level provides all functionalities to manage the logical network; moreover, JXTA includes a set of fault tolerance mechanisms needed to guarantee robustness and reliability.

The resulting application used to implement a peer is extremely compact, portable and easily expandable/modifiable. JXTA already manages advertisements by an embedded native XML database that has been also used to store resource descriptors. All code is entirely written in Java and the application has been designed as a set of modules communicating by means of well defined interfaces.

Acknowledgements

Authors wish to thank Marco Ciancimino for his support in the design and the developing of the proposed architecture.

References

[1] Clip2 The Gnutella Protocol Specification v0.41 Document Revision 1.2.

[2] Clarke, I., Sandberg, O., & Wiley, B., Freenet: A distributed anonymous information storage and retrieval system, 2000, In Proceedings of the Workshop on Design Issues in Anonymity and Unobservability, Berkeley, California.

[3] Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., & Balakrishnan, H., Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications, 2001, In ACM SIGCOMM, 149-160.

[4] Ratnasamy, S., Francis, P., Handley, M., Karp, R., & Shenker, S., A Scalable Content-Addressable Network, 2001, In ACM SIGCOMM, 161-172.

[5] Rowstron, A., & Druschel, P., Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems, 2001, In Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms.

[6] Androutsellis-Theotokis, S., A Survey of Peer-to-Peer File Sharing Technologies, 2002, White paper, Athens University of Economics and Business.

[7] Harold, E.R. & Means, W.S.: *XML in a Nutshell*, 2nd edition, O'Reilly & Associates (2002).

[8] Champion, M., Storing XML in Databases, 2001, eAI Journal, http://www.eaijournal.com/PDF/StoringXMLChampion.pdf.

[9] Gong., L., JXTA: A network programming environment, 2001, IEEE Internet Computing, 5, 88-95.

[10] Botros, S., & Waterhouse, S., Search in JXTA and other distributed networks, 2001, Proceedings of the First International Conference on Peer-to-Peer Computing, 30-35.

[11] Traversat, B., Abdelaziz, M., & Pouyoul., E., A Loosely-Consistent DHT Rendezvous Walker, white paper, Sun Microsystems, Inc.

[12] Sun JXTA Engineering Team, 2002. JXTA Platform Scalability Proposed Design, Draft Version 0.5.

[13] Halepovic, E., & Ralph, D., JXTA Performance Study, 2003, Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and signal Processing, 149-154.

[14] Halepovic, E., & Ralph, D., The Cost of Using JXTA, 2003, Proceedings of the third International Conference on Peer-to-Peer Computing, 160-168.