Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

David Hausheer, Burkhard Stiller

Design of a Distributed P2P-based Content Management Middleware

TIK-Report
Nr. 156, January 2003

Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

David Hausheer, Burkhard Stiller:
Design of a Distributed P2P-based Content Management Middleware
January 2003
Version 1
TIK-Report Nr. 156

Computer Engineering and Networks Laboratory,
Swiss Federal Institute of Technology (ETH) Zurich

Institut für Technische Informatik und Kommunikationsnetze,
Eidgenössische Technische Hochschule Zürich

Gloriastrasse 35, ETH-Zentrum, CH-8092 Zürich, Switzerland

Design of a Distributed P2P-based Content Management Middleware

David Hausheer[1], Burkhard Stiller[2,1]

[1] Computer Engineering and Networks Laboratory TIK
Swiss Federal Institute of Technology, ETH Zurich, Switzerland
[2] Information Systems Laboratory IIS
University of the Federal Armed Forces Munich, Germany
[hausheer|stiller]@tik.ee.ethz.ch

Abstract. There is an emerging need for Content Management Systems (CMS) enabling collaborative development, administration, and distribution of large amounts of content, in particular web content, which enfolds an increasing number of different types and formats of content that is made accessible over the Internet. Most of these systems are currently based on a client/server architecture, where content storage and management are performed centrally on a dedicated server. While such centralized systems simplify the management with respect to data consistency, security, and accountability, they usually lack scalability and reliability. Pure peer-to-peer (P2P) network architectures are based on the assumption that no central server exists and has to be relied on, since every peer can (or rather needs to) participate in the provisioning of core system functionality that would be provided otherwise by a single server. A P2P-based CMS scales much better, because available system resources increase linearly with the number of participating peers. Moreover, the reliability of the system can be made very high, since the degree of system redundancy can be increased without much effort. However, compared to the centralized approach, conventional CMS tasks, such as versioning as well as access control and accounting, become more complicated in a distributed environment with potentially many replicas of the same content. This paper proposes a completely distributed P2P-based content management system in support of access control and accounting while taking into account key aspects following from the nature of distributed systems.
Keywords: Content Management, Peer-to-Peer, Access Control, Accounting, Content Distribution, Collaboration

1 Introduction

The entire amount of digital information in the world represents a huge distributed and loosely connected information system which is growing very fast on a daily basis. A lot of this information is available as content in the World Wide Web and can be accessed over the Internet. The maintenance of this huge amount of distributed content is a difficult task that already can be challenging for only a small part of the system, *e.g.*, the information of a single company [13]. Content Management Systems (CMS) determine a valuable means to simplify the creation and maintenance of content, *e.g.*, WebDAV for web pages [27] or CVS for programming code [11]. The major set of tasks for these

systems are thereby versioning control, access control, and transparent data storage. Moreover, by means of concurrent versioning mechanisms they enable collaborative work. During the last few years a set of new frameworks have been developed focusing specifically on Web Content Management (WCM), *e.g.*, the Python-based Zope [30] or the Java-based Cocoon [10]. These powerful WCM frameworks enable fast deployment and simple maintenance of large web sites, while supporting versioning and access control, workflow management, and other CM functionalities.

Most of these systems, however, are entirely based on a client/server architecture where data storage, access control, and management tasks are performed on a central server, that can be accessed by all clients. This works well for a small static amount of content and clients, where the need for hardware resources can be determined very well. However, for a large and highly fluctuating demand of resources it does not scale anymore. Moreover, if for any reason the central server fails, the entire system would not be available anymore.

In a pure peer-to-peer-based network, in the absence of any central server, the content as well as its management and provisioning functionality are distributed and replicated among several peers. This approach has a number of benefits: First, it enables the deployment of suitable load balancing mechanisms and removes the potential bottleneck of the central server. The more content and system functionality is replicated on several peers, the higher the overall system reliability grows. In addition, a P2P-based content management system is much more scalable and robust, as the load caused by a participating peer can be compensated with the resources provided by that peer.

On the other hand, the P2P approach results in a set of challenges with respect to the key aspects of distributed and replicated content that a distributed content management solution has to deal with. Major problems being identified are replication consistency, search, access control, and accounting. While distributed search and replication are currently widely investigated, access control and accounting, which are crucial mechanisms with respect to digital rights management and content charging, are yet in its infancy in P2P systems.

This paper proposes a distributed P2P-based content management system dealing with the maintenance and distribution of high-value content in P2P networks. High-value content requires the deployment of suitable protection and charging mechanisms. The presented system focusses on these specific requirements and tackles the challenges of P2P systems mentioned above. The approach proposed is based on a P2P services architecture [14] which is currently being developed within the MMAPPS project [20].

The remainder of this paper is organized as follows: Section 2 details and discusses the requirements for the proposed system based on different content management scenarios. Section 3 introduces the generic content management architecture presenting the content model and those components of major functionality. Furthermore, it sketches the adopted P2P services architecture and summarizes key design issues. While Section 4 offers the design and implementation of the proposed distributed content management system, its evaluation on the basis of a specific scenario is discussed in Section 5.

2 Scenarios and Requirements

Content management systems are used for different purposes being referred to here as content management *scenarios*. Each of these scenarios has different requirements on functionalities and their levels of strength that need to be provided by the system. These requirements are affiliated with properties, which differentiate types of content and content management scenarios.

2.1 Content Properties

Popularity. Content can range from popular to unpopular. The more people are interested in specific content, the more popular it is. The latest album of a famous musician is a good example for popular content, while most of those emails being sent might be very unpopular. The popularity of content has a number of impacts on content management systems. Popular content will be acquired very frequently and, therefore, will cause heavy loads on the provisioning infrastructure. Thus, there is a strong need for content replication and appropriate load balancing as well as scheduling mechanisms, which are able to distribute the load among caches or replicas of the same content in a fair manner. Moreover, scalability of the provisioning infrastructure is an important requirement, especially, if size and popularity of specific content varies heavily.

Topicality. Content can remain unchanged for a long time (which does not necessarily have to be negative) or be changed very frequently. In the extreme case, content might be generated dynamically, *e.g.*, based on a user profile and interaction (content personalization). The topicality might be more or less crucial in a specific scenario. In some cases, where content changes quite frequently, *e.g.*, in an on-line newspaper, it may not be a tragedy, if one gets a slightly older version of the content. However, often it is important to be able to access the latest version, *e.g.*, in a collaboration scenario. Thus, the required level of consistency among several potential replicas of content has a strong influence on appropriate versioning control mechanisms in content management systems.

Quality. Content can be sensitive with respect to quality aspects of the content distribution service, *e.g.*, latency, bandwidth, or jitter. Depending on these sensitivities a content management system faces different QoS requirements that will lead to the allocation of enough resources for content provisioning and distribution. A content-based approach that deals with resource provisioning for multiple levels-of-service can, *e.g.*, be found in [17].

Value. Content can have a large or a small value. The value of content depends on people's valuation to the content, *e.g.*, depending on their interests and the reception of its quality. Usually the value of content can be represented by an according monetary price. [18] describes a content valuation model, which dynamically determines the optimal price for content based on customer behavior. Thus, the content value has again a strong impact on the content management system. Content to be charged for needs accounting support and has to be protected appropriately by means of access control and other suitable digital rights enforcement mechanisms. While free content does not necessarily

need charging, accounting and access control might still be necessary for monitoring and protection purposes.

Security. Content may have a strong need for protection based on different levels of security. Large-value content needs to be protected against unauthorized access. The same holds true for content, which needs to be kept secret for other reasons. Different security properties may also apply for read, write, and other form of access to content.

Reliability. Content can be sensible in terms of reliability, i.e., service availability and content persistency. It may be tolerated, if a personal homepage is only available 90% of the time. However, in other scenarios an availability and persistency of as close as 99.999% is required. Therefore, content has to be replicated and appropriate consistency mechanisms are required.

2.2 CMS Scenarios

During its life cycle content usually experiences several different phases, each of which addressing particular aspects. Figure 1 depicts those three main phases, which can be observed in many scenarios:

- collaboration,
- management, and
- distribution.

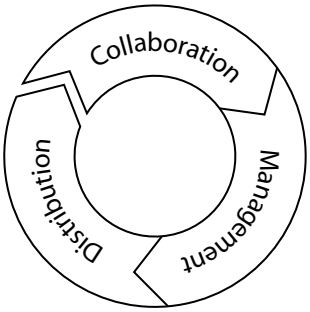

Figure 1: Phases in a Content Life Cycle

As indicated, a content life cycle can actually be represented as a closed loop since previously distributed content might influence or be used in new collaboration scenarios. In general, a CMS has to deal with diverse requirements derived from content properties in each of these phases. However, a clear observation determines that a single CMS rarely covers all phases together. Usually there exist individual solutions covering only parts of one or two phases. For a complete support of content these specialized CMSs have to interoperate with each other, *e.g.*, by applying a standardized scheme for content exchange.

A set of three different content management scenarios are described below, which have been selected to represent each of the three content life cycle phases presented. As shown in Table 1 these different scenarios are evaluated according to those content

properties being outlined in Section 2.1. Thus, specific requirements of each scenario are elaborated separately.

Table 1: Evaluation of Properties for Different CMS Scenarios[a]

	Collaborative Content Development	Corporate Web Content Management	Multimedia Content Distribution
Popularity	--	+	++
Topicality	++	0	-
Quality	0	+	++
Value	--	-	++
Security	+	0	++
Reliability	++	0	0

a. The notations --, -, 0, +, and ++, respectively, relate to the importance of a content property in the according scenario.

Collaborative Content Development. This scenario focusses on the content creation process within a small group of people. A CMS can help to perform a reliable archiving and versioning of content, which is transparent to users. Such collaborative development is bounded usually to a small group, i.e., the popularity of such content is small in that sense. The topicality on the other hand is very high, since during a creation phase, content will change very frequently. Quality-of-Service (QoS) aspects are normally not an important issue in such a scenario, however, short latencies and high bandwidth to accelerate processes are of interest. The content value might in fact be quite high, however, since selling is normally not yet an issue, accounting is usually not necessary at this stage. In addition, security mechanisms are of importance, if access to the content needs to be restricted to a specific group of people. Therefore, there a strong need for distributed versioning, archiving, and access control mechanisms exists.

Corporate Web Content Management. This scenario deals with the maintenance of large corporate web portals. A CMS helps to administer the structure, design, and data of such web content separately. Moreover, it supports reuse of content and the definition of workflows for content editing and publishing, while it hides the underlying complexity for such processes to users. In such a scenario the content popularity might vary a lot, so the content distribution infrastructure needs to be very flexible in order to be able to react on variations of the current load. Unless content is generated dynamically based on the interaction between users and providers, *e.g.*, for the personalization of content, the topicality is not so important, at least it might not be necessary to keep content replicas globally consistent. Although content charging is usually not an issue in such a scenario, quality aspects might still be important for corporate identity rea-

sons, and accounting could be required for monitoring purposes. Content protection is important with respect to its modification, however, apart from that, any other security mechanisms are usually not required. Finally, reliability, although normally not required, determines an optional add-on, since the 99.999% availability of a corporate's web page is important for the corporate's identity. Therefore, there is need for scalability and QoS, as well as a minor need for security, accounting, and distributed versioning mechanisms.

Multimedia Content Distribution. In this scenario the focus is on distribution of large, high-quality, and high-value multimedia content, such as audio and video streams. A CMS can support the maintenance and distribution of such content in many different ways. The popularity of such content is normally very high unless, *e.g.*, a multimedia conference between two parties is considered. With respect to topicality, once the content has left its source, it is usually not changed anymore. The most important requirements are the compliance of QoS parameters, and, for high-value marketed content, the accounting and protection of content access throughout the entire content distribution path. Therefore, scalability, QoS as well as accounting and security mechanisms are required.

Each of these scenarios can be found in a number of existing systems. While WebDAV [27] and CVS [11] are examples for the collaborative content development scenario, Zope [30] and Cocoon [10] exemplify the corporate web content management scenario. Finally, multimedia content distribution technologies can be found, *e.g.*, in Akamai's EdgeSuite [1] or NextPage's NXT3 [23].

3 Architecture Models and Design

Based on such real-world CMS scenarios and their requirements as discussed above, building blocks of a Content Management Architecture (CMA) in terms of a set of generic models have been developed. Flexibility of the new CMA model is the key, mainly to be able to address any scenario's requirements. While those generic content management models being presented below are independent of any underlying network architecture (client/server, peer-to-peer, or hybrid), a dedicated peer-to-peer service architecture is used to design the distributed content management system, which is presented in Section 4.

3.1 Content Management Architecture

Key building blocks of the generic CMA are focused on two main models, a *content model* and a *content management model*. While the content model describes the "passive" part of content, such as its structure, format, and meta-information, the content management model represents the "active" part, i.e. the set of mechanisms or functions, which operate on the underlying content repository, *e.g.*, by adding, manipulating, or removing content.

3.1.1 Content Model

For an accurate view on a generic CMA it is necessary to specify what content is. In the literature many different views and definitions of content and content manage-

ment exist. *E.g.*, a detailed description of the problem field can be found in [4]. Following those definitions [4], content is considered as “user-friendly” information in a specific context. In contrast, data represents “computer-friendly” information, encoded as a set of bits or bytes. Data is usually of little use outside of any context, thus content, or rather a CMS, combines this raw information and determines its semantic. Thereby, necessary context information can be provided usually by meta data, i.e. information which describes the context by a set of property values or keywords. [5] is an approach that deals with context exchange in a distributed environment.

This fundamental distinction between content and data is crucial for many CMS mechanisms, which will be discussed in more detail in Section 4. From a user perspective, accounting may be reasonable only on the content level, while computers usually can provide accounting on the data level only. Therefore, it is essential to find a mechanism enabling the mapping of data-level accounting records into the content-level.

To illustrate the difference and relation between content and data, Figure 2 defines an entity-relationship model with its main building blocks of multimedia content broken down to bits representing data[1]. Multimedia content is a combination of different types of content, such as text documents, images, and audio/video streams. Each content type may be encoded in a variety of formats, such as gif, jpeg, or png for images. Depending on the content type and its format, different atomic data types may underlie the content, *e.g.*, an image that could either be based on pixels or vectors. In any case, content is always based on bits and bytes. By looking at a single data bit it is not possible to find out what type of content it actually represents.

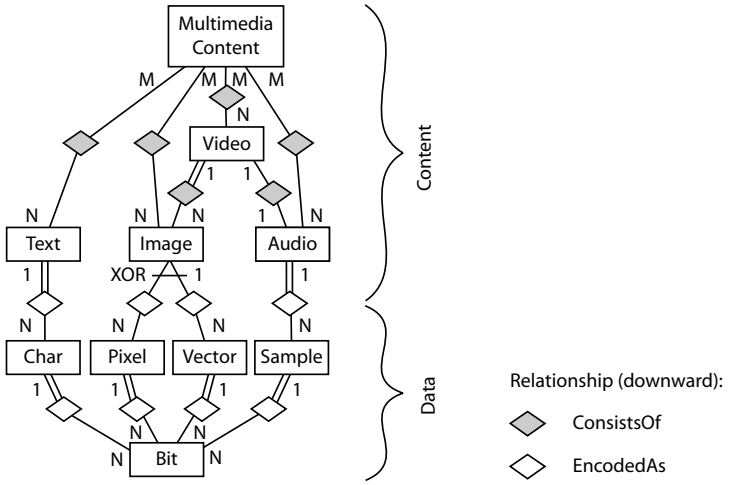

Figure 2: From Content to Data

As a result of this content variety, a CMS has to deal with a large number of different encoding and representation formats. A common solution to this problem today is the use of XML-based formats, which can be transformed into other formats on demand

1. A combination of those building blocks is discussed in more detail in Figure 3.

using different style sheets. For simplicity reasons and without loss of generality, it is assumed that content is formatted in a XML-based language. Furthermore, it is assumed that binary data could be encapsulated in XML using a Base64 encoding as specified in MIME [12].

As content is embedded in a specific context, a precise boundary of content can not be given. Therefore, it is difficult to map content onto individual data files without loosing context information. However, to store or deliver content it is necessary to divide content into separate parts. Multimedia content as a combination of several different content types supports this requirement. The proposed content model (cf. Figure 3) addresses this problem. It describes the overall structure of content, which contains a number of embedded content objects. A content object can be atomic or serve as a container for other content objects itself. Atomic content objects are the smallest possible pieces of content, that a content management system can deal with. Within a container objects may be ordered in a designated sequence. *E.g.*, consider a book with a number of chapters, which are intended to be read in a specific order. Alternatively, content objects might be linked together which allows a user to navigate through the content himself.

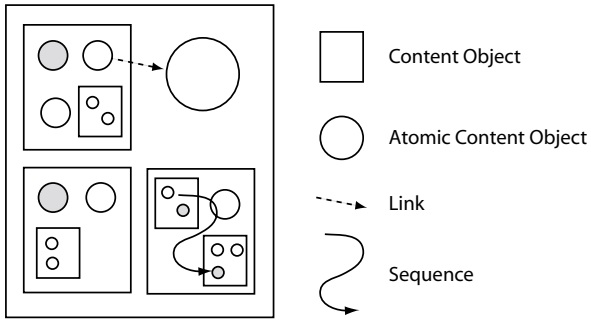

Figure 3: Content Model

A content object could appear simultaneously in different contexts, but actually may be stored in the same place as represented by shaded objects in Figure 3. However, content objects do not necessarily have to be mapped 1:1 into data files. In the most extreme case, if the file size is fixed, like the page size in a book, content objects and data files may not overlap at all. This content model represents the structure of content independent from the structure of any underlying storage and network infrastructure. Content could either be stored in files, databases, or other data storage environments. Moreover, any underlying network topology is hidden in the content model. Therefore, it is irrelevant, whether individual content objects are stored centrally or distributed. A similar hierarchical model for content-aware management can be found in [9].

3.1.2 Content Management Model

Content management can be described by a set of functionality and interfaces that need to be provided on behalf of one or several users to create, maintain, and distribute content on a *content base*, i.e. a repository which is able to host any type of content in a reliable and efficient way.

The abstract model of a generic CMA (cf. Figure 4) assumes implicitly that components of the CMS are distributed. A user can access the CMS by means of a user application, *e.g.*, a web browser, which interacts with the CMS interface and deals with the audiovisual representation of the content.

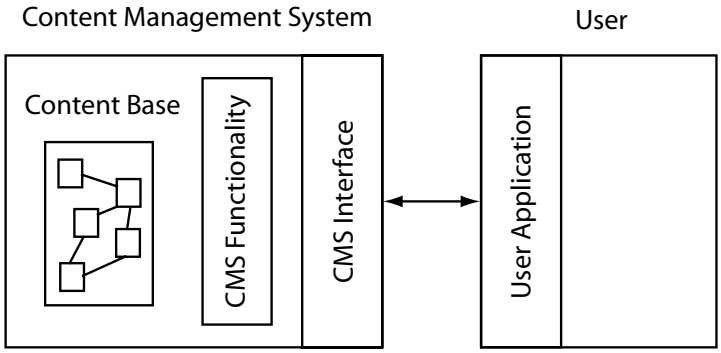

Figure 4: CMA Model

Such a generalized CMS can be considered as *content management middleware* in the sense that it provides a standardized interface abstracting away from the complexity of the system. Moreover, the approach on a detailed content storage in the content base is hidden from the user, which enables the user application to be implemented independently of any specific underlying platform. In the following the list of key CMS interface methods and functionality components is given and several options for content bases are outlined.

CMS Interface Methods. This API specification aims at a standardized semantic for basic methods describing the access of content by a user.[1] The list of methods are intended for a high-level interface description abstracting away from implementation technologies, such as Java RMI or any other specific protocol. The syntax of those methods follows the common form of *returnValue = methodName (parameters).*

- *id = putContent (content, userId)* enables a user to upload content from the local content repository into a CMS content base. In case that the local content repository is physically already part of the CMS content base, as it will appear in a distributed CMS, *putContent* would denote the ability by a user to select specific content for sharing with other users through the CMS. As a result a user will get back an identifier, *e.g.*, a hash code or an URL, which uniquely identifies the content and could be used later as a key for accessing it. [8] presents a content addressing scheme for HTTP which uses SHA-1 hashes as URNs.
- *content = getContent (id, userId)* enables a user to download or read content from a CMS' content base. Thereby the content might either be streamed or download-

1. More sophisticated CMS interface methods being necessary for a further improved CMS functionality are not covered within this paper's scope.

ed as one or several files. Download does not necessarily mean that content will permanently be stored locally, it could also just be cached temporarily during its rendering. However, a distributed CMS might benefit from local replicas, which could immediately be accessed by third-party users.

- *boolean = removeContent (id, userId)* enables a user to remove content from the CMS content base. This determines the reversed method to the *putContent* method. This does not necessarily mean that the content is really deleted, it might just not being shared through the CMS any more. The actual deletion may be performed otherwise, *e.g.*, locally through the operating system or by an additional CMS method. The boolean return value indicates whether the action has been performed or not.
- *boolean = modifyContent(id, patch, userId)* enables a user to change content, *e.g.*, by applying a patch, i.e. the differences to the content, which has previously been uploaded into the CMS content base. The boolean return value indicates the result of the action.
- *id[] = searchContent(metadata, searchType, userId)* enables a user to search for content matching with a specific meta data description. This method also allows for different types of search, such as basic keyword-based lookup or sophisticated best-matching search. Furthermore, browsing in an index list or in metadata-based categories is supported by the CMS interface. As a result a user gets a list of content identifiers, which match with the according search query.

CMS Functionality Components. The following set of functionality is required to be provided by a CMS. Specialized CMS', which focus on a specific CMS scenario as exemplified in Section 2.2, may only support a subset of these components depending on given requirements. The CMS functionality components and their dependencies are illustrated in Figure 5.

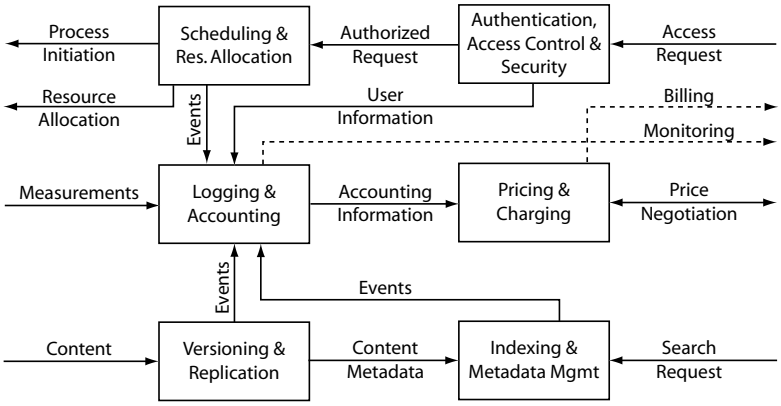

Figure 5: CMS Functionality Components

- *Logging and Accounting Support:* This functionality includes metering and mediation of all types of transactions and method invocations, including usage of

functionality components. It combines this information with the user information collected by the authentication component to perform logging and charging and to provide monitoring support if necessary. Accounting support will be detailed in Section 4.

- *Pricing and Charging Support:* The pricing and charging functionality comprises valuation and pricing of content, as well as charge calculation and billing mechanisms. The pricing component sets and disseminates the price. It interacts with the user application to negotiate a dynamic price with the user. The settled tariff will be used by the charge calculation to combine the accounting information with the specified price to calculate the according content charge.
- *Scheduling and Resource Allocation Support:* Scheduling is needed to process any CMS method invocations as well as to perform the allocation of resources in a fair manner. Different scheduling methods are in place, such as round-robin scheduling, priority-based scheduling, shortest-job-first scheduling, or even techniques like earliest-deadline-first mechanisms that support real-time applications such as audio/video streaming.
- *Authentication, Access Control, and Security Support:* This functionality includes a set of mechanisms meeting various types of security requirements, such as privacy, access protection, integrity, authenticity, and non-repudiation. Different levels of security are supported, *e.g.*, from public and anonymous to private and authenticated. In particular, access control requires the authentication of users, *e.g.*, implemented by a shared secret, and it might authorize the access of content based on this information. These mechanisms are detailed in Section 4. Furthermore, suitable encryption and signing mechanisms might be adopted to meet further security requirements.
- *Versioning and Replication Support:* Versioning keeps track of different versions of the same content, which has been modified by one or several users. In addition it enables the provisioning of personalized content, *e.g.*, content in different languages, different quality levels or different formats. In a distributed environment with replication, versioning grows even more important, as it has to ensure that modifications on content are performed consistently over several potential replicas of the same content.
- *Indexing and Meta Data Management Support:* Indexing manages a list of available content, which has previously been uploaded and may now be shared through the CMS. Such a list could either be allocated centrally or distributed. This functionality might further support the attachment of meta-information to content, which it could extract automatically from content or receive it from the user through additional interfaces. Options on attaching meta information to content are discussed just below.

In addition, further optional management functionality might be supported by a CMS, such as mass operations or workflow management.

CMS Content Base. The content base should efficiently and reliably store the content in a way which is transparent to the user. In particular, content might be stored centrally or distributed depending on the requirements of a specific application scenario.

Several storage options are discussed in [7]. The major problem on attaching meta-information to content is performed differently for each option.

- *File-oriented:* The content is stored in files on a local or distributed file system. Meta-information, which might not be supported by the file system, can be provided within the file, e.g., as meta-information tags or it needs to be stored separately.
- *Object-oriented:* The content is encapsulated as an object, which is stored in an object-oriented database. Meta-information can be attached directly to objects.
- *Relational:* The content is stored as data in two-dimensional tables and is assembled dynamically upon request using logic expressions. Meta-information can be stored in separate fields within a table.

Apart from these options other approaches are discussed in [7] such as, *e.g.*, index-sequential files, which are not covered within this paper.

3.2 Peer-to-peer Services Architecture

The aim of the CMA presented above was to abstract away from any underlying infrastructure or network topology. The design of the Distributed Content Management System (cf. Section 4) is based on SOPPS, a service-oriented P2P architecture [14], [26]. SOPPS has been developed within the MMAPPS project [20]. It describes four different models, a use model, a network model, a peer model, and a service model covering the most relevant aspects of a P2P services architecture. While the network model considers transparent end-to-end connections between different peers, the service model illustrates the use of peers and the network to provide services. An other approach for a P2P service architecture can be found in [16].

3.2.1 Peer Model

The peer model of SOPPS describes the internal structure of a peer, which is composed of four layers as shown in Figure 6. Each peer provides several interfaces allowing networked peers to interact with each other. The following list briefly outlines major characteristics of these different layers.

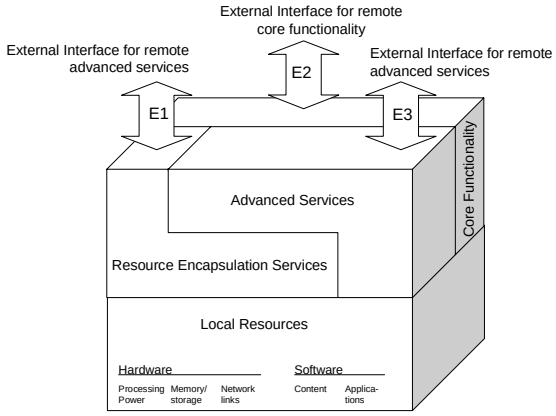

Figure 6: SOPPS Peer Model

- *Local Resources:* Each peer has a limited number of hardware and software resources. Hardware resources, such as processing power, memory or storage, and network links, cannot be copied or transmitted. In contrast, software resources, such as content and applications, can be duplicated and transmitted through the network. It is assumed that both types of resources are offered to other peers. It is important to note, however, that software resources always draw on hardware resources, *e.g.*, for their storage and distribution.
- *Resource Encapsulation Services:* To resolve the duality between resources and services, resource encapsulation services encapsulate access to local resources through the provisioning of standardized interfaces. Drawing directly on local operating system functionality, resource encapsulation services provide access to local resources for local as well as remote advanced services.
- *Advanced Services:* Advanced services provide more than just a standard interface to local resources. They typically involve several resources to work together to provide an advanced service. *E.g.*, consider an image comparison service taking two pictures, comparing one with another, and yielding a degree of similarity as its output. Such a service builds on several local resources, i.e. the image content, storage resources, and an application providing the algorithm for the comparison. In the most extreme case these resources could all come from resource encapsulation services on different peers.
- *Core Functionality:* This layer provides the functionality needed to uphold peer network services. It is in charge of basic functionality, like access control, accounting, and service management as well as protocols for distributed functionality. In particular, it includes those functionality that supports the market management of the system, such as pricing and charging. Core functionality modules may access and cooperate with the corresponding modules on remote peers.

4 Fine Design and Implementation of the DCMS

The design of the DCMS implements the generic content management model introduced in Section 3.1. The architecture design is based on a P2P network infrastructure (cf. Figure 6). Peers participating in a P2P system can offer services and simultaneously use services offered by other peers. The three dimensional illustration in Figure 6 represents peers as slices of a hollow cylinder. Every peer is running parts of a CMS middleware and may, at the same time, make use of this middleware by means of a user application. Middleware and user application are connected through a link representing the underlying P2P network.

Looking at a single peer from the front the design in Figure 6 exactly reproduces the two dimensional CMA model shown in Figure 4 except that the CMS functionality and interfaces on every peer are now part of an overall CMS middleware, which is jointly provided by all peers. In order to interact with each other, every peer has to provide a set of core functionality and interfaces which needs to be the same on all peers. Apart from that peers might provide additional functionality, which can be offered as services to other peers. The content managed within the CMS middleware is stored in a distributed content base, which spans the entirety of all peers. A specific content object is pro-

vided by a single peer or several peers together. The CMS middleware could, *e.g.*, make use of replicas of the same content and distribute parts of them in parallel.

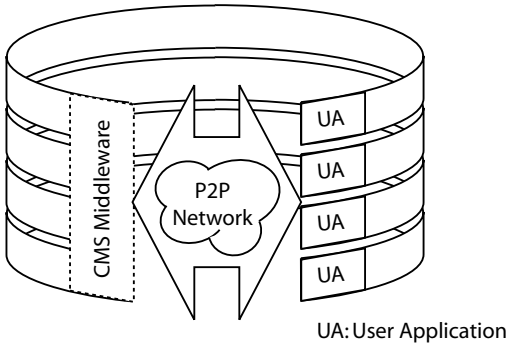

Figure 7: A Distributed Content Management System

The design of the CMS middleware is based on the following assumptions:

- There exists an authentication infrastructure, such as X.509 [15], which is able to setup trust relationships between peers, authenticate providers and users of the CMS middleware, and encrypt content upon need.
- There exists a shared space, such as a distributed hash table or a database, which may be used to store and lookup content meta data, user and group information, accounting information, or other shared data. Currently, many approaches based on distributed hash tables are available for search in P2P systems. Among them are Tapestry [29], Pastry [25], Chord [2], and Content-addressable Networks (CAN) [24].
- There are suitable mechanisms, *e.g.*, voting support or reputation management in place, enabling the enforcement, but not necessarily guaranteeing compliance with predefined rules for content exchanged among the peers.

4.1 CMS Middleware Breakdown

The CMS middleware draws on the peer model. Based on different layers of a peer (cf. Figure 6) the CMS middleware is described as it would be deployed on all peers. In Figure 8 all different components of the CMS middleware are illustrated. Those parts of the middleware, which are labeled in italics, are adopted from the underlying SOPPS architecture. In addition, the CMS middleware introduces two kind of services, basic content encapsulation services and advanced multipart content services.

Content Encapsulation Services. Any collection of raw data being identified as an atomic content object, *e.g.*, a text document, an image, or a video (cf. Section 3.1.1), is wrapped by the instance of a content encapsulation service. The aim of this service is to represent a content object, which may reside on different types of data sources, such as file systems and relational or object-oriented databases. Thereby, data remains in the according data source and is referenced in the service by a pointer to the actual location. Only if the content is requested by another peer the data is de-referenced and appropriately formatted to be distributed.

Multipart Content Services. Those content objects, which serve as a container for a collection of other content objects, such as a web page containing images or any other type of multimedia content, are represented by the instance of a multipart content service. The purpose of this service is to represent the structure of content. Multipart content services draw on content encapsulation services, but they may also access local resources directly to increase performance and flexibility of the middleware using, *e.g.*, the functionality provided by a local file system.

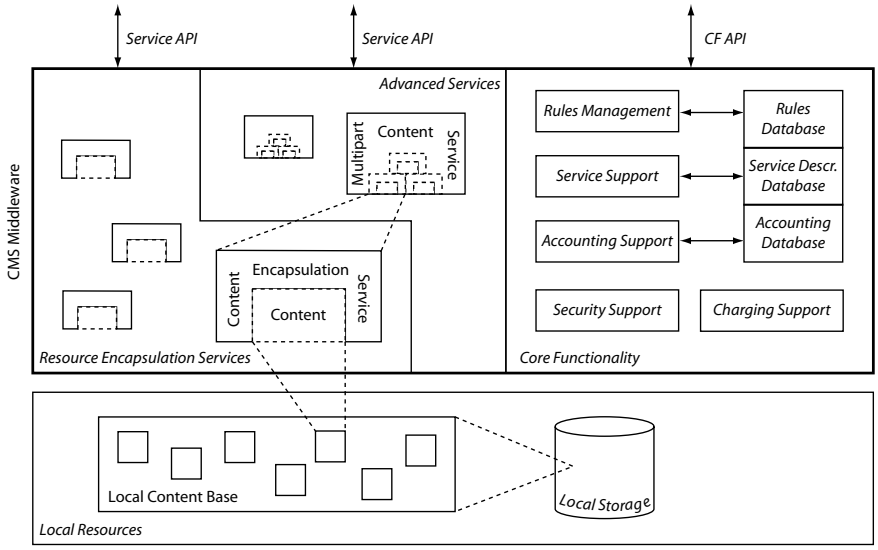

Figure 8: CMS Middleware Breakdown

Both types of services implement a service API, which enables a user application on another peer to interact with these services. All requests are catched by one or several service dispatchers, which activate corresponding services based on the content id and hand on the according request. The service API inherits methods of the corresponding basic service type, i.e. resource encapsulation service and advanced service, respectively, and, in addition, implements CMS interface methods presented in Section 3.1.2. To perform the necessary CMS functionality, such as indexing, versioning control, access control, accounting, and charging, these services make use of a set of core functionality modules. The interaction between services and core functionality is described in detail below, whereas access control and accounting are specifically focussed at. Thereby, the CMS middleware introduces four new data types.

Access Rules. The purpose of an access rule is to specify under which conditions access to a specific content is allowed or denied. [28] presents different security models for content-based access control. Services may use access right information to decide, whether an access request, such as getContent or modifyContent (cf. Section 3.1.2), is performed or not. Conditions are represented as expressions including user information, security or quality level, accounting information from previous requests, payment information, time information or other necessary data. Condition expressions may be

based on the result of other conditions. Hence, a service can restrict the access to content depending on who, when, how often, and under what circumstances makes a request. Information, which is used in a condition, is gathered dynamically from different core functionality modules, or, if provided as a request parameter, such as user information or a content id, sent to the according module which looks up, decrypts, or authenticates the respective information.

Accounting Rules. The aim of accounting rules is to specify under which conditions a service has to log access events, perform measurements of an according transaction, or even attach specific information to a response such as a token. Mojo Nation [21] is an approach that deals with decentralized accounting in P2P networks. Conditions of an accounting rule may basically use the same information as access rules. In addition, an accounting rule could indicate which values have to be measured and it contains expressions specifying how these measurements are aggregated. Depending on the evaluation result of a rule expression, a service may create an accounting record.

Access and accounting rules are created and maintained by the rules management module. For rules storage a local or distributed database can be used, which is accessible by all peers. Alternatively, rules could be stored within a service and transmitted at every request together with the content, *e.g.*, attached as meta-information. If an accounting rule and an access rule use the same set of conditions, according rules are combined to increase the performance of rule evaluations. An other approach which deals with the attachment of rules to services can be found in [3].

There are many different solutions available dealing with access control and accounting on the network/data level. Iptables [22] is a powerful traffic filter which is deployed on Linux-based systems and can be used to perform access control and logging, while NeTraMet [6] is a useful tool which enables to account for any kind of traffic data. Approaches like these could be adopted to perform access control and accounting on the application/content-level.

Accounting Records. Accounting records are created by services and may contain any accounting information as specified in an accounting rule, *e.g.*, content id, user id, delivery time, quality level, or information about usage of local resources. Records are transmitted to the accounting support module, which may store it in a local or distributed database. The charging support module can make use of the accounting records to perform the settlement for the content accessed.

Service Descriptions. Content encapsulation services and multipart content services create a service description specifying necessary content meta information. Service descriptions are maintained by the service support module, which uses these descriptions to reply on specific search or index requests as well as for versioning control and replication management. Alternatively, search descriptions can be stored in a distributed hash table being accessed by any peer. Search, versioning control, and replication will not further be detailed at this point.

5 Evaluation

The proposed DCMS design is analyzed and evaluated based on a typical CMS scenario covering the most relevant aspects of those scenarios presented in Section 2.2. As an example Figure 9 depicts a set of four different peers A, B, C, and D, where all of them operate the CMS middleware (MW) presented in Section 4.1. Respective local content bases and applied user applications (UA) may be different on every peer as long as they support or are supported by the MW. In the example a set of three different content objects are considered, two atomic objects, and a third container object infolding the other two, all of which have not yet been developed at the beginning. This scenario reveals the following operations (cf. Figure 9):

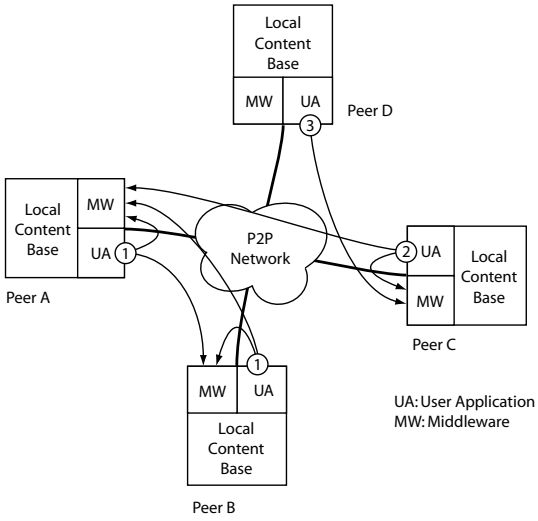

Figure 9: Example Scenario

1. *Collaboration and Management:* Peer A and B start with the collaboration on the content. Peer A creates locally a first version of the two atomic content objects. The UA of peer A uses the local MW interface (*putContent*) to make the content available for peer B. While there are no accounting rules attached to these objects, the access rule for both objects reveals that only peer A and B are allowed to get and modify the content. The same applies for the container object shared by peer B. As peer B now accesses the container object on the local content base, the two embedded atomic objects are retrieved from peer A. If peer A or B modify any of the shared objects, the versioning module synchronizes the modifications with the respective object on the other peer.
2. *Distribution:* As the content developed by peer A and B is settled, they might designate peer C as an authorized distributor of their content. Access rules are therefore changed and peer C is assigned the right to get, but not modify the content. The new accounting rule might determine the obligation to account for all get-

Content requests. An appropriate service description is made accessible for search by any peer.

3. *Re-Distribution:* Peer D might have accessed the service description at peer A (not shown). Peer C sells the content to peer D on behalf of peer A and B. The access of the content by peer C might depend on several conditions regarding, *e.g.*, payment information and previous accounting information. During the transaction the accounting rule triggers the generation of accounting records at peer C.
4. *Settlement (not shown):* On the basis of accounting records generated by peer C, it is possible to calculate the charges, bill the customers, and share revenues between peer A, B, and C. As mentioned earlier, it is assumed that there are mechanisms in place, which may be used to store reliably and eventually prove such sensitive data.

6 Conclusions

This paper presented the design of a Distributed Content Management System (DCMS) based on a generic Content Management Architecure (CMA). The proposed P2P-based CMS middleware implements the CMA in a completely distributed fashion, which forgoes any centralized components. The presented approach is very flexible as it supports different types of CMS scenarios, which content might experience during its life cycle. Requirements gained from a detailed analysis of the different CMS scenarios have been addressed, while in particular, access control and accounting were focussed and investigated.

The generic CMA discusses in detail the content model and the management functionality, independent of any underlying infrastructure for storage or delivery. Considerations taken there, *e.g.*, the clear distinction between content and data, and the presented CMS interface methods are, therefore, also valid for environments other than P2P. A comprehensive list of CMS functionality components tackles those requirements of current and future content management solutions and delineates key issues and design options.

The proposed implementation of the generic CMA based on a service-oriented P2P architecture reveals the feasibility of those concepts on a P2P platform which is currently being developed. Introducing a set of new data types, in particular access rules and accounting rules, the key requirements of a DCMS can be met based on a set of valid assumptions. Finally, the DCMS design can be evaluated sucessfully based on a specific scenario covering the most relevant aspects of any CMS scenario.

Further work will detail accounting, charging, and billing issues on a content-based level. While search and replication are currently widely investigated in P2P networks research, accounting and charging are still at its infancy.

Acknowledgements

This work has been performed partially in the framework of the EU IST project “Market Management of Peer-to-Peer Services” (MMAPPS, IST-2001-34201), where ETH Zürich has been funded by the Swiss Bundesministerium für Bildung und Wissenschaft

BBW, Bern (Grant No. 00.0275). The authors like to acknowledge various discussions and implementation help from J. Gerke and J. Mischke. Furthermore, many thanks go to B. Wicki who discussed and worked at CMS and P2P characterizations.

References

[1] Akamai: *Turbo-Charging Dynamic Web Sites with Akamai EdgeSuite*; White Paper, Akamai Technologies, http://www.akamai.com/, 2001.

[2] H. Balakrishnan, M. Kaashoek, D. Karger, R. Morris, I. Stoica: *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*; ACM SIGCOMM, San Diego, California, U.S.A., August 2001, pp. 149-160.

[3] A. Beck, M. Hofmann: *Enabling the Internet to Deliver Content-Oriented Services*; 6th International Workshop on Web Caching and Content Distribution (WCW), Boston, Massachusetts, U.S.A., June 20-22, 2001.

[4] B. Boiko: *Content Management Bible*; John Wiley & Sons, ISBN: 0-7645-4862-X, November 2001.

[5] M. Bonifacio, R. Cuel, G. Mameli, M. Nori: *A Peer-to-Peer Architecture for Distributed Knowledge Management*; 3rd International Symposium on Multi-Agent Systems, Large Complex Systems, and E-Businesses (MALCEB), Erfurt/Thuringia, Germany, October 8-10, 2002.

[6] N. Brownlee: *NeTraMet: Network Traffic Meter*, Version 4.4, http://www2.auckland.ac.nz/net/NeTraMet/, February 2001.

[7] H.-J. Bullinger, E. Schuster, S. Wilhelm: *Content Management Systeme - Auswahlstrategien, Architekturen und Produkte.* Fraunhofer-Institut für Arbeitswirtschaft und Organisation IAO, WirtschaftsWoche, Düsseldorf, 2001.

[8] J. Chapweske: *HTTP Extensions for a Content-Addressable Web*, Onion Networks, 2001.

[9] K. Cheng, Y. Kambayashi: *Enhanced Proxy Caching with Content Management*; In Knowledge and Information Systems, Vol. 4, No. 2, March 2002, pp. 202-218.

[10] Cocoon: *XML publishing framework*; Version 2.0.4, Apache Software Foundation, http://xml.apache.org/cocoon/, 2002.

[11] CVS: *Concurrent Versions System*; Version 1.11.2, CollabNet, http://www.cvshome.org/, 2002.

[12] N. Freed, N. Borenstein: *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, RFC 2045, November 1996.

[13] GartnerConsulting: *The Emergence of Distributed Content Management and Peer-to-Peer Content Networks*, White Paper, January 2001.

[14] J. Gerke, D. Hausheer, J. Mischke, B. Stiller: *An Architecture for a Service Oriented Peer-to-Peer System (SOPPS)*; Praxis der Informations- und Kommunikationsverarbeitung PIK, Special Issue on Peer-to-Peer Systems, 2003, to appear.

[15] R. Housley, W. Ford, W. Polk, D. Solo: *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*, RFC 2459, January 1999.

[16] J. Hwang, P. Aravamudham, E. Liddy, J. Stanton, I. MacInnes: *Charging Control and Transaction Accounting Mechanisms Using IRTL (Information Resource Transaction Layer) Middleware for P2P Services*; QofIS/ICQT 2002, LNCS Vol. 2511, Zürich, Switzerland, 2002.

[17] S. Jagannathan, K. C. Almeroth: *Pricing and Resource Provisioning for Delivering E-content On-Demand with Multiple Levels-of-Service*; QofIS/ICQT 2002, LNCS Vol. 2511, Zürich, Switzerland, 2002.

[18] S. Jagannathan, J. Nayak, K. Almeroth, M. Hofmann: *A Model for Discovering Customer Value for E-Content*; ACM SIGKDD, Edmonton, Alberta, Canada, July 23-26, 2002.

[19] Project JXTA: *JXTA Content Management System*; http://cms.jxta.org/, 2002.

[20] MMAPPS Consortium: *Market Management of Peer to Peer Services (MMAPPS)*; EU IST Project, http://www.mmapps.org/, April 2002.

[21] Project Mojo Nation: *Peer-driven Content Distribution Technology*; http://www.mojonation.net/, February 2000.

[22] The Netfilter/Iptables Project: *Linux 2.4.x / 2.5.x Firewalling Subsystem*; Version 1.2.7, http://www.netfilter.org/, 2002.

[23] NextPage: *NXT 3 Building a Content Network*; White Paper, NextPage, 2001.

[24] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker: *A Scalable Content-Addressable Network*; ACM SIGCOMM, San Diego, California, U.S.A., August 2001, pp. 161-172.

[25] A. Rowstron, P. Druschel: *Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems*; IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, November 12-16, 2001, pp. 329-350.

[26] B. Stiller, J. Gerke, D. Hausheer, J. Mischke: *Peer-to-peer Services Architecture*; Deliverable 4 to the MMAPPS Project, January 2003.

[27] WebDAV: *Web-based Distributed Authoring and Versioning*; Extensions to the HTTP Protocol, http://www.webdav.org/, February 1999.

[28] E. Weippl, I. K. Ibrahim, W. Winiwarter: *Content-based Management of Document Access Control*; 14th International Conference on Applications of Prolog (INAP), Prolog Association of Japan, Tokyo, Japan, October 20-22, 2001, pp. 78-86.

[29] B. Zhao, J. Kubiatowicz, A. Joseph: *Tapestry: An infrastructure for fault-tolerant wide-area location and routing*; Technical Report UCB/CSB-01-1141, Computer Science Division, U.C. Berkeley, California, U.S.A., April 2001.

[30] Zope: *Open Source Application Server*; Version 2.6, Zope Corporation, http://www.zope.org/, 2002.