ContentP2P: a peer-to-peer content management system

Gerardo Canfora, Sandro Manzo, Vincenzo F. Rollo, Maria Luisa Villani,
(canfora@unisannio.it, s.manzo@unisannio.it, f.rollo@unisannio.it, villani@unisannio.it)

RCOST – Research Centre on Software Technology
Department of Engineering, University of Sannio

Abstract

The paper presents a new application of content management based on a peer-to-peer platform. Several advantages following our choice of the peer-to-peer paradigm within the content management context are discussed, such as the improved scalability and flexibility of the system and the preserved ownership and off-line control of content from content creators.

1 Introduction

The expression “Content Management” is largely used in industry, but sometimes with different meanings. Following [6], a content management system is a distributed software system that treats information in a granular way, enabling the access, versioning, and dynamic assembly of pieces of information, named contents, such as diagrams, tables, images, or pieces of text.

Companies’ recent interest towards content management solutions has increased on the basis of three main issues:

1. The spreading of semi or non-structured data. Documents have irregular structure and great importance is placed on the order of data within the document. In this respect, the introduction and subsequent widespread adoption of XML [15] as a way of storing and exchanging information has opened new challenges in the content management context. In fact, the fast development of query languages for XML documents has made such a technology playing a fundamental role within information retrieval systems.

2. The need for managing and controlling contents rather than entire documents. Content management systems may be considered as an extension of document management systems. Similarly to document management systems, content management systems use metadata to classify and search information and address the entire lifecycle of information, from its creation to the presentation to end users. However, an essential difference on the objectives of the two systems makes them divergent. Namely, a document management system allows users to deal with files that are stored in, and controlled by, the system, whereas the end user of a content management system aims to the creation of a document, which might not exist statically, as a composition of contents that are stored in, and retrieved by, the system after the user’s request. In other words, a content management system focuses on content reusability.

3. Collaboration and internetworking within and between enterprises. Distributed content management systems support the syndication process, namely they allow the transfer of information from a company to its business partners, still maintaining control over the content. In fact, such systems satisfy the general need for producing and accessing content wherever it is, thus allowing for collaboration within and between enterprises by sharing data in real time. Furthermore, the fast spreading of new and mobile devices pushes content creators in structuring information so that the same document can be presented in different ways depending on the device from which the request is issued.

The distributed architecture of a content management system can be thought of as made of a relatively small set of servers, each with its own repository, on which all functionalities of the system act. However, several reasons suggest the peer-to-peer paradigm be more appropriated for this kind of applications. In a peer-to-peer environment, in fact, content can be stored and controlled in the place where it is created, avoiding the need for a structure where on one side there are specific machines devoted to accessing and managing the repository of data and on the other side machines who can only require services and need to send off to a server the content they have produced.

By choosing a peer-to-peer platform for our application, we also have turned around the need for building a centralized repository for a large amount of semi-structured data, which would have also lead to the problem of synchronizing multiple accesses to the data and would have increased the net load towards each server. In fact, our repository simply consists of a collection of XML files located on each peer of the
community. Combining a content management application with a peer-to-peer architecture, content can be easily accessed and searched inside a company as well as from outside; users do not have to look directly for a repository that contains the information they need. Instead, they perform a global search in a virtual environment and may even ignore where the information has been actually found. Two additional features of the peer-to-peer architecture are low costs in the booting phase and the possibility to keep the existing infrastructure.

The idea of building a content management application upon a peer-to-peer platform is not new. This has been independently proposed by others, Gartner Consulting [16] among those, after the success of the well-known systems Napster [17] and Gnutella [18]. These systems have proved the peer-to-peer paradigm be suitable in an environment where the better chance of finding the desired files and the robustness and scalability of the system come before reliability or performance issues. However, as far as we know, content management applications using the peer-to-peer paradigm, as the system we present here, are not available.

The client-server architecture has monopolized the software market for long time before the re-discovery of peer-to-peer. Most of the existing distributed applications are client-server, including most content management software systems. From a research that we have made on the state-of-the art in this area, we have come across the products listed below, which use a client-server platform. Moreover, all these systems use XML technology for the reasons discussed earlier in this introduction.

- Blade-Runner by INTERLEAF [6];
- Documentum 4i [7];
- Frontier by USERLAND [8];
- Content Management Suite by POET [9];
- Prowler (INFOZONE, open source) [10];
- Vignette Content Suite v6 [11];
- Slide (open source) [12].

However, we could not compare their architectural styles with that of our system, as the list above consists mainly of commercial products and their internal structure is not transparent from outside.

In this paper we present a prototype implementation of our idea, which applies the existing technologies of JXTA platform [1] as middleware peer-to-peer, and Kweel [2] as information retrieval system. At the moment this prototype fulfils only the basic functions of a content management system, such as content creation and search, as it is meant to be part of our feasibility analysis of the system. The paper is organized as follows. Section 2 is devoted to the description of the requirements that we have set for our system. In Section 3 the architecture of the system is exposed, together with a brief description of its enabling technologies. In Section 4 we introduce an example of implementation and, finally, in Section 5 we discuss some remarks and ideas for improvements to the system.

2 System Requirements

In this section we outline the requirements we aim to fulfill in our system, giving prominence to those who differentiate our system, named ContentP2P, from the existing content management systems.

2.1 Content classification and control

A Content Management System (CMS) should be able to manage content coming from two main sources: customers and repositories. Moreover, the system should access to whichever database, legacy system, XML archives, remote web server and business applications content. To this aim, such content may be converted by a system into a XML format, via a suitable tool. Metadata offer a good way to classify any piece of information and this makes it possible to search for content in a very simple and efficient way. In this respect, the Dublin Core schema [5], provides a consolidated and nearly standardized way for documents classification.

2.2 Off-line content management

The content management system we propose addresses itself to users however dislocated. Users should be able to access content independently of their physical location; they should also be able to change content for which they have adequate permissions. Consequently, the system must maintain the coherence and consistency of content. This requirement may be satisfied if the activity of content manipulation is carried out off-line and new content is made accessible as a new version of the old one.

2.3 Integration with productivity tools

Our content management system should be easy to use by users able to work with productivity tools. Indeed, the aim is to allow users to work with the tools they are familiar with, including text editors, word-processors, editors of graphs, tables and matrices, and spreadsheets.

2.4 Dynamic assembly

A content management system should be able to dynamically create documents, including Web pages, with a predefined structure. By this we mean the possibility to get, as a result of a query, a page containing the most recent version of the content stored in the repository.
2.5 Separation of content from its presentation style

A modern content management system should allow content to be independent from its style of presentation. This can be obtained through XML technology, where the style for content presentation is specified in a XSL file. A great advantage is the context awareness feature: once a device connects to a content management site, we can imagine that the software recognizes that particular device and consequently visualizes the information using a suitable style sheet (as an example, if we are connected through a cellular device WAP, the presentation style should be lightened from all the graphical part). Unfortunately, the technologies currently available do not allow for a cellular WAP to act as a peer node, for obvious software reasons. However, in this field technology evolves quickly and cellular devices supporting light Java platforms already exist. Also, we might suppose to start a connection from a mobile device to one of the active peers (in this case we need to know the address of that peer) and make the last one acting as a server for the mobile. We aim to a deep investigation of these aspects in the near future.

2.6 Repository based on content

The choice of the type of the repository is strongly influenced by the structure of the data to be stored. In our case, we have some XML documents and we want to offer to users a means for searching them. These documents have a highly irregular structure, which makes it difficult to store them in a relational database. Relational databases offer several services, such as security, transactions and data integrity, multi-user access, and queries across multiple documents, which ease the management of content. However, in our case we primarily seek a flexible storage that keeps information about the physical structure of each document and the order in which content occurs in the document. This makes relational databases unsuitable. As an example, if we want to keep in a relational database the information about the order of appearance of children elements in their father element, we would be obliged to store this information in a separate column. There are systems that offer the possibility to easily store XML structured documents (see for example dbXML [3]) but they use a dedicated server. Another solution that we have considered is the opportunity to pass from a XML document to an object oriented database [4]. However, finally we opted for a native XML repository rather than passing through a database because we think it is more efficient in terms of answer time.

2.7 Use of open standards

The various nature of information sources calls for a standard way of communication among different systems. Currently, the combined use of Java and XML [13] appears to be the best solution as on one side there is a portable language that allows applications to access data and, on the other side, through XML technology information is treated independently from its visualization.

Apart its structural features (architecture based on a peer-to-peer platform and lack of a proper database), our system is different from the existing ones because of the following two unique requirements:

- the user’s ownership over the content, and
- his capability of choosing the granularity of the content once it is retrieved by the system.

2.8 Peer-to-peer and content ownership

Within highly decentralized and mobile systems, the traditional client-server paradigm seems not be sufficient. Imposing a tight coupling between clients and server, this paradigm does not satisfy the scalability and flexibility requirements imposed by modern distributed domains. In this context, the peer-to-peer paradigm has been recently revalued. Napster, Gnutella and, most recently, the JXTA project are examples of applications in which every local host shares information with all other peers of the community, leading to a new way of thinking about the Internet. However, other reasons motivate the migration from client-server to a peer-to-peer environment. In the last one, in fact, customers decide directly what resources they want to share globally, without the need for publishing them on some server. Information and services are not on a single accumulation point of the network; instead, every peer is responsible for a subset of services it makes available on the net (content ownership). In this respect, it is worth stressing that the content management system we have developed is based on the JXTA peer-to-peer architecture [1]. Like Gnutella, our system is characterized by the lack of a single server: many servants (every machine is indeed both client and server) that receive the user’s queries, forward them to all known peers, as well as attempting to satisfy them.

2.9 Content granularity

After the searching phase, the user is able to specify the granularity of the content of interest. This is done through XML. Given a query, our software retrieves, from each original XML document, the minimal element containing all the tags of the document that satisfy the query. From this, the user will extract and compose the most meaningful piece of information he needs. More details on the control over the granularity of the content retrieved can be found in [14].
At this point, we should remark that ContentP2P is still in a prototyping phase, and only some of the requirements above have been addressed. Namely, in ContentP2P:

- we assume that all information to export is already stored in XML files. The data is then classified through Dublin Core metadata and transferred to the repository.
- requirements in sections 2.2 and 2.3 have not been implemented yet.
- the response to a query is a new XML file obtained by dynamically assembling contents from different XML sources of any peer of the community.
- the repository consists of a collection of all XML files located on the peers of the community, hence each peer controls and manages the content it has exported.
- the requirement in section 2.5 follows from using XML. In fact, the retrieved content can be presented to end users according to the model outlined in section 3.1.3.
- our idea on the choice of content granularity has been implemented successfully.

3 Architecture

In this section we describe the architecture of ContentP2P, in particular we emphasize the novelty and the impact of using peer-to-peer communication protocols with the architecture of a content management system.

3.1 Modular view

By looking at the set of functionalities to be fulfilled by our system, we have distinguished three essential components that naturally map onto a layered architectural style. 

At the bottom level there is the set of protocols that regulate connection among peers, manage communication and message routing and implement other low level functions. The central layer comprises services such as content indexing, searching, and sharing. The higher layer deals with the presentation of XML "pieces of information", available from every peer, and dynamically assembled to form a logically coherent page created after the user’s request. The set of XML files forms a distributed database that is queried through a specific query language for XML documents, namely Quilt [19].

Our choice of a layered architecture derives from the need for modularity and aims at easing the reuse of existing tools and enabling technologies. Indeed we have been able to use different tools and technologies to develop the layers. In the following subsections we describe the three layers and discuss the technologies.

Fig. 1: ContentP2P Architecture

3.1.1 The communication level

The communication mechanism in a distributed application is composed by a number of basic operations common to all architectural types, such as the request for connection between two nodes of the network and the opening of a communication channel. The substantial difference among the several peer-to-peer paradigms consists on the management policy in the peers’ community, which could be characterized by the lack of a server, or the presence of a central server that fulfils all client coordination functions.

Our application uses the communication protocols of JXTA platform that allow each peer to operate both as a client and a server. Peers use the JXTA protocols to advertise their resources and to discover resources from the network, such as pipes and services, available from other peers. The JXTA protocols are specified as a set of XML messages exchanged among peers, so different kinds of peers may participate in a protocol.

One important feature of the JXTA platform is the concept of peergroup, that is a group of peers offering a specific set of services. Peergroups form logical regions whose boundaries limit access to the resources of the group. Our aim is that of creating a new peergroup within the global community of JXTA peers, called ContentGroup and defined by offering the set of services specified in ContentP2P.

For a deeper insight on JXTA technology, we refer to [1].

3.1.2 The data management level

This layer is concerned with content classification and search. Our system offers the following services:

- content insertion and validation;
- information retrieval.

To this aim, we have programmed suitable forms where a user inserts the necessary information to write the request he wishes to submit.

The data management level is the main part of the application as it contains the business logic. In fact, this part is responsible for the construction of the
ContentGroup community, based on the network configuration services of the JXTA platform. The capability of querying a global area of information that changes continuously depending on the peers connected at a certain time is provided by the JXTA platform, in a way transparent to this layer.

The data management levels of the various peers interact with each other in several moments:

- during registration to the content sharing service;
- when a query must be forwarded by a peer to the other peers of ContentGroup;
- when the result is sent back to the requesting peer.

During the registration phase of one peer, its presence is notified to the other connected peers of ContentGroup. During the content creation phase, a descriptive form (metadata) of the new content to be published is filled in by the user and is added by this layer to a XML file containing metadata related to all XML files of the local repository. Metadata allow for document identification and keep track of the documents in the repository. Content classification is obtained by means of the Dublin Core Metadata Element Set [5], which is a recognized standard for the description of metadata. It defines 15 elements, such as: Title, Creator, Subject, Description, Publisher, Contributor, Date, Type, Format, Identifier, Source, Language, Relation, Coverage, and Rights. In case the new document is already present in the local repository, the user can modify it and the old document is automatically updated.

During the searching phase, the local data manager converts all data inserted by the user into a XML query language query (Quilt in our application). This query is both processed on the local repository and sent to remote peers. From the query execution on the metadata file of each peer, a certain number of URLs of XML files of the repository are retrieved. Each of these files is then processed by a new query to retrieve the requested content.

3.1.3 The data presentation layer

This layer addresses the requirement of content separation from its presentation style. It gathers all data inserted by the user in order to formulate two Quilt queries (one on metadata and one on content), and subsequently deals with the presentation style of the answer. The resulting content is an XML document that will be viewed as a tree with all minimal elements responding to the query highlighted through colors. More specifically, we color these nodes as follows:

- the root node of the subtree of interest is highlighted by the blue color, to mean that it includes all elements, of the original XML document, containing at least one of the searched words;
- all nodes, internal to the above subtree, that satisfy the query will be highlighted with green color, to mean that they may be selected and picked up individually;
- instead, all nodes colored by red cannot be selected individually, as they satisfy the query only partially. This happens if the query logical expression contains the AND operator.

For content search and retrieval, our system uses Kweelt [2], an evaluation engine for the Quilt query language. In order to implement the colouring model outlined above, we had to access and manipulate the internal tree representation of an XML document. This required reverse engineering techniques to locate and understand the classes to be modified.

4 Example

In the following we illustrate, through an example scenario, the basic functionalities of our system, namely we describe the phases of content classification and search, with the help of one of the graphical interfaces we have made for our first prototype implementation. At present, ContentP2P consists of three separated programs implementing distinct sets of functionalities of our system. Namely, there are three components (see Fig. 2):

- a server (named S in the figure), which deals with the services of searching on the local repository and responding to the user’s queries;
- a querying client (CReq), which allows users to formulate their queries, forward the queries to the known peers, and presents the results to the users;
- a utility to manage the local repository (CDB), which enables users to add new XML files to the repository and update the metadata.

With this distinction, the user is given the possibility to choose among the roles of content creator, provider, or consumer. A consequence of this choice, compared to that of making a single program implementing all functionalities of the system, is that distinct peer-profiles have to be defined when starting ContentP2P.

![Fig.2: Interactions view.](image-url)
As we have already mentioned, content classification is done via metadata, that is a XML file containing some descriptive information about the content to the published. In a peer-to-peer environment, the number of potential servers might increase drastically so that searching information can become a complex problem. Through metadata, ContentP2P provides publishers with formalism to describe the local repository that one peer makes available to the other peers of the community. The client supports two kinds of content search, a basic search based on keyword and an advanced search that exploits metadata and Boolean expressions on the content. As an example, Fig. 3 shows the advanced search interface.

**Fig.3: Advanced Search interface**

The form is mapped onto two Quilt queries, one acting upon metadata and the other dealing with the actual contents. The queries are both processed by the local instance of Content2P server, in case this is active, and sent to other server peers.

Accordingly, the content retrieval action consists of two phases:
1. process of metadata in order to extract a list of all URLs of XML files responding to the first query.
2. query on the content of each of the files above.

In the case of a basic search, the first query is not present and the second step involves all the files in a node repository.

The result of these actions from every server peer contacted consists of the following set:

a) metadata attached to a selected file whose interrogation turned out to be successful;

b) the minimal element of the XML file covering all cases that satisfy the query logical expression;

c) the tag names of the elements of the XML file containing the element of point b), written in the same order as they appear in the file.

The attached metadata information on the XML file is useful to the user as he can make a first quick selection on all retrieved contents.

The element of point b) contains, in addition to the elements responding exactly to the query, also the other elements needed to complete the description of the context being returned.

Finally, point c) specifies the depth of the retrieved content inside the original XML document. This information might turn be useful to keep track of the content position in the document.

All responses coming from both local and remote server peers are collected to form a XML file that is visualized as a tree through a browser. The user can select among all XML documents that have been retrieved being able to choose the content granularity.

### 5 Conclusions and future work

Our software has several qualities. One follows directly from our choice of a three-layered architecture, which allows us to use existing products for each layer and confers more flexibility to our system. All code is written in Java, so that the application is portable. Moreover, our system is easy to maintain: it is highly modular, hence separated testing of each component is possible.

In our first prototype of ContentP2P we have only handled the basic aspects of content search and retrieval. The prototype needs further refinements with respect to robustness and stability issues. These are also a consequence of the fact that we have adopted emerging technologies. In this respect, we have used version 1.0 of JXTA platform which is very recent (it is available from Sun’s website since end of April) and whose new improved versions are rapidly published.

Our next concern will be that of testing the configuration for a JXTA peer to be rendezvous node, which is a special peer who keeps in memory a list of addresses of other peers with whom it has been connected. This property, available within the JXTA package, will improve the system performance in the case of a wide network. In fact, a peer rendezvous may be contacted by any peer in its booting phase and could also be used for “intelligent” routing of requests depending, for example, on the subject of the query.

Furthermore, we are considering a solution for the off line content management requirement. In a future release of ContentP2P, in fact, the part of the utility to manage the
local repository will be substituted by a content publishing client that will be able to connect to any server peer of the ContentGroup, and remotely access its repository in order to upload or update content.

6 References

http://www.jxta.org/
http://db.cis.upenn.edu/Kweelt/
http://www.dis.uu.se/~udbl/publ/hui_xml.pdf
http://dublincore.org/documents/dces/
March 1999
[7] Documentum 4i XML Initiatives
http://www.documentum.com/products/content/xml_initiatives.html
[8] UserLand Frontier
http://frontier.userland.com/
http://www.sorman.se/products/cms/index.asp
http://www.vignette.com/
[19] “Quilt: an XML query language”