

Specification

Cobricks-2

Community Bricks and Web Application Server

v0.15 - January 2004 (mk)



Abstract

This document describes the basic concept and the main ideas in the design of the Cobricks-2 platform toolkit. The specification includes a brief requirements specification and a system model. A detailed description of the functionality, data and user interface of the different basic components (Core, Portal, Context, User, Item, Category and Message) can be found in separate documents.

The document is aimed at anyone who plans to use or integrate the Cobricks-2 platform toolkit to build a community platform by customizing the components and user interfaces, and at anyone who plans to extend the Cobricks-2 platform toolkit by extending existing components or build new components.

Document History

V0.15	Jan 2004	mk	
V0.14	Dec 2003	mk	Split up sepcification document in separate documents for Core, Portal, User, Context, Item, Category and Message components
V0.13	Dec 2003	mk	Update of Category Component
V0.12	Nov 2003	mk	Update of Item Component
V0.11	Oct 2003	mk	Update of Calendar Component
V0.1	Aug 2003	mk	Initial version

Contents

1	Overview and Basic Requirements.....	4
1.1	Purpose.....	4
1.2	Requirements	7
1.3	Toolkit Overview	9
2	Basic Components	15
2.1	Core Component	15
2.2	User Component	15
2.3	Context Component	15
2.4	Item Component.....	15
2.5	Category Component	15
2.6	MessageComponent.....	15
3	Future Work / Extensions.....	16
3.1	System Architecture.....	16
3.2	Extensions.....	17
3.3	Portal Component	18
3.4	New Components.....	18
4	Miscellaneous.....	19
4.1	Open Issues / Unsorted Ideas	19
4.2	Related Products	19

1 Overview and Basic Requirements

1.1 Purpose

Cobricks-2 is a modular and extensible toolkit for building and operating *platforms for (virtual) communities*. A platform for virtual communities is a software installation configured and prepared to serve the needs of a particular community, i.e. an instantiation of a generic software product. The toolkit comes with a *Web based user interface* and a content management functionality and portal container for managing the user interface. In the basic setup the core platform is executed as a Web application in a Web application server. In addition to the Web user interface Cobricks-2 supports a *Web Services interface* to the functionality and data of the platform. Via this interface it is possible to access the platform from clients and (external) agents, to integrate the platform with other applications, and to connect different Cobricks-2 platforms.

1.1.1 Communities¹

In general a *community* is a group of people who share some interest, identify with a common idea or more generally belong to a common context. Thus, a community can be seen as a descriptive identity for a set of people.

Early sociological work points out, that communities always need a locality and interaction [Hillery 1955]². While the demand for a common physical locality is no longer seen necessary (due to electronic communication and virtual places), the demand for interaction or possibility to interact is still valid. In practical terms this possibility to interact implies the existence of a common communication medium, of common protocols and awareness of the existence and of the membership in the community.

The interaction in communities is said to serve a goal that is characteristic for communities: collaboration towards common goals or mutual support. A community is not just a set of people who have something in common and who have the possibility to communicate, but of people who are willing to help each other, who are collaborating to the advantage of all. This collaboration is often presented as a will to exchange knowledge – which leads us to communities of practice – but can also be found in collaboration to maintain sport facilities and set up sports events in sport clubs.

To summarize so far, a community is characterized by

- a boundary (common interest, common idea, common context),
- a sense of membership,
- ongoing interaction, and
- collaboration, mutual support.

Computer mediated communication allows communities to interact and collaborate without meeting face to face. *Virtual communities* are groups of people who would not otherwise form a community without the assistance of electronic media. The members of a virtual community only or at least mainly communicate through electronic communication channels. Computer

¹ For more information on the characterisation of community and on community support see for example <http://www.communixx.de/> or the additional material at <http://www.cobricks.de/>

² [Hillery 1955] G. A. Hillery: "Definitions of Community: Areas of Agreement", *Rural Sociology*, 20, pp. 111 – 123.

mediated communication is an enabler for the virtual community. In contrast to local communities, this situation offers new possibilities and dangers of anonymity.

To achieve the (spontaneous, unorganized) mutual collaboration, the main activities in communities are communication among members and finding people to communicate with. Communication includes exchange of information (indirect communication) and generating and sharing of knowledge. Hence, community support can be seen as “communication and matchmaking support”.

1.1.2 Community Support

Community support systems are software applications with functionality to support the information flow between the community members (providing a communication medium), and to support finding other members (matchmaking support).

Community, Community Support System, Community Platform

A *community* is a descriptive identity for a set of people and usually stands for this set of people. Hence, a community is a group of people with a commonality who interact to collaborate in the context of this commonality. An *online community* is a community whose members mainly interact through electronic communication.

A *community support system* is software with functionality to support communities.

A *community platform* is a particular instance of a community support system, installed and configured to support a particular community or set of communities. Community platform are usually realized as client server systems with Web browsers as client software. Client computers and client software are usually not considered part of the community platform. An exception from this is, if proprietary client software or hardware is used.

Hence, Cobricks-2 is a community support system that can be used to set up community platforms for supporting (online) communities. However, you will often find that the term (online) community is used synonymous with the term community platform.

Basic Support Concepts – Users and Context

While providing means for publishing and categorizing information is important for community support, this is not the most characteristic concept. The core of a community support medium is to draw attention to users, by capturing and visualizing electronic user representations, and by capturing and visualizing links between users and between users and content, messages or categorizing concepts.

A central issue in community support systems therefore is the user representation and the representation of relations between users and other data concepts on the platform.

In [Koch 2003]³ we argue that for modular and customizable community support only a very small number of different data concepts are needed on a platform:

- user representations (user profiles)
- items (resources) for dynamic content
- categories (communities) to cluster/categorize items and users

³ [Koch 2003] Michael Koch: Community-Unterstützungssysteme – Architektur und Interoperabilität. Habilitationsschrift, Fakultät für Informatik, Technische Universität München, Dec. 2003, see <http://www.cobricks.de/>

- messages (between users)
- relations among the different data concepts (especially from user to user or any other data concept) defining contexts

The Cobricks-2 platform follows this approach. See Section 1.3.1 for more information.

Tool vs. Medium

Current community support solutions mainly encapsulate data structures in (support) tools and provide the user with tool specific user interfaces. These tools and their user interfaces often predefine how the basic support medium can and should be used.

In contrast to the tool centric approach, Cobricks-2 follows a medium centric approach. The basic support medium (data concepts and channels/processes that distribute the data) is not hidden inside tools but provided in the open.

This approach has several advantages

- it allows us to focus on core concepts/media of community support
- it provides a platform that later can easily be (deeply) integrated with existing tools (like portals, content management systems, identity management solutions, or in the way of contextual collaboration with productivity applications)

Community Management

In addition to the medium centric functional focus on users and on relations, community support also has to provide means for community management.

Communities usually are rather unstructured and unorganized sets of people. Making a community / community platform work involves several prerequisites, the technology for the communication medium only being one. Explicit moderation and different means for supporting intrinsic and extrinsic motivation are needed in addition to the basic functionality.

Support for community management includes

- possibility to have moderators in the system with additional responsibilities and permissions
- possibility to monitor and visualize the activity in the community (for implicit and explicit management)
- tools for supporting intrinsic and extrinsic motivation (like bonus systems, top lists)

The Cobricks-2 system provides means for community management already built into the core of the system.

1.1.3 Cobricks-2 Use Cases

Use Cases

Following the motivation in the previous section, the main purpose of Cobricks-2 is to

- provide means for supporting direct and indirect communication in communities (of practice) - direct exchange of messages, indirect exchange of information and user representation in the system (for matchmaking)
- focus on user presentation and relations
- provide means for community management

The functionality is mainly directed towards authenticated users (community members). Other user classes are anonymous users (mainly for browsing in a sub-set of the information), community administrators and platform administrators. Finally, a platform build from Cobricks-2 should also be accessible by user agents or other Cobricks-2 platforms.

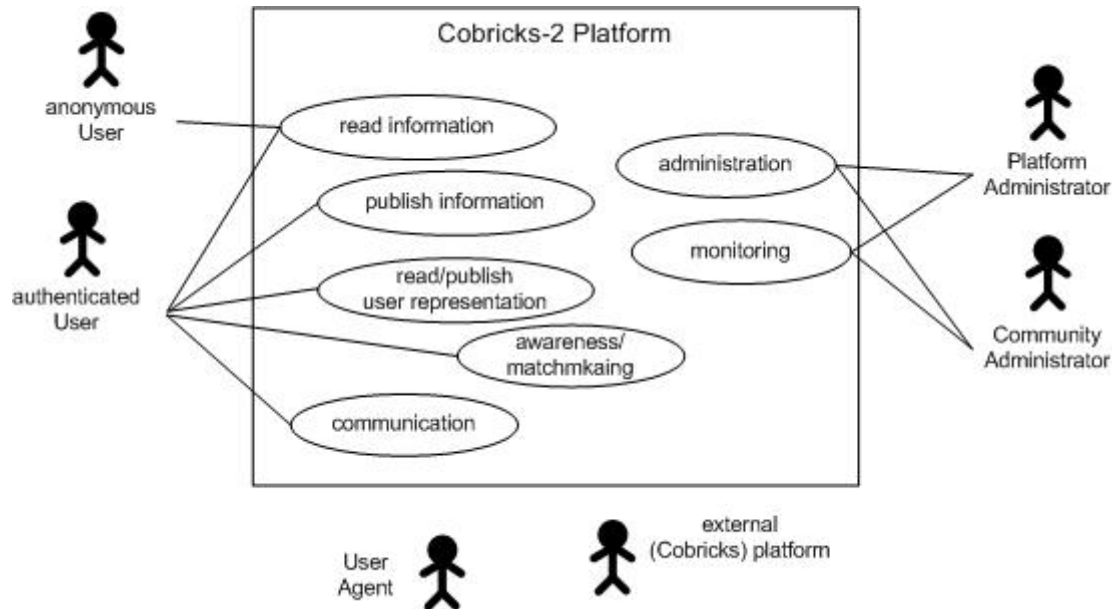


Figure 1. Cobricks-2 Use Cases

Target Scenarios

Cobricks-2 can be used to build:

- individual (interactive) web sites for companies or departments
- knowledge management platforms for communities of practice
- communication/interaction platforms for more general communities

The product can be used out of the box but is mainly targeted to be customized for a special usage scenario.

1.2 Requirements

1.2.1 Functional Requirements

One source for the functional requirements is the discussion of community support functionality in Section 1.1. This leads to requirements for particular functionality:

- *User Representation and Context / Relationship support*: The platform provides basic support for user representation and for handling and presenting relationship of users with other users and content
- *Content Management*: The platform provides means for user-centric content management – i.e. for publishing content that will stay connected to the publishing user.
- *(User) Messaging*: The platform provides means for direct communication with known and unknown community members.

- *Community Management*: The platform provides means for supporting community management

All functionalities should be highly customizable (to be able to build platforms for different types of communities), and be accessible through Web user interfaces and Web Services interfaces:

- *Basic customizable components*: Cobricks-2 comes with six highly customizable basic components for user management (user), context or relationship management (context), item management (item), category management (category), user messaging (message) and web-page management (portal).
- *Web user interface*: The main user interface of Cobricks-2 is the Web (summarizing all HTTP-based content formats like HTML, WAP or XML). The web user interface should be highly adaptable. In addition to the adaption of the web user interface with means of the platform it should also be possible to use the platform functionality from other portals, i.e. use a different portal software for creating and managing the (web) user interface.
- *Web-services user interface*: In addition to the Web user interface the components of Cobricks-2 provide their services to external applications via a Web-services interface. Via Web-service interface different platforms can be connected (see below) and the platform functionality can be integrated in web service enabled applications (contextual collaboration).
- *Multi-language*: The user interfaces (and the resource data schemas) should support multi language interfaces.

An additional and even more important source for requirements is the intended usage of the system. Cobricks-2 should not only be used to build stand-alone community support platforms. While it should be possible to build a full platform of Cobricks-2 core elements only, the main usage will be to integrate Cobricks-2 with other systems in an enterprise IT. Examples are the integration in portal services, the integration with user profile servers or the integration with content management system.

Functional requirements resulting from the need for integration are:

- *Modular, customizable and extensible*: According to the requirements of the community it is possible to customize the existing components and to extend the platform by modified or new components. This means that Cobricks-2 consists of a basic framework and of components providing the functionality that can be provided/used on a particular community platform.
- Focus on *integration* – user management, web portal – but also provide own portal framework (to be workable out of the box). Cobricks-2 provides basic functionality for collaboration (especially in loosely coupled communities), is usable out of the box, but easily integrates with other tools.
- *Connecting platforms*: Different Cobricks-2 platforms can be connected for exchanging community content and for reusing user profile information. Especially the integration of external user directories (identity management solutions) should be very strong and support different directories (LDAP, Liberty Alliance, IDRepository).

1.2.2 Interface & Software Requirements

In addition to the functional requirements listed above we have defined some additional (non functional) requirements:

- The platform should be usable out-of-the-box, i.e. after installing and initializing the software (and the databases for persistent storage) the user interface of a default platform should be available.
- The implementation should be done for/with JDK 1.4
- The implementation should be based on standard frameworks and standard libraries from Java-based Open Source projects, i.e. Apache Jakarta Tomcat, Apache Jakarta Velocity, Apache Jakarta log4j
- The platform should be structured in a way or provide tools that allow easy migration from Corbricks-1- or Cassiopeia-CAS-based platforms
- The platform should use relational databases for persistent data storage and support different database servers (minimum: MySQL, PostgreSQL, Oracle)

More details on these interface and software requirements, together with some design constraints and documentation requirements can be found in the documentation of the Development Environment of Cobricks-2 (separate document).

1.3 Toolkit Overview

1.3.1 Data Concepts

Following the discussion in the section on community support, the Cobricks-2 platform is based on simple but extensible and customizable data concepts providing a community support medium:

- user
- context (relation)
- item
- category
- message

Detailed information on the data concepts can be found in the chapters on the components covering each concept. We will just give a brief overview here.

In general the data concepts are modelled as frames with slots (objects with (object) attributes). Sometimes we also describe the data objects as set of attribute value pairs. The models (classes) for the data objects are described in ontologies, i.e. the attribute names and the possible values for the attributes. Objects are identified via (concept specific) integer identifiers or via global URIs.

User

The concept of a “user” describes digital user representations. This is a set of attribute value pairs identified by a (local) identifier, a user login. As with all other data concepts there can be different classes of users which are defined in an ontology. The ontology does not only define the user classes and the relationships among user classes, but also defines which attributes can be set for a user, and what (data) type the values of the attributes have. (As with all other data concepts it is possible to use attributes with names not defined in the model in user objects of any class – however, there is no special support for these additional attributes and they default to string values.)

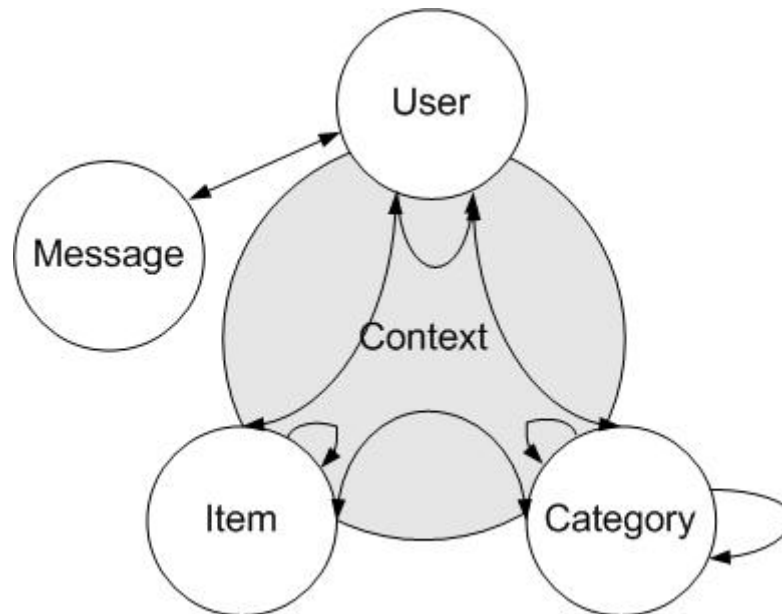


Figure 2. Core Data Concepts

Attributes of user objects can define relationships with other users or other data concepts of the platform. An example for this is a buddy list attribute that stores a set of references to other user objects. Relations like this are (at least virtually) modelled as context objects (see next section), but can still be accessed from the user object directly. See the chapter on the Context Component for more information about how this is handled in the platform.

Context (Relation)

The core concepts of community support are users and contexts. Thereby, contexts are relations of users (or other data concepts) to 1) other users, 2) content (items or messages), 3) categorizing concepts (categories), or 4) external resources (identified by URIs). Since context is of major importance for community support, it is modelled as a separate data class. As with users, contexts are sets of attribute value pairs that are grouped into context classes. The context classes describe the actual type of relationship and define which attributes (and attribute value types) the context will have.

It is possible to completely separate the management of context from the management of users and other data concepts. However, it is more intuitive to manage some contexts with the concepts they are mainly bound to. Therefore, the context management will not explicitly store all contexts but derive some dynamically from other contents with the help of other components. That is why we speak of virtual context information sometimes. In addition to this it is also possible to derive context information in a really dynamic manner based on rules. See description of the Context Component for more information on this issue.

Item

Items are content objects that are published by users. As users, item entities are basically built from a set of attribute value pairs identified by a local or a global identifier. There are several (hierarchically structured) item classes available, that define what attributes an item should have and what type the attribute values have. An item instance is associated to one of these item classes or item types.

The core attribute of an item is the “content”. This attribute gets special handling and can be of four different types: text, xml, binary, URI.

Items can have attachments and simple annotations/ratings. Additionally, it is possible to relate items to each other. Item to user relations are usually done using item attributes with value type user – or in the other direction by using user attributes with value type item.

Examples:

- Item classes: announcement, project, publication, web-page
- Item instances: a particular announcement, project, publication, web-page

Category

Categories are focal points to cluster entities of users or items. Hence, the main task of a category is to hold a set of items (item references since items may belong to a set of categories at once) or to be associated to a set of users (users are said to select categories as interests or to be member of a category – there are different classes of membership).

Since it is a very generic clustering object, there might be several classes of categories. For the beginning categories again are defined as attribute value pairs. In the most simple version there is just a title and an entifier attribute. For more complex versions additional attributes (defining how a category reacts) can be defined.

Examples:

- Category classes: InterestCategories (no members possible), SharedBuddyLists (interestCategories with members and admin roles), PrivateFolders (no members), WebFolder
- Category objects: “course.program.informatics”, “teaching-staff”, “kochm-publications”, “/course/info/”

Messages

Messages are the second type of content objects on the platform. Messages are published (sent) by users with the intention to deliver them to a specified set of other users. This set of recipients can be defined explicitly (by a set of recipient addresses) or implicitly (by a set of attributes the recipients must match).

While the main task of items is to be persistent information, the main task of messages is to be delivered over predefined channels.

1.3.2 Architectural Concepts

To achieve modularity and extensibility the Cobricks-2 toolkit is structured into components. First there is a *core component* covering component management and component configuration and communication/messaging.

Then there are components for the basic data concepts discussed in the previous section:

- *user component*
- *context component*
- *item component*
- *category component*
- *message component*

The basic distribution finally includes a *portal component* that provides functionality for making the functionality available via a Web based user interface in a portal context.

The components can be replaced by alternative implementations, and new components (building on old ones or not) can be added to the platform. For component management the platform currently offers a light-weight component management functionality. A more heavy weight component approach building on Enterprise Java Beans (EJB) is planned for the future.

Components provide interfaces to the outside (to other components etc.) through two types of classes. The first type of class, the so called *manager class*, belongs to the application layer. The second type of classes belongs to the presentation layer (presenter classes, servlet classes, templates). Finally, components usually store information in the persistent database and therefore include database schemata for the data layer. Figure 3 shows an overview of the basic platform concept and the different modules and sub-modules in Cobricks-2.

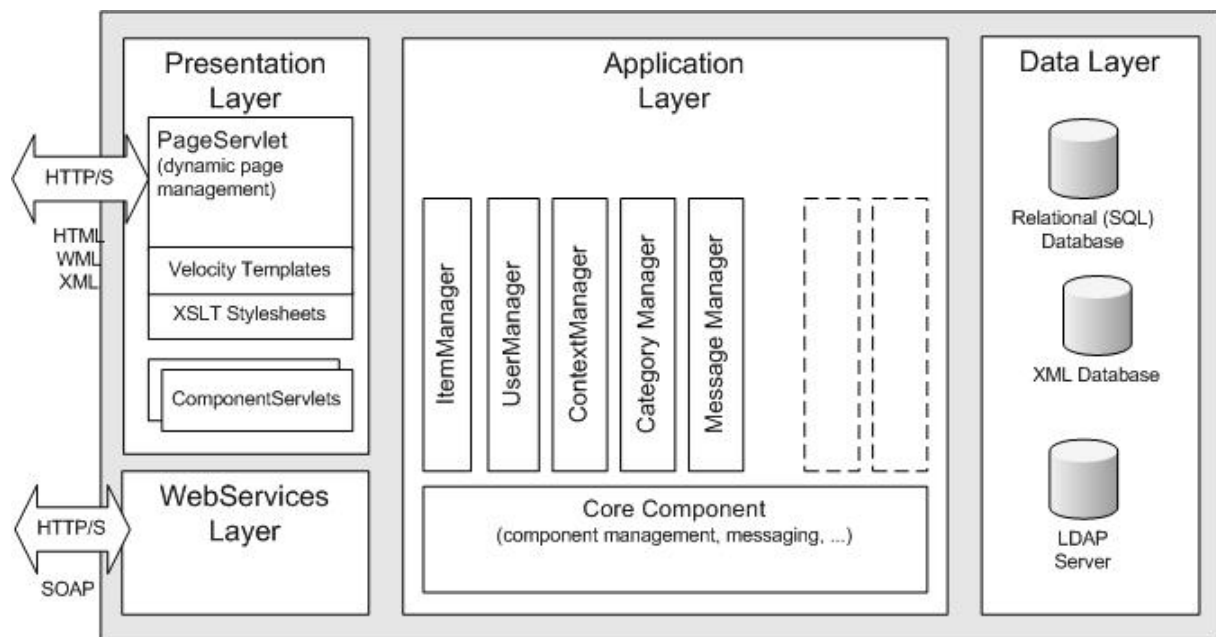


Figure 3. Architecture overview

The standard setup is to instantiate all the objects of the platform in one Java virtual machine, i.e. in one Tomcat application server instance. However, there are several possibilities for structuring the objects into more than one virtual machine. The main idea is to separate the portal functionality and the application components. Figure 4 shows a setup where the objects are distributed into two environments (virtual machines). In both environments a full Cobricks platform is instantiated, i.e. a Tomcat application server with core component and other components. However, in the first environment the item, user, category and message managers act as stubs only. They provide the interfaces and forward requests by RPC (RMI, SOAP, Corba) to the second environment. In the second environment the real manager objects handle the requests and return the results by RPC to the first environment. The portal component is installed in the first environment only. The core manager is instantiated in both environments.

The setup in Figure 4 already presents the idea of integrating Cobricks-2 with existing portal products. In this case the first machine is an existing portal implementation. This portal implementation is provided with stubs of the manager classes that can be used in Java Server Pages (JSPs) or similar template mechanisms. See Figure 5 for this setup. In this scenario it is

of special importance to couple the user management of the 3rd-party portal and the user management of the Cobricks platform. Different possibilities for doing this are described in the user component chapter.

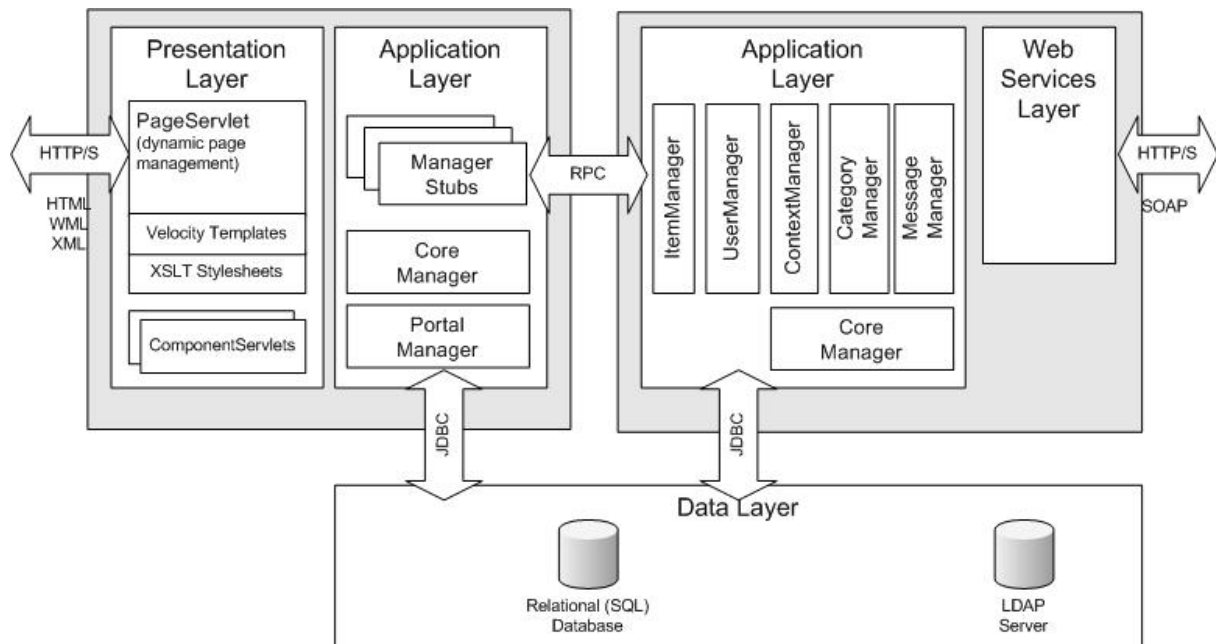


Figure 4: Cobricks-2 on two machines

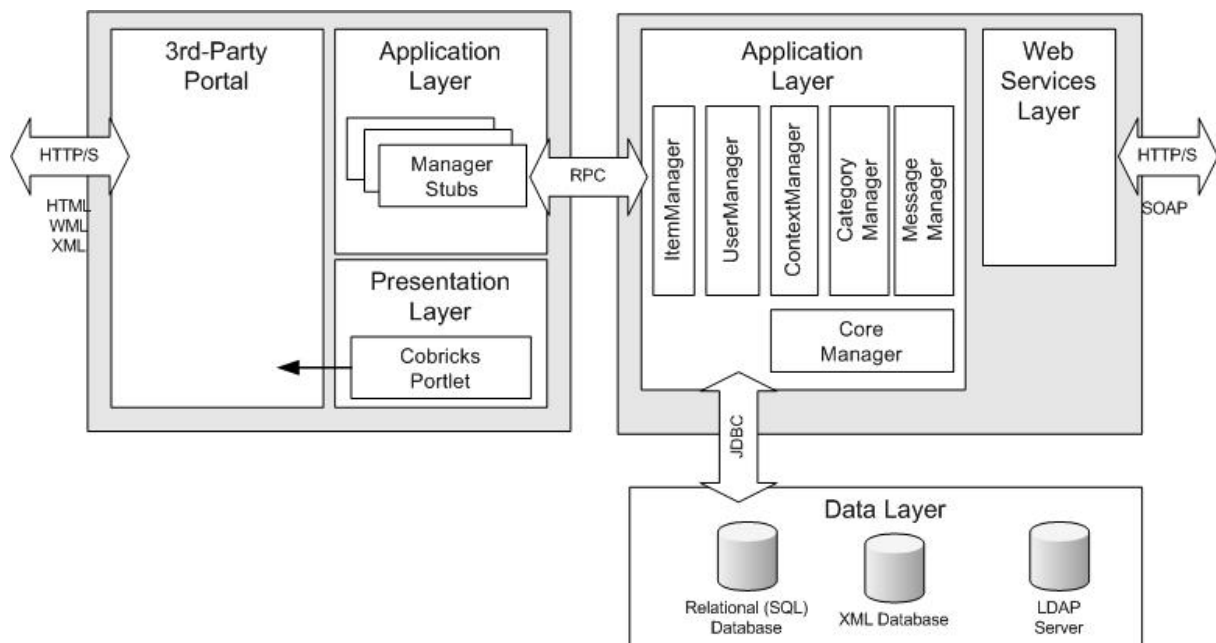


Figure 5: Cobricks-2 and third-party portals

The modular design of Cobrick-2 allows for even further integrative scenarios. Examples are the integration with 3rd party portals (for presenting the user interfaces), with 3rd party document management system (for storing content – item management), and user and access rights management (authentication and authorization) in external directories. In this case

Cobricks-2 only provides community enhancements to existing components. Examples for the community enhancements are user representation, awareness, annotations/ratings, and categories/communities.

The modularity and the Web-services user interface make Cobricks-2 ideally equipped for “*contextual collaboration*” – i.e. to provide a background layer that offer collaboration services that can be integrated in different user applications.

Figure 6 shows a possible overview of applications at the workplace of an office or knowledge worker. Collaboration and community functionality is only part of the picture, and should be integrated by the service aggregation components like portals and teamrooms, and be usable from other applications like productivity applications and office applications.

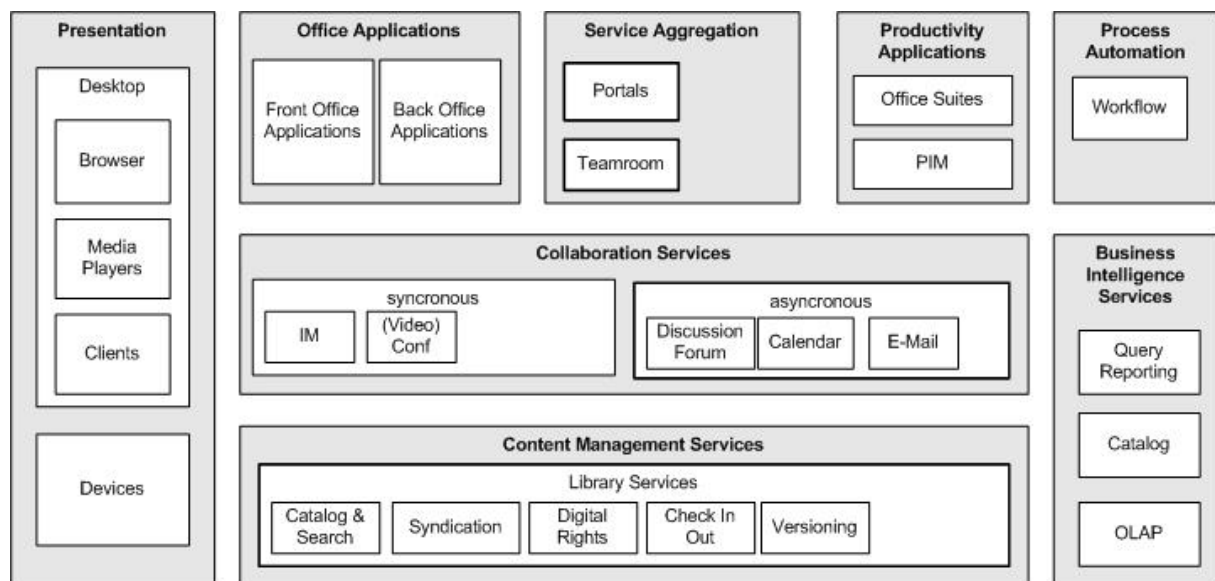


Figure 6: Applications at the workplace of an office worker⁴

For the basic components we follow a consistent structure: After a short overview of the document we describe the functionality. In this section we outline the basic data concepts of the component and the functionality provided by the manager class. In the section on data we concentrate on the persistent storage of data in the database, and in the section on user interface we concentrate on the web user interface provided by the component.

⁴ Figure adapted from a slide describing the “Knowledge Worker Infrastructure (KWI)” from the META Group METAView on Content and Collaboration: 2003/04 Issues and Trends

2 Basic Components

The Cobricks-2 toolkit comes with some basic components representing the basic data concepts discussed before. All components come with a Web user interface and a WebServices Interface.

These components are described in separate documents. In the following we will just list the components and provide an overview of their functionality.

2.1 Core Component

tbd

2.2 User Component

tbd

2.3 Context Component

tbd

2.4 Item Component

tbd

2.5 Category Component

tbd

2.6 MessageComponent

tbd

3 Future Work / Extensions

This specification represents Version 1 of the Cobricks-2 toolkit. This specification currently focuses on clearly separating the basic concepts and on defining a modular toolkit that supports setting up complete community platforms. In addition to focussing on the medium and integration aspects we therefore also cover the “tool aspect” of community support.

Some design decisions have been made to make implementation of the first version possible. In this context we also have decided not to implement some functionality or standards that would make sense. This is especially true for the “medium aspect” and the integration aspects mentioned in earlier sections.

This chapter will give an overview of what we are planning for future versions of Cobricks-2. The purpose of including the ideas in this document is to create an awareness for the future goals – and thereby enable current developers to prepare their components for future developments.

3.1 System Architecture

3.1.1 Portal Integration

As already discussed in Chapter 1 and 2 we foresee the usage of Cobricks-2 in 3rd-party portals. That is, the portal component is omitted and all the presentation of cobricks functionality is done in the external portal software.

Most of the commercial portals provide some means of “portlets” to integrate external functionality. This often is based on Java Server Pages (JSP) technology.

To provide open portal integration we therefore first have to provide portlet modules for the different commercial portals. The goal should be to

- allow usage of full Cobricks templates in the external portal – to be able to reuse the user interface developed for the components
- allow usage of Cobricks functionality (from managers objects and from presenter objects) in portlets that include other functions

In addition to the integration into portlets, the portal integration needs support for single-sign-on, a common user database for portal and Cobricks platform. See Section 3.1.3 for more information on this issue.

Portal integration should be available for a variety of portal products:

- BEA Web Logic Portal
- IBM WebSphere Portal
- Sun iPlanet Portal
- Oracle 9i Portal
- SAP Portal
- Apache Jakarta JetSpeed

To achieve this broad support the emerging Portlet standard JCP 168 should be used.

The whole portal component could also be based on the Apache Jakarta JetSpeed portal – at least it would be a good first step to do a JetSpeed integration.

3.1.2 Resource Integration

The item components provides a generic resource management framework. We already have mentioned that it should be possible that the contents of the resources are stored elsewhere, e.g. in document management systems.

Integrating document management systems should be a task to address.

In this context again the integration of the user databases is an important issue.

TBD

3.1.3 Identity Management Integration (Single-Sign-On)

The third large area of integration is integrating existing user management systems.

Here we will provide support for

- LDAP
- Windows2000/2003 user domains
- Liberty Alliance, MS Passport
- Our own IDRepository

3.1.4 Java-2 Enterprise Edition (J2EE)

The current implementation of Cobricks-2 provides a very lightweight component and (event) messaging framework. To make distribution and deployment of components in enterprise environments easier it would make sense to make the components available as Enterprise Java Beans (EJBs) and use the full functionality of the J2EE framework for Cobricks-2, including the Java Messaging Extension (JME).

One possibility to easily migrate in this direction would be to use the OpenEJB container inside Tomcat. That would allow us to model components – especially manager classes as EJBs and keep all the Tomcat/Web-Application-Server specific stuff.

3.2 Extensions

3.2.1 Community Management

Currently community management is not really integrated ... so we surely have to do some work here ...

Incentive / Point System

One essential part of community management can be an efficient reward system – i.e. maintaining points that are earned for activities on the platform.

We need something like this, which is very easy to configure.

Examples for a Reward system (from TeleMat)

- for registration: +10
- for login: +3
- for published item: +200
- for message to member: +3
- for password change: +5

- for entry in profile: +3
- for upload of photo: +15
- for item recommendation: +10
- for item comment: +50
- for read item: -1
- for read document: -10

Community Logs

Another part of community management will be a very configurable log system – enabling community managers to set sensors for different activities – and do public or private reports on the logged data.

3.2.2 Ontology definition

Support RDF and/or OWL for defining the component ontologies

3.3 Portal Component

Top 10 lists

- list of the pages that are visited most
- list of the pages that have been updated most recently

3.4 New Components

Some ideas for additional components – also see similar sections in the component documents.

3.4.1 Course (information, registration, timetable)

3.4.2 IDRepository

3.4.3 Chat

4 Miscellaneous

4.1 *Open Issues / Unsorted Ideas*

Authenticated access to manager functions from templates

When templates are processed it is currently possible to call any manager or presenter function. Even when these functions ask for authentication – i.e. for a user login as parameter, any value can be specified. This is a problem when users are empowered to edit the web pages themselves.

Solution idea:

- do not add reference to manager and presenter objects in velocity context, but references to user specific wrapper objects (can be generated once per session)
- these wrapper objects store authentication and forward it to the manager and presenter methods
- possibility to automatically generate the wrapper classes – for all public functions that have a first parameter userid/userlogin or so

4.2 *Related Products*

4.2.1 ZOPE

ZOPE: “Zope is a framework for building web applications” – from the Zope Book, Chapter 1: Introducing Zope

- We want to go further: Web application but also access through agents
- Zope: Python – hard to extend and to integrate – we want to tackle these issues!
- Zope: No special functionality for community management

4.2.2 PHP, PHP Nuke

TBD