

# Design of a Distributed P2P-based Content Management Middleware

David Hausheer<sup>1</sup>

<sup>1</sup>*Computer Engineering and Networks  
Laboratory TIK, ETH Zurich, Switzerland  
hausheer@tik.ee.ethz.ch*

Burkhard Stiller<sup>2,1</sup>

<sup>2</sup>*Information Systems Laboratory IIS, University  
of Federal Armed Forces Munich, Germany  
stiller@tik.ee.ethz.ch*

## Abstract

*There is an emerging need for Content Management Systems (CMS) enabling collaborative development, administration, and distribution of content over the Internet. Many CMS solutions are currently based on a client/server architecture with a central server for storage and management. While such centralized systems simplify management with respect to data consistency, security, and accountability, they 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 peers collaboratively provide content and core system functionality that would be provided otherwise by a single server. A P2P-based CMS scales much better, since available system resources increase linearly with the number of participating peers. In addition, the availability of the system can be raised without much effort, by increasing the redundancy of content across the peer.*

## 1. Introduction

The amount of content available over the Internet is growing very fast on a daily basis and its maintenance determines a difficult task. Content Management Systems (CMS) determine a valuable means to simplify the creation, maintenance, and distribution of content [12]. The major set of tasks for a CMS are versioning, access control, transparent data storage and mechanisms enabling collaborative work. During the last few years a set of new frameworks have been developed focusing specifically on content management for the Web, e.g., Zope [26] or Cocoon [10].

Most of these systems, however, are entirely based on a centralized architecture where data storage, access control, and management tasks are performed on a server which is accessed by many clients. Although, e.g., Zope enables load balancing through a cluster of ZEO clients, it still relies on a central server for storage. This works well for a small or constant amount of content and clients, where the need for hardware resources can be determined very well. However, for large and highly fluctuating demand of resources it does not scale. Moreover, if for any reason the central server fails, the entire system is not available anymore.

In a pure P2P-based network, in the absence of any central server, content and its management 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, since the load caused by a participating peer is 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 CMS has to deal with. Major problems being identified are replica 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 CMS supporting maintenance and distribution of high-value multimedia content using. Such 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 remainder of this paper is organized as follows: Section 2 details and discusses the requirements for the proposed system based on different scenarios. Section 3 introduces the generic content management architecture with those components of major functionality. Furthermore, it sketches the adopted P2P services architecture and summarizes key design issues. While Section 4 offers the design of the proposed distributed CMS, its evaluation on the basis of the presented scenarios is discussed in Section 5.

## 2. Scenarios and Requirements

CMS solutions 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.** The popularity of content has a number of impacts on a CMS. Popular content such as famous music songs is acquired very frequently and, therefore, can cause heavy loads on the provisioning infrastructure. Thus, there is a need for content replication and scheduling algorithms enabling load balancing of requests among different replicas of content in a fair manner. Furthermore, scalability of the provisioning infrastructure is an important requirement, especially, if amount and demand of content vary heavily.

**Topicality.** Content topicality refers to the frequency of changes applied to specific content. Each time content is changed, potential replicas of that content need to be updated accordingly. In the extreme case, if content is generated dynamically, *e.g.*, based on a user profile, content replication is only beneficial if such kind of personalization parameters are supported. In scenarios, where content changes quite frequently, *e.g.*, during collaborative content development, it is crucial to be able to access the latest version of content, while for other scenarios, *e.g.*, the distribution of an on-line newspaper, it may be sufficient to get a slightly older version. Thus, the required level of consistency among several potential replicas of content has a strong influence on appropriate versioning control mechanisms in a CMS.

**Quality.** Multimedia content is sensitive with respect to quality aspects of the content distribution service, *e.g.*, latency, bandwidth, or jitter. Depending on these sensitivities a CMS faces different QoS requirements that 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 [16].

**Value.** The value of content refers to people's valuation to the content, *e.g.*, based on their interests and reception of content quality. To determine the value of content, *e.g.*, represented by an according monetary price, is a difficult problem. [17] describes a content valuation model, which dynamically determines the optimal price for content based on customer behavior. 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. Accounting and access control can further also be used for monitoring purposes.

**Security.** Content needs to be protected based on different levels of security. High-value content has a need for protection against unauthorized access. The same holds true for content, which needs to be kept secret for other reasons. Thereby, different security properties apply for read, write, and other form of access to content.

**Reliability.** Content reliability describes the service availability and content persistency. While for a personal homepage, 90% availability might be tolerated, other applications require an availability and persistency of as close as 99.999%. Therefore, content needs to be replicated and appropriate consistency mechanisms are required.

## 2.2. CMS Scenarios

During its life cycle content experiences several different phases, each of which addressing particular aspects. Figure 1 depicts those three main phases, which can be observed in many scenarios, *i.e.* *collaboration*, *management*, and *distribution*.

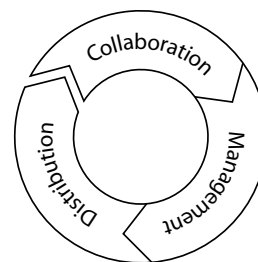


Figure 1. Phases in a content life cycle

A content life cycle can be represented as a closed loop since previously distributed content might influence or be used in new collaboration scenarios. A CMS has to deal with the diverse requirements derived from specific 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 specialized CMSs have to interoperate with each other and apply a standardized scheme for content exchange.

Three different content management scenarios are described below, which have been selected to represent each of the content life cycle phases presented. As shown in Table 1 these different scenarios are evaluated according to the content properties being outlined in Section 2.1. From this evaluation the specific requirements of each scenario are elaborated.

**Collaborative Content Development.** This scenario focusses on the content creation process where a CMS mainly supports archiving and versioning of content. Such collaborative work is bounded usually to a small group, thus, the popularity of such content is small. The topicality on the other hand is very high, since during a creation phase, content changes very frequently. Quality-of-Service (QoS) aspects are normally not an important issue here, 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 not an issue during creation, accounting is usually not necessary at this stage. Security

mechanisms are of importance, if access to the content needs to be restricted to a specific group of people.

**Table 1. Evaluation of properties for different CMS scenarios<sup>a</sup>**

	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.*

**Corporate Web Content Management.** This scenario deals with the maintenance of large corporate web portals where a CMS helps to administer the structure, design, and data of such web content separately. Furthermore, 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 varies a lot, so the content distribution infrastructure needs to be very flexible in order to be able to react on variations of the current demand. Topicality is usually not so important, however, quality aspects are crucial for corporate identity reasons, and accounting is 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 a desirable add-on.

**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 supports the maintenance and distribution of such content in many different ways. The popularity of multimedia content is usually very high. With respect to topicality, once the content has left its source, it is normally not changed any more. 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.

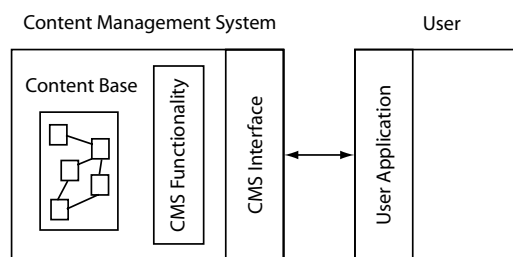
Each of these scenarios can be found in a number of existing systems. While WebDAV [24] and CVS [11] are examples for the collaborative content development scenario, Zope [26] and Cocoon [10] exemplify the corporate web content management scenario. Finally, multimedia content distribution technologies can be found, e.g., in Akamai's EdgeSuite [1] or NextPage's NXT3 [21].

### 3. Generic Architecture Design

Based on real-world CMS scenarios and their requirements as discussed above, a Content Management Architecture (CMA) has been developed. Flexibility of the new CMA is the key, mainly to be able to address any scenario's requirements. While the generic CMA being presented below is independent of any underlying network architecture (client/server, peer-to-peer, or hybrid), a dedicated P2P service architecture is used to design the distributed content management system, which is presented in Section 4.

#### 3.1. Content Management Architecture

A CMS can be described by a set of functionalities 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 hosts any type of content in a reliable and efficient way.



**Figure 2. Content management architecture**

The abstract model of the generic CMA (cf. Figure 2) assumes implicitly that components of the CMS may be distributed. A user can access the CMS by means of a user application, i.e. a file manager or web browser which interacts with the CMS interface and deals with the representation of the content.

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 content base is hidden from the user, which enables the user application to be implemented independently of any specific underlying storage platform. A list of key CMS interface methods and functionality components is given below and several options for content bases are outlined.

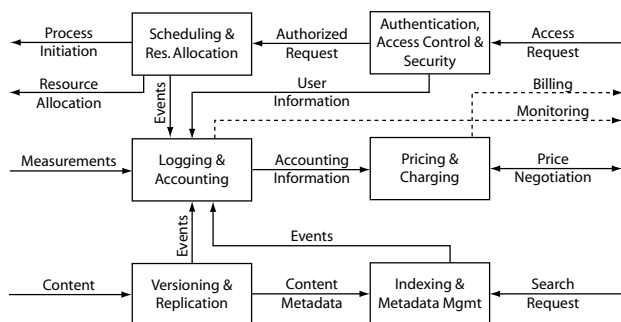
**CMS Interface Methods.** The following list of basic methods describing the access of content by a user are intended for a high-level interface description. The syntax of those methods follows the common form of *returnValue methodName(parameters)*.

- *id putContent(content, userId)* enables to upload content from a user's local content repository into the CMS. In case that the local repository is physically part of the CMS content base, as it appears in a distributed CMS, *putContent* denotes the method by which a user selects specific content

for sharing with other users. As a result a user receives an identifier, *e.g.*, a hash code or an URI, which uniquely identifies the content and is used as an access key. *E.g.*, [8] presents a content addressing scheme for HTTP which uses SHA-1 hashes as URNs.

- *content getContent(id, userId)* enables a user to retrieve content from a CMS' content base. Thereby content is either streamed or downloaded as one or several files. Content does not necessarily need to be stored permanently in a user's local repository. However, a distributed CMS benefits from local replicas, which can immediately be accessed by further users.
- *void removeContent(id, userId)* enables a user to remove content from the CMS content base, which determines the reversed method to *putContent*. Content is not necessarily deleted if available locally, but it will not be available any more through the CMS. The actual deletion may be performed otherwise, *e.g.*, locally through the operating system or through an additional CMS method.
- *void modifyContent(id, content, userId)* enables a user to overwrite previously uploaded content. Alternatively, the content parameter can be replaced by a patch file, containing only the differences between the old and the new content. Further improved CMS' could support content-specific editors.
- *id[] searchContent(metadata, searchType, userId)* enables a user to search for content matching with a specific metadata description. This method allows for different types of search, such as basic keyword-based lookup or sophisticated best-matching search. In addition, browsing in an index list or in metadata-based categories might be 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 list outlines those functionalities that are provided by a generic CMS. Specialized CMS', which focus on a specific CMS



**Figure 3. CMS functionality components**

scenario as exemplified in Section 2.2, may only support a subset of these components depending on given require-

ments. The CMS functionality components and their dependencies are illustrated in Figure 3.

- *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.
- *Pricing and Charging Support:* This 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 dynamic prices with the user. The settled tariff is used by the charging module 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 CMS method invocations and perform the allocation of resources. Different scheduling methods are in place, among them are algorithms that support real-time applications such as audio/video streaming.
- *Authentication, Access Control, and Security Support:* This functionality includes mechanisms to support various types of security requirements, such as privacy, access protection, integrity, authenticity, and non-repudiation. Different levels of security are provided, from public and anonymous to private and authenticated. In particular, access control requires the authentication of users, *e.g.*, realized by a shared secret, and it authorizes the access of content based on this information. Furthermore, suitable encryption and signing mechanisms are adopted to meet further security requirements.
- *Versioning and Replication Support:* Versioning keeps track of different versions of content, modified by one or several users. In addition it enables the provisioning of personalized content, *i.e.* content in different languages, different quality levels or different formats. In a distributed environment with replication, versioning ensures that modifications on content are performed consistently across all replicas.

• *Indexing and Meta Data Management Support:* Indexing manages a list of content, available through the CMS. This list can either be stored centrally or distributed. Furthermore, the attachment of meta-information to content is supported. Meta-data could either be extracted automatically from content or provided by the user through an additional interface. Options on attaching meta information to content are discussed just below.

**CMS Content Base.** The content base stores the content in a way which is transparent to the user. A CMS usually has to deal with many different types and formats of content [4]. The mapping of content on data determines a difficult task

which is out of the scope of this paper. Several storage options are discussed in [7]:

- *File-oriented*: Content is stored in files using a specific file system. Meta-information can be provided within the file, e.g., as meta-tags, or needs to be stored separately.
- *Object-oriented*: Content is encapsulated as an object, which is stored in an object-oriented database. Meta-information can be attached directly to objects.
- *Relational*: 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 the table.

The storage of content using file systems, databases or tables as discussed above can be done centrally or distributed. The presented options for the realization of the content base are therefore independent of any underlying network infrastructure.

### 3.2. Peer-to-peer Services Architecture

The aim of the CMA presented above was to abstract away from the underlying infrastructure or network topology. In the following, the underlying architecture for the proposed Distributed Content Management System (DCMS) is presented. The DCMS design is based on SOPPS, a service-oriented P2P architecture, that has been developed within the MMAPPs project [13], [18], [23]. The network model of SOPPS considers transparent end-to-end connections between different peers. In the following the peer model of SOPPS is briefly presented to illustrate key components of the adopted architecture. Another approach for a P2P service architecture is IRTL [15]. IRTL is very similar to SOPPS, however, it does not support versioning and replication.

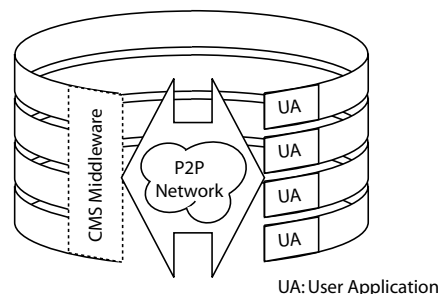
**Peer Model.** The peer model of SOPPS describes the internal structure of a peer, which is composed of four layers. Each peer provides several interfaces allowing peers to interact with each other. The following list briefly outlines major characteristics of these different layers.

- *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.
- *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 involve several resources and services to work together to provide a more sophisticated service. E.g., consider an image comparison service yielding a degree of similarity between images as its output. Such a service builds on several local resources, i.e. the images, storage and CPU 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 services. It is in charge of basic functionality, like access control, accounting, and service management as well as the protocols for distributed functionality. In particular, it includes functionalities that support the market management of the system, such as pricing and charging, as well as required versioning and replication functionality.

### 4. Fine Design of the DCMS

The design of the DCMS realizes the generic CMA introduced in Section 3.1 based on the peer model presented in Section 3.2. Peers participating in the P2P system can simultaneously offer and use services. The three dimensional illustration in Figure 4 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 the user application. Middleware and user application are connected through a link representing the underlying P2P network.



**Figure 4. Design of the DCMS**

Looking at a single peer from the front the design in Figure 4 exactly reproduces the two-dimensional CMA model shown in Figure 2 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 all peers. A specific content object is provided by a single peer or sever-

al peers together. The CMS middleware can make use of replicas of the same content and distribute parts of them in parallel.

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

- There exists an authentication infrastructure, such as X.509 [14], 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 (DHT), which is used to efficiently store and lookup content ids, user information and accounting data. Currently, many approaches based on DHTs are available for P2P systems. Among them are Chord [2] and CAN [22]. For the DCMS just any of these can be adopted.
- There are suitable mechanisms such as rating and reputation 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 and is described in the following as it would be deployed on all peers. In Figure 5 the 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*.

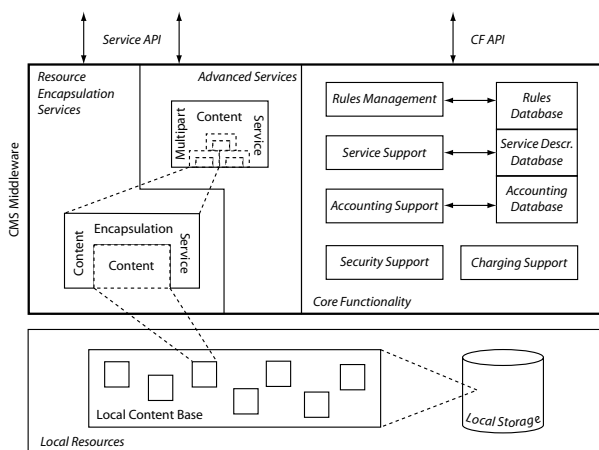


Figure 5. CMS middleware breakdown

**Content Encapsulation Services.** Any collection of data being identified as content, *e.g.*, a text document, an image, or a video, is wrapped by a content encapsulation service. The aim of this service is to represent content, which may reside on different types of data sources, such as discussed in Section 3.1. Thereby, data remains in the according data source and is referenced in the service by a pointer to the ac-

tual location. Only if the content is requested by another peer the data is de-referenced and appropriately formatted for distribution. To improve performance, appropriate caching mechanisms can be applied.

**Multipart Content Services.** Content which serves as a container for a collection of other content, such as web pages containing images or other types of multimedia content, are represented by a multipart content service. Multipart content services draw on content encapsulation services, but can also access local resources directly to increase performance and flexibility of the middleware using, *e.g.*, the functionality provided by a local file system.

Both types of services implement the service API, which enables a user application on another peer to interact with these services. All requests are caught 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 or advanced service, and it implements CMS interface methods presented in Section 3.1. To perform the necessary CMS functionalities these services make use of the 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 specific content is allowed or denied. [25] presents different security models for content-based access control. Services use access right information to decide, whether an access request, such as `getContent` or `modifyContent` (*cf.* Section 3.1), 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 can 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, and 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. For example, Mojo Nation [19] is a token-based approach that deals with decentralized accounting in P2P networks. Conditions of an accounting rule can basically use the same information as access rules. In addition, an accounting rule can indicate which values have to be meas-

ured and it contains expressions specifying how these measurements are aggregated. Depending on the evaluation result of a rule expression, a service creates 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 can be stored within a service and transmitted at every request together with the content, e.g., attached as metadata. 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. Another 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 level. Iptables [20] 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 can be adopted to perform access control and accounting on application-level.

**Accounting Records.** Accounting records are created by services and contain 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 stores the data in a local or distributed database. The charging support module makes 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 metadata. 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, the search descriptions can be stored in a DHT that is accessible by all peers.

## 5. Evaluation

The proposed DCMS design is analyzed and evaluated based on a typical CMS scenario covering the most relevant aspects of those three phases presented in Section 2.2. Figure 6 depicts four different peers A, B, C, and D, on which the CMS middleware (MW) presented in Section 4.1 is deployed. Different types of content bases and user applications (UA) may be used by the peers as long as they support the MW interfaces. In the following example a single content object is considered. The scenario reveals the following operations (cf. Figure 6):

1. *Collaboration and Management:* Peer A and B start with the collaboration on the content. Peer A locally creates a

first version of the content object. 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 the content, the access rule for both objects reveals that only peer A and B are allowed to get and modify the content. As peer B accesses the content through the MW interface, the content object is retrieved from peer A. If peer A or B modify the shared content, the versioning module synchronizes the modifications with the respective content object on the other peer.

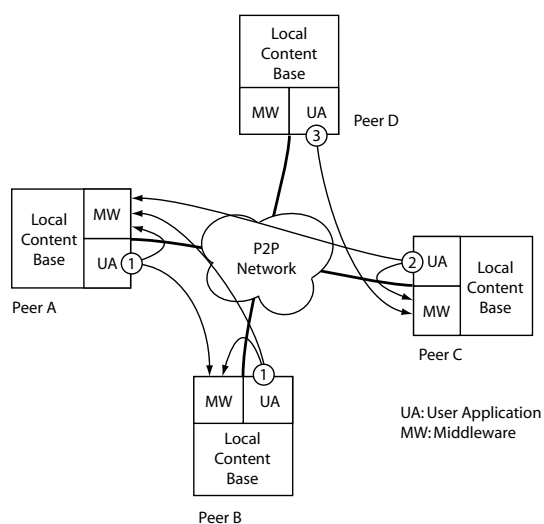


Figure 6. Example scenario

2. *Distribution:* As the content developed by peer A and B is settled, they designate peer C as an authorized distributor of their content. The access rule is therefore changed and peer C is assigned the right to get, but not modify the content. A new accounting rule determines the obligation to account for all *getContent* requests. An according service description is made accessible for lookup by any peer.

3. *Re-Distribution and Settlement:* Peer D discovers the service description for the provided content (not shown). Peer C sells the content to peer D on behalf of peer A and B. The access of the content by peer D depends on several conditions regarding payment information and previous accounting information. During the transaction the accounting rule triggers the generation of accounting records at peer C. On the basis of these records the charge can be calculated and peer D is billed. The revenues are shared between the involved peers. As mentioned earlier, it is assumed that reliable and secure mechanisms are in place, which are used to store and prove such sensitive data.

## 6. Summary and Conclusions

This paper presented the design of a Distributed Content Management System based on a generic Content Management Architecture (CMA). The proposed P2P-based CMS

middleware realizes the CMA in a completely distributed fashion, which forgoes any centralized components. The presented approach is very flexible as it supports all phases in the content 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 management functionalities, independent of any underlying infrastructure for storage or delivery. Considerations taken there, e.g., 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 realization 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. The DCMS design can be verified successfully based on a typical CMS scenario. A future prototype will implement the proposed DCMS design and show its operability in a real environment. Further work will detail accounting, charging, and billing issues on a content-based level.

## 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 design 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, Collab-Net, <http://www.cvshome.org/>, 2002.
- [12] GartnerConsulting: *The Emergence of Distributed Content Management and Peer-to-Peer Content Networks*, White Paper, January 2001.
- [13] 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.
- [14] R. Housley, W. Ford, W. Polk, D. Solo: *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*, RFC 2459, January 1999.
- [15] 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*; QoS/ICQT 2002, LNCS Vol. 2511, Zürich, Switzerland, 2002.
- [16] S. Jagannathan, K. C. Almeroth: *Pricing and Resource Provisioning for Delivering E-content On-Demand with Multiple Levels-of-Service*; QoS/ICQT 2002, LNCS Vol. 2511, Zürich, Switzerland, 2002.
- [17] 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.
- [18] MMAPPS Consortium: *Market Management of Peer to Peer Services (MMAPPS)*; EU IST Project, <http://www.mmapps.org/>, April 2002.
- [19] Project Mojo Nation: *Peer-driven Content Distribution Technology*; <http://www.mojonation.net/>, February 2000.
- [20] The Netfilter/Iptables Project: *Linux 2.4.x / 2.5.x Firewalling Subsystem*; Version 1.2.7, <http://www.netfilter.org/>, 2002.
- [21] NextPage: *NXT 3 Building a Content Network*; White Paper, NextPage, 2001.
- [22] 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.
- [23] B. Stiller, J. Gerke, D. Hausheer, J. Mischke: *Peer-to-peer Services Architecture*; Deliverable 4 to the MMAPPS Project, January 2003.
- [24] WebDAV: *Web-based Distributed Authoring and Versioning*; Extensions to the HTTP Protocol, <http://www.webdav.org/>, February 1999.
- [25] 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.
- [26] Zope: *Open Source Application Server*; Version 2.6, Zope Corporation, <http://www.zope.org/>, 2002.