The BRICKS Infrastructure - An Overview

Thomas Risse[1], Predrag Knežević[1], Carlo Meghini[2], Robert Hecht[3], and Fiore Basile[4]

[1] Fraunhofer IPSI
Integrated Publication and Information Systems Institute
Dolivostrasse 15, 64293 Darmstadt, Germany
{risse|knezevic}@ipsi.fhg.de

[2] CNR ISTI
Via G. Moruzzi 1, 56124 Pisa, Italy
carlo.meghini@isti.cnr.it

[3] ARC Seibersdorf Research - Research Studios
Studio Digital Memory Engineering
Thurngasse 8, A-1090 Wien, Austria
robert.hecht@researchstudio.at

[4] METAware S.p.A.
Via F. Turati 43/45, 56125 Pisa, Italy
f.basile@metaware.it

Abstract. The aim of the BRICKS project is to design, develop and maintain an open user and service-oriented infrastructure to share knowledge and resources in the Cultural Heritage domain. Typical usage scenarios are integrated queries among several knowledge resource, e.g. to discover all Italian artifacts from renaissance in the European museums or to follow the life cycle of historic documents. These examples are specific applications, which are running on top of the BRICKS infrastructure. The BRICKS infrastructure will use the Internet as a backbone and will fulfil the requirements of expandability, graduality of engagement, scalability, availability, and interoperability.

In addition, the user community has the economic requirement to be low-cost. This means (1) that an institution should be able to become a BRICKS member with minimal investments, and (2) that the maintenance costs of the infrastructure, and in consequence the running costs of each BRICKS member, are minimised. Also, the BRICKS membership will be flexible, such that parties can join or leave the system at any point in time without administrative overheads.

In order to fulfil these requirements the BRICKS architecture will be decentralised, based on a peer-to-peer (P2P) paradigm, i.e. no central server will be employed. Every member institution of a BRICKS installation is a node (a BNode in the BRICKS jargon) of the distributed architecture. The foundation components (bricks, in the BRICKS jargon) making up the BRICKS architecture are Web Services, and provide functionalities for maintaining the system, for content- and metadata management, or security. Applications are composed out of these base components and can add new ones to provide additional functionalities.

1 Introduction

The aim of the BRICKS project [1] is to design, develop and maintain an open user and service-oriented infrastructure to share knowledge and resources in the Cultural Heritage domain. The target audience is very broad and heterogeneous and involves cultural heritage and educational institutions, research community, industry, and citizens. Typical usage scenarios are integrated queries among several knowledge resource, e.g. to discover all Italian artefacts from renaissance in the European museums. Another example is to follow the life cycle of historic documents, whose manual copies are distributed all over Europe. These examples are specific application, which are running on top of the BRICKS infrastructure. The BRICKS infrastructure uses the Internet as a backbone and has to fulfil the following requirements:

- Expandability, which means the ability to acquire new services, new content, or new users, without any interruption of service.
- Scalability, which means the ability to maintain excellence in service quality, as the volumes of requests, of content and of users increase.
- Availability, which means the ability to operate in a reliable way over the longest possible time interval.
- Graduality of Engagement, which means the ability to offer a wide spectrum of solutions to the content and service providers that want to become members of BRICKS.
- Interoperability, which means the ability to make available services to and exploit services from other digital libraries.

In addition, the user community has the economic requirement to be low-cost. This means (1) that an institution should be able to become a BRICKS member with minimal investments, and (2) that the maintenance costs of the infrastructure, and in consequence the running costs of each BRICKS member, are minimized.

Interested institution should not invest much additional money in its already existing infrastructure to become a member of BRICKS. In the ideal case the institution should only get the BRICKS software distribution, which will be available for free, install it, connect to the internet and become a BRICKS member. This will already gives the possibility to search for content and access some services. For sure, additional work is necessary to integrate and adapt existing content and services to provide them in BRICKS.

Also, the BRICKS membership will be flexible, such that parties can join or leave the system at any point in time without administrative overheads. To minimize the maintenance cost of the infrastructure any central organization, which maintains e.g. the service directory, should be avoided. Instead, the infrastructure should be self-organizing in a way that the scalability and availability of the fundamental services, e.g. service discovery or metadata storage, are guaranteed.

The paper is structured as follows. In the next Section we give details about the overall architecture. Afterwards in Section 3 we describe our approach to handle data in decentralized environments. Section 4 describes the content management of BRICKS. The handling metadata will be described in Section 5. Afterwards in Section 6 we introduce the BRICKS approach for personalized, cross-language information access. The

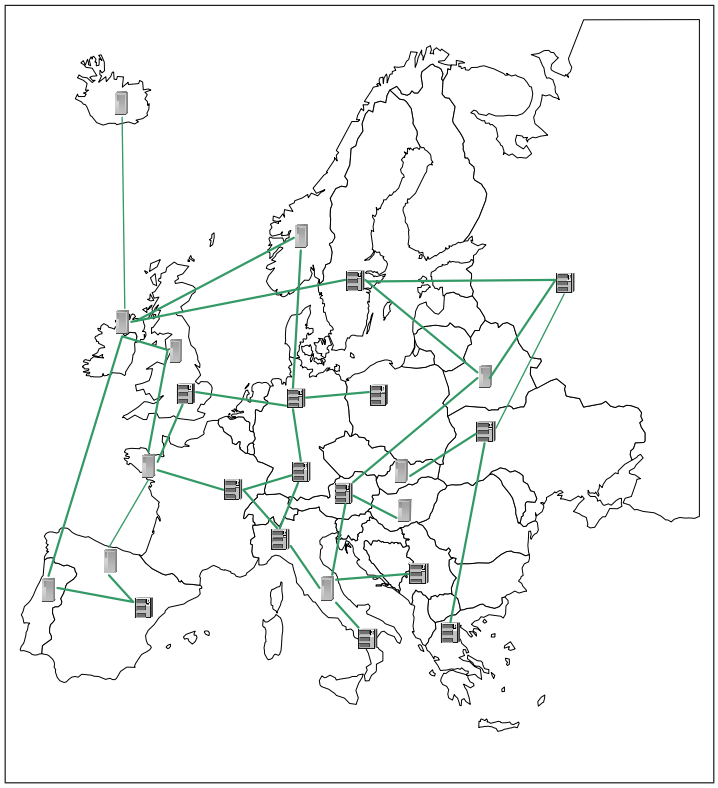

Fig. 1. Decentralized BRICKS Topology

security architecture will be described in Section 7. Finally Section 8 gives conclusions and outlook.

2 Overall Architecture

In order to fulfil these requirements the BRICKS architecture will be service-oriented and decentralized. There are several reasons for the decentralization approach: Decentralized architectures do not have a central points, which could stop or slowdown in failure or overload situations. Instead of that, the decentralized architectures offer better load-balancing, and a failure will not influence system availability in whole. Only parts of the system might be affected. Furthermore, having central points (servers) limits their scalability in a long-run. It is always a good idea to design a system in a way that it is able to handle loads, which was not foreseen during the initial design phase. For standard client-server the system load must be carefully estimated, e.g. by guessing the maximum number of users. In open systems like BRICKS this is not possible. Due to the simplicity of joining the system the number of nodes can increase rapidly. Hence upper limits can not be given. With the decentralization approach we can avoid single

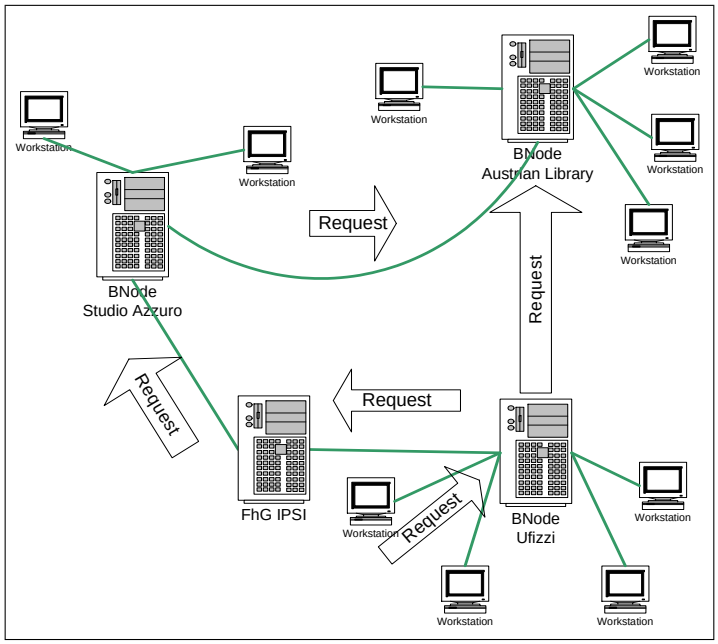

Fig. 2. Request routing in BRICKS

points of failure of core functionalities, e.g. service discovery, by completely distributing them.

Furthermore, the lack of central points removes the needs for centralized maintenances. Important infrastructure functionalities are all implemented in a self-organizing way or using services which are implement in that way. That is a strong advantage of the decentralized approach because a centralized administration costs additional money and personnel must be dedicated for the tasks. BRICKS is going to be very heterogeneous system without a central body that will maintain the system. Also, our aim is to make a system whose infrastructure costs are as low as possible, and a decentralized architecture is good approach that has the necessary properties.

Figure 1 shows a possible topology of the future BRICKS system. Every node represents a member institution, where the software for accessing the BRICKS is installed. Such nodes are called BNodes. BNodes communicate among each-other and use available resources for content and metadata management. Every BNode knows directly only a subset of other BNodes in the system. However, if a BNode wants to reach another member that is directly unknown to it, it will forward request to some of its known neighbour BNodes that will deliver the request to the final destination or forward again. This is depicted in Figure 2. It shows also that BRICKS users access the system only through a local BNode available at their institution. Hence every user request is first sent to the institution's BNode and then the request is routed between other BNodes to the final destination. Search requests behave like that; the BNode will preselect a list of BNodes where a search request could be fulfilled, and then the BNode will route it

there. When the location of the content is known, e.g. as a result of the query, the BNode will directly be contacted.

3 Decentralized XML Data Management

It is very common in a system architecture to have a place where different sorts of data are stored and later retrieved for a variety of purposes. Usually, such places are centralised, i.e. there are one or many know servers and large number of clients that use the storage.

One of important requirements of the BRICKS project is defining a completely decentralised architecture, even for needed storages. Thus, we cannot apply standard centralised approach, i.e. our storage that should be accessible by every BNode must be decentralised as well. At the same time, usage of such storage must not be different from the centralised approach, i.e. all additional complexity introduced by decentralisation should be hidden in exposed application interface (API).

In order to fulfill the above requirements, we have designed a P2P datastore that manages XML documents within a P2P community. The approach differs from the current P2P datastores, where each peer makes their local files available for community download. XML documents are split into finer pieces that are spread then in the community. The documents are created and modified by the community during system run-time and they can be accessed from any peer in a uniform way, e.g. a peer does not have to know anything about the data allocation.

The decentralized XML storage should be used for all data that must have very high availability. Examples are service descriptions, administrative information about collections, and annotations which need to be globally available during the whole life of the system. Other sorts of data like descriptive metadata of content are not required to be stored there, as they are kept locally under the control of the owner.

Database Architecture

Within BRICKS we design and implement a decentralized/P2P datastore that manages XML documents within a P2P community. The approach differs from current P2P datastores, like Gnutella or Kazaa, where each peer makes their local files available for community download. In our approach XML documents are split into finer pieces that are spread then in the community. The documents are created and modified by the community during system run-time and they can be accessed from any peer in a uniform way, e.g. a peer does not have to know anything about the data allocation.

The proposed datastore mimics a subset of the Document Object Model (DOM) [2] interface, which has been widely adopted among developers, and a significant legacy of application is already built on top of the DOM. Therefore, many applications could be ported easily to the new environment. XML query languages like XPath [3] or XQuery [4] can use the DOM as well, so they could be used for querying of the datastore. Finer data granularity will make querying and updating more efficient.

The storage must be able to operate in highly dynamic communities. Peers can depart or join at any time, nobody has a global system overview or can rely on any

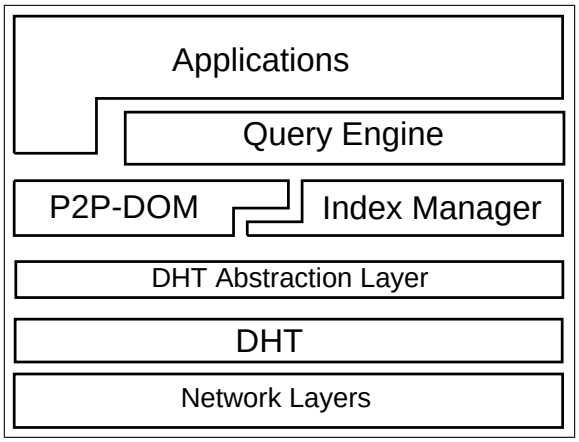

Fig. 3. Peer-to-peer XML Storage Architecture

particular peer. These requirements differ significantly from those in the distributed databases where node leaving is only due to some node failure and the system overview is globally known.

Figure 3 presents the proposed database architecture. All layers exist on every peer in the system. The datastore is accessed through the P2P-DOM component or by using the query engine. The query engine could be supported by an optional index manager that maintains indices. Note, that the query engine and index manager for the XML storage are different from those used to query metadata on a BNode (cf. Section 6).

P2P-DOM is the core system layer. It exports a large portion of the DOM interface to the upper layers, and maintains a part of a XML tree in a local store (e.g. files, database). P2P-DOM serves local requests (adding, updating and removing of DOM-tree nodes) and requests coming from other peers through a Distributed Hash Table (DHT) [5–7] overlay, and tries to keep the decentralized database in a consistent state. In order to make the DHT layer pluggable, it is wrapped in a tiny layer that unifies APIs of particular implementations, so the upper layer does not need to be modified.

4 Content Management

BRICKS decentralized architecture mandates that each BNode should take care of storing its locally produced content. Such content is organized in a hierarchical way inside the so called *physical collections*. Each content item is stored within one and just one physical collection, inside a structure called *DLObject*. The DLObject is the minimal entity in the BRICKS information model. The *Content Management brick* takes care of maintaining the physical collection hierarchy and DLObject stored inside it.

4.1 BRICKS DLObjects and the BRICKS Content Model

As one may easily imagine, digital objects play a key role in a digital library, therefore one of the biggest challenges in defining the BRICKS architecture was to identify a general model for the storage and handling of digital objects.

Each digital object must be able to hold structured textual and binary data, in order to allow end-user applications to manage the complex data structures present in modern cultural-heritage applications. Moreover digital objects must be easily imported and exported to and from the digital library, so it is necessary to support standard interchange formats like XML.

For this reason, within BRICKS, digital objects are modeled through a meta content model, that allows to define, through a full type hierarchy and inheritance, complex content type definitions. A BRICKS DLObject, uniquely identified by a URN [8], can therefore be associated a specific type definition (which inherits from the basic *DLObject* type) and be composed according to this definition by a hierarchy of *nodes* and *properties* of several kind (text, binary file, etc).

The type definition states which nodes and which properties are allowed to be created under a specific DLObject, and can be created by importing a plain XML Schema [9] file. This also implies a very strict mapping to an XML format representation of all BRICKS DLObjects, allowing easy import and export of content in BRICKS.

The meta content model definition is based on the Java Content Repository (JCR) [10] standard, which is also used as a basis for the Content Manager architecture and reference implementation, as described in the following section.

BRICKS provides a set of pre-defined content models, including FRBR [11]-inspired by ECHO and DoMDL [12], MPEG-21 DIDL, TEI-Lite [13] and a subset of DocBook. Additional content models will be added as new user applications are produced.

4.2 Content Manager Architecture

As all other BRICKS Foundation component the BRICKS Content Manager exposes a web service API to be used by other components and by end-user applications.

Such API closely mirrors the Java Content Repository (JCR), recently standardized as JSR-170. Java Content Repository was defined as a common API for all Java-based content management systems, able to act as glue between different vendor's implementations, and allowing replacing one content management system with another without necessarily rewriting all the content presentation code.

JCR defines both a meta content model and a full set of functionality for accessing and manipulating content. The BRICKS Content Manager, based on the reference implementation of the JCR standard, provided by the Apache Jackrabbit project, adds an additional web-service layer on top of the JCR API, and provides a set of pre-defined type definitions and utilities to allow easy integration of JCR within the BRICKS platform.

Figure 4 illustrates the Content Manager architecture: the web-service interface extends and uses Apache Jackrabbit over a set of several system data stores, that may include file system, relational databases and XML storage facilities. Through configuration files the BRICKS Content Manager allows the definition of new types of DLOb-

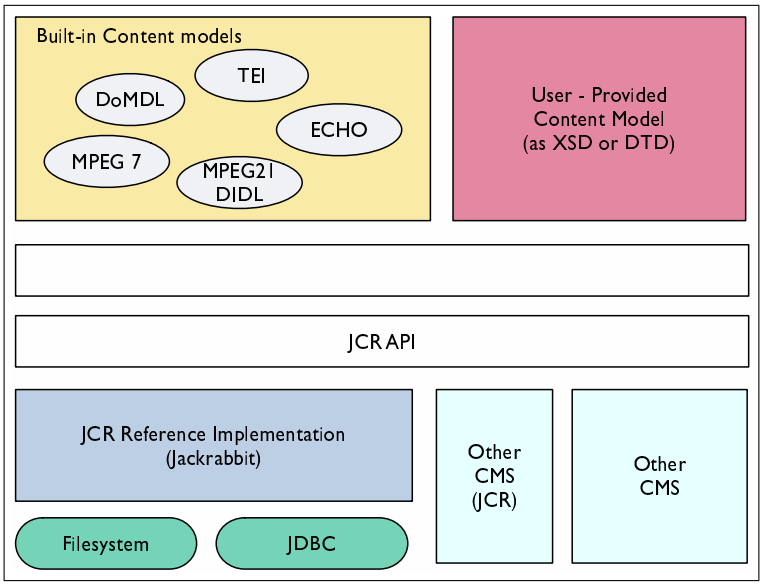

Fig. 4. The BRICKS Content Manager Architecture

jects, all based on a common type, which defines basic properties such as the BRICKS unique identifier.

In the future, an additional wrapper to the Content Manager web-service interface will be provided, allowing user applications to use the standard JCR Java API instead of the BRICKS web-service implementation, when writing Java software. This will allow greater flexibility for developers to port their application from and to non-BRICKS platforms.

4.3 Content Manager functionality

BRICKS Content Manager offers a great deal of functionality, thanks to the underlying JCR engine. The BRICKS specific functionality includes the creation and management of Physical Collections and DLObjects, the locking and versioning of DLObjects, browsing of DLObject content, type and version hierarchy, the management of the BRICKS unique identifiers (in the form of URNs). Advanced functionality like concurrent editing of same content items, staging of content, merging of different content revisions is also offered through the BRICKS content manager API.

5 Metadata Management

The BRICKS Metadata Manager handles *descriptive* Metadata, i. e. Metadata describing the contents of a document or resource. Descriptive Metadata for a text document might include e.g. title, author, creation date, a set of keywords, publisher etc. Other

kinds of metadata are e.g. information on file size and format, access rights, intellectual property rights etc. These Metadata are not managed by the Metadata Manager.

Descriptive Metadata are mainly used to find documents matching certain conditions (see section 6), e.g. to retrieve all documents by a certain author or on a given topic. In fact, for non-textual documents like pictures or audio files, the descriptive Metadata are the *only* basis to find documents matching certain criteria. Therefore, they play a key role in BRICKS.

5.1 Metadata Manager Architecture

The architecture of the Metadata Manager is shown in fig. 5. It consists of two layers: the lower layer (RDF graph layer) uses the Resource Description Framework (RDF) [14] as internal representation language. RDF models all data as *triples* of the form *Subject - Predicate - Object*. For example the fact that a certain painting was created by Kazimir Malevič would be modeled in RDF as triple *"Black square" (Subject) - createdBy (Predicate) - "Kazimir Malevič" (Object)*. Since the Object can be the Subject of another triple, complex connected graphs can be built. Another way to view RDF triples would be Attribute-Value-pairs: the subject can be thought of having a set of attributes (the predicates of the triples) with given values (the objects).

The upper layer (presentation layer) of the Metadata Manager - its external interface - summarizes triples that have the same subject into structures called Metadata Records. The reason for this is that the RDF graph, while very useful for inferencing, can be complex and confusing.

Furthermore, all metadata are required to conform to a given *Metadata Schema*. In the RDF graph layer, we use (a subset of) the Web Ontology Language (OWL) [15] for the definition of Metadata Schemas. Metadata Schemas describing the most important standards will be pre-loaded at the startup of the Metadata Manager. Additional Metadata Schemas can be loaded at any time by submitting a file containing an XML serialisation of the OWL ontology. We want to point out that a given Metadata Schema *must* be loaded into the Metadata Manager before metadata using this schema can be stored. In most cases, Metadata Schemas contain just a hierarchy of (ontology) classes and a set of properties (attributes) defined on these classes. A Metadata Schema can import an arbitrary number of other Metadata Schemas, but these have to be loaded before the schema that imports them. The relation between the Metadata Schema (defined in OWL) and the actual Metadata (defined in RDF) is similar to the relation of an XML Schema definition and the corresponding XML File.

In the presentation layer, the contents of a Metadata Schema can be retrieved, e.g. it is possible to query for classes, attributes, etc. defined in the schema.

5.2 Functionality of the Metadata Manager

The Metadata Manager offers functionality to create, modify, and delete Metadata Records and to load Metadata Schemas. Besides, it supports the Query Processor by allowing retrieving all Metadata Records that fulfill certain criteria, e.g. all documents by a certain author. It should be pointed out that client applications should not use this functionality

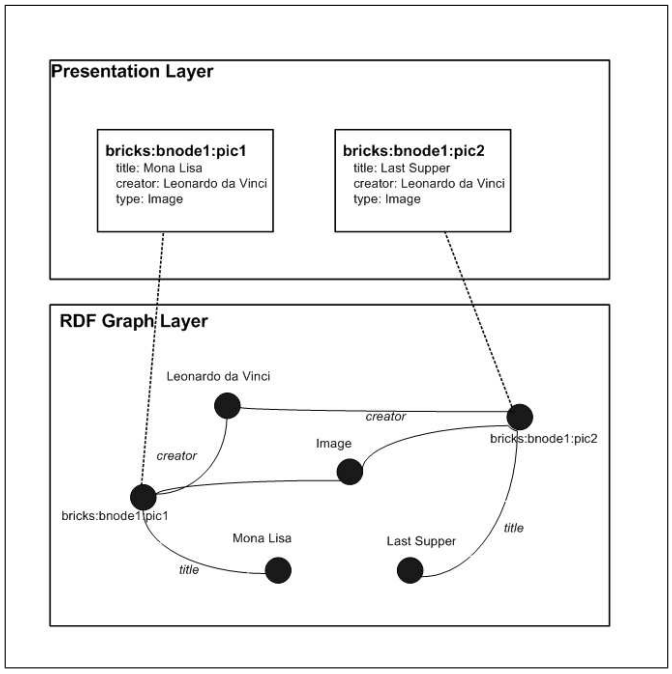

Fig. 5. The BRICKS Metadata Model

directly, because the Metadata Manager resolves queries only locally (i.e. on the BNode where it runs). This is because also the Metadata are stored only locally. To enable resolving queries that affect several BNodes, the Metadata Manager collaborates with the Indexer component: whenever a Metadata Record is created, changed, or deleted, the Metadata Manager propagates this information to the Indexer. The indexer is a distributed component which the Query Processor uses to resolve distributed queries.

6 Personalized, cross-language information access in BRICKS

The information access facility of the BRICKS architecture is based on the BRICKS query language, which supports 3 types of searches on metadata:

- *simple search*, which is a full-text search on metadata records;
- *advanced search*, expressed in terms of a Boolean combination of simple (*attribute operator value*) conditions, where *attribute* is a field of a metadata schema;
- *ontology search*, expressed in terms of a complex query on an OWL DL ontology.

The latter two types of queries are expressed in the syntax of SPARQL, an emerging query language for RDF. The evaluation of each type of queries is based on a 2-level architecture: The *Query Mediator* (QM) on the global level and the *local query processors*. The Query Mediator undertakes the query evaluation process, by appropriately using local query processors. At the local level exist a number of local query processors,

each dedicated to a specific type of queries and a specific set of data. These processors are not implemented by BRICKS: they are built by integrating into the BRICKS architecture existing software components. Specifically we use Lucene [5] for simple search, and Jena [6] for advanced and ontology search.

The BRICKS Query Mediator

Upon receiving a query, the QM performs the following operations.

1. The QM identifies the nodes of the network where the relevant data and processors are placed; this operation may be complicated by the fact that the user has specified no collection in the query, thus all nodes are potentially relevant; since only a limited number of nodes can be addressed, for efficiency reasons, the QM adopts a suitable *source selection strategy* to limit the search;
2. The QM breaks the given query into sub-queries, which are subsequently delivered to the appropriate nodes for evaluation. This operation may be complicated by several factors.
   - First, there may exist mappings defined on the terms used in the query to other terms of a different ontology or metadata schema. In this case, the Ontology Manager is invoked in order to re-write the query to take into account such mappings.
   - Second, the user may have requested a personalized query evaluation. In this case, the query needs to be re-written in order to take into account user preferences as represented in profiles.
   - Third, the query may need to be expanded to take into account multi-linguism. This possibility only exists for simple searches: the expansion concerns the addition of the translations of the involved terms in the appropriate languages.
3. The QM collects the results and delivers them back to the users. This operation is complicated by the fact that the size of the result may be very large, potentially as large as the whole digital library. For this reason, the QM always requires the invoker to specify the desired portion of the result, which is considered to be linearly ordered. Such portion is specified by 2 integers:
   - *maxBunch*, giving the size of the desired portion (clearly, this must be not larger than a prefixed parameter); and
   - *startingPosition*, giving the relative position of the first desired item in the result.

In order to achieve scalability, the QM operates in a stateless way: it never caches anything. It is the responsibility of the application to manage local caching, if required. For instance, a GUI application might ask 100 objects at a time to the QM (*e.g., maxBunch=100*) but only display 20 of them to the user at any one time.

[5] `http://lucene.apache.org/java/docs/`
[6] `http://jena.sourceforge.net/`

7 Security

Security of a complex framework like BRICKS must cover a wide array of topics: services like authentication, authorization, accounting, privacy, trust and digital rights management have to be provided as a basis for building of base services and applications.

7.1 BRICKS Security features

Providing security in BRICKS, which is the basis for a trusted digital library framework, requires a series of features which include:

- Asserting user identity, in order to guarantee secure operation on all the BNodes. Moreover user identity must be federated, as each BNode maintains information only on his registered users.
- Providing access control, to allow only authorized users to perform sensible operations on content and related structures. An additional required feature is to enrich access control with dynamic, reputation-based *trust* information about the organization to which a user belongs.
- Logging and billing, as each user operation should be traceable and billable when needed.
- Safeguard of intellectual property rights, to provide a trustful environment for content producers, where they could collect revenue and protect their rights while allowing content to be distributed to final consumers.
- Providing interoperability with existing security infrastructures, as many organization want to keep their investments in infrastructure and organization.

In order to offer these features BRICKS provides a modular security architecture, described in the following section.

7.2 BRICKS Security architecture

BRICKS security architecture as shown in Figure 6 can be seen as a set of interoperable modules that cooperate inside a BNode. Security is also based on external systems such as user directories, external digital rights stores, organization accounting systems and payment gateways, that interact with the security architecture each with their own interface.

A service-oriented interface is available to application services and other BNodes. Each service invocation must carry information about the user who is invoking the service, the credentials provided, the security context on which the user wants to operate and the name of the operation to be executed. The *Security Management* module obtain these information and then uses the underlying modules as follows: *User Management* module is used to verify the user identity, for example by checking the user-provided signed certificate, and gathering which groups the user belongs to along with roles associated to them.

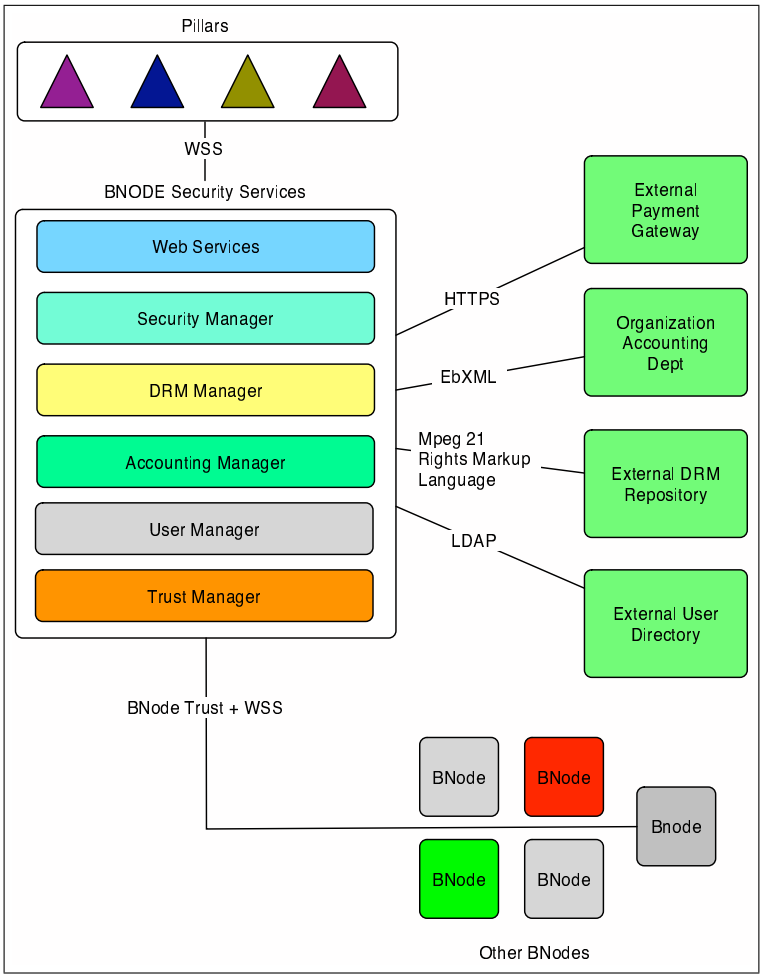

Fig. 6. The BRICKS Security Architecture

The *Digital Rights Management* module takes part to the process by checking if the security context is protected by digital rights and which steps are required to obtain digital rights clearance. The DRM module can deny the service invocation by providing such instructions, or directly issue authorization and generating accounting events to be handled by the Accounting Management module.

The *Accounting Management* module is used to check and handle accounting events linked to the operation and the security context on which it has to be executed, generating one or more accounting events, for logging purposes or for billing. As already said there is a strong dependency on the Digital Rights Management module, which is the principal source of accounting events and which use accounting manager transactions as conditions to obtain clearance for digital rights protection.

The BNode acts as a proxy to the BRICKS community on behalf of the user. As the user requires to access a service offered by another BNode the service invocation must be forwarded to the proper BNode via the Trust module. The *Trust Management* module maintains structures that hold security information on the surrounding neighbor BNodes. It forwards user and role information along the service request to the other BNode service interface. The request, when received, is processed according to external role and trust policies and the answers forwarded to the user.

7.3 Core security technologies

BRICKS makes use of several state-of-the-art standards and technologies in order to implement a modern and robust security architecture.

In particular all web-service communication is secured through the WS-Security [16] set of standards, which also include XML-Encryption [17] and XML-Signature [18].

The federated identity service is implemented by using the SAML (Security Assertions Markup Language) [19], while access control policies are defined via XACML (eXtensible Access Control Markup Language). The Trust Management module is modeled after a variant of the Poblano trust model. Finally the DRM module uses the widely adopted MPEG-21 REL (Rights Expression Language) [20] for the definition of DRM policies and licenses.

Other standards used include HTTPS, x509 [21] certificates, LDAP (to access external user directories) and ebXML (for integration with organization's billing and accounting systems).

8 Conclusions and Future Work

In this paper we presented the approach we have taken in BRICKS to address the problem of sharing knowledge and resources in the Cultural Heritage domain. The BRICKS architecture is layered, Web-service-based and decentralized. The paper describes how our approach is different from the existing ones in the Cultural Heritage domain, i.e. we give details how some specific issues like content management, metadata management, and security are handled in the decentralized system. The next steps are the implementation of the components and services of the BRICKS framework and the instantiation of the infrastructure, by installing some BNodes at the partner organizations.

References

1. BRICKS Project: BRICKS - Building Resources for Integrated Cultural Knowledge Services (IST 507457). (2004) http://www.brickscommunity.org/.
2. : Document Object Model. (2002) http://www.w3.org/DOM/.
3. W3C: XML Path Language (XPath) Version 1.0. (1999) http://www.w3c.org/TR/xpath.
4. W3C: XML Query. (2003) http://www.w3c.org/XML/Query.
5. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. Lecture Notes in Computer Science **2218** (2001)
6. Aberer, K.: P-Grid: A self-organizing access structure for P2Pinformation systems. Lecture Notes in Computer Science **2172** (2001)

7. Stoica, I., Morris, R., Karger, D., Kaashoek, F., Balakrishnan, H.: Chord: A scalable Peer-To-Peer lookup service for internet applications. In: Proceedings of the 2001 ACM SIGCOMM Conference. (2001) 149–160
8. R.Moats: RFC2141: URN Syntax. (1997) http://www.ietf.org/rfc/rfc2141.txt.
9. W3C: XML Schema Part 0: Primer Second Edition - W3C Recommendation. (2004) http://www.w3.org/TR/xmlschema-0/.
10. David Nuescheler, Day Software: JSR-000170 Content Repository for JavaTM technology API. (2005) http://jcp.org/aboutJava/communityprocess/final/jsr170/index.html.
11. IFLA: Functional Requirements for Bibliographic Records. (1998) http://www.ifla.org/VII/s13/frbr/frbr.pdf.
12. OpenDLib: DoMDL XML Schema. (2004) http://www.opendlib.com/resources/schemas/domdl.xsd.
13. Lou Burnard: TEI Lite: An Introduction to Text Encoding for Interchange. (2002) http://www.tei-c.org/Lite/teiu5_en.html.
14. W3C: Resource Description Framework (RDF). (2005) http://www.w3c.org/RDF.
15. W3C: OWL Web Ontology Language Reference. (2004) http://www.w3.org/TR/owl-ref/.
16. OASIS Web Services Security (WSS) Technical Committee: Web Services Security: SOAP Message Security 1.0. (2004)
17. Draft, W.W.: XML Encryption Syntax and Processing. (2002)
18. Draft, W.W.: XML Signature Syntax and Processing. (2001)
19. OASIS Web Services Security (WSS) Technical Committee: Web Services Security: SAML Token Profile. (2004)
20. ISO/IEC JTC1/SC29/WG11, N.: MPEG-21, Information Technology, Multimedia Framework, Part 5: MPEG-21 Rights Expression Language. (2003)
21. S. Santesson et al. and R. Fielding and L. Masinter: Internet X.509 Public Key Infrastructure Qualified Certificates Profile (2000)