FINALIST

never stop developing

Design CMS Container, Iteration 1

FINALIST
never stop developing

{V} Version history

Date	Version	Comment	Author
26-1-2006	1.0	Initial	Nico Klasens
6-2-2006	1.1	review	Peter Reitsma

Title	CMS container
Status	Draft
Date	26-1-2006
Authors	Nico Klasens
Document owner	Nico Klasens
Distribution list	
Approval	

{C} Contact list

Name	Jean-Luc van Hulst
Function	CEO
Address	
Telephone	010-217.08.02
Fax	010-280.96.20
E-mail	JL@finalist.com

Name	Robbrecht van Amerongen
Function	Projectmanager
Address	
Telephone	
Fax	
E-mail	

Name	Hessel Rosbergen
Function	Sales
Address	
Telephone	
Fax	
E-mail	

Copyright 2005, Finalist IT Group

Niets uit deze uitgave mag worden verveelvoudigd en/of openbaar gemaakt door middel van druk, fotokopie, microfilm of op welke andere wijze dan ook zonder voorafgaande schriftelijke toestemming van Finalist IT Group.

FINALIST
never stop developing

{C} Contents

{1} Preface

In the last few years MMBase implementers have build several content management systems for customers. Many projects tried to create a generic CMS which could be used for new customers. At the moment, everyone still builds CMS-ses with help from earlier projects. All systems are hard to generalize, because they are too specific for the customer domain. This document describes a generic layer as the base for a custom implementation. Most concepts of this generic layer are already implemented in projects and could easily be assimilated into this layer.

MMBase has been the base for most CMS-ses. MMBase is a very dynamic, flexible and versatile content engine. The generic layer should keep this behaviour and add functionality to become the container for a CMS. See the appendix for more information about MMBase.

1.1 Who should read this document

The intended audience for this document includes the following groups:

1. Developers who implement and extend the CMS container functionality
2. Developers of CMS container components.
3. Developers who implement new sites based on the CMS container and the components.

1.2 Glossary of terms

MMBase	a general purpose object oriented content management system.
Content Repository	A data repository with added content services . The services a content repository implements are searching, fine-grained access control, content categorization and content event monitoring
Contentchannel	A node in the tree of the content repository. A contentchannel categorizes content by type of information
Cms application	A CMS driven website developed for one customer
Data Repository	A database acting as an information storage facility
Site management	Management of web pages and navigation in a web application
Contentelement	Base object type which is stored in the content repository
Content instance	An instance of a contentelement
Object type	Definition of attributes (fields) and operations (functions) for an object
Maven	A Tool to build projects and generate documentation
Portal	A portal is a web based application that –commonly- provides personalization, single-sign-on, content aggregation from different sources and hosts the presentation layer of Information Systems. Aggregation is the action of integrating content from different sources within a web page. A portal may have sophisticated personalization features to provide customized content to users. Portal pages may have different set of portlets creating content for different users

Portlet	A portlet is a web component, managed by a portlet container, that processes requests and generates dynamic content. Portlets are used by portals as pluggable user interface components that provide a presentation layer to Information Systems.
Primary content	Different kind of content elements like Article, News item, etc consisting of text field and a set of secondary content.
Secondary content	Content that is related to primary content: URL's images and attachments.
Content block	A content element including it's secondary content.

FINALIST
never stop developing

{2} Abstract

2.1 Why a CMS container?

The first question that comes to mind is what a CMS container is going to offer compared to the current project process. The current lifecycle of a project is something like the following. The Customer requests a project proposal. Implementer writes a project proposal based on previous projects and existing frameworks. After the proposal is signed, the development process starts. A more detailed functional design is made and the sources of previous projects and frameworks are integrated into a custom application. When the application is in its final stage, it is tested and accepted by the customer. Issues found by the customer are not applied to previous projects when it is in a "borrowed" source.

The CMS container project lifecycle could be like the following:

- A customer request a project proposal and mentions that it wants to use the CMS container.
- Implementer has a few questions for the customer. How should the main layout of the front-end site look like? What type of information (article, faq, etc.) should be displayed on the site? How should this information be displayed (list, item, graph, etc.)? What extra functionality (newsletter, poll, etc.) should the site have?
- Implementer implements the answers of these questions in the development process Extra functionality on top of the standard CMS container functions are implemented as components that are reusable by default.
- In the final stage the customer tests and accepts the custom application. Issues found in the sources of the CMS container will be fixed and applied to the next release of all custom applications.

The advantages of the CMS container

- Customer already knows the basic features of his application when requesting a project proposal. The basic features are present in the CMS container.
- The CMS container defines a standard which requires more component-like sources. Development time for common features will go down over time.
- Features made for other customers can 'easily' be re-used.
- Customers in the same domain can share features.
- The customer can preview the project as soon as the development process starts.
- Multiple sites for the same customer have the same editor user interface.
- Issues in CMS container features will be fixed for all customers. Multiple customers will find issues which result in a robust and stable container.
- Customers only have to thoroughly test the custom made features..
- Upgrades to a newer version of MMBase or other frameworks require less development time for all customers.
- With every new customer the CMS container can grow and improve its features.
- Development time per customer will go down and developers can switch easier between CMS container based projects.
- Set of automated tests for the CMS container which assures quality over time.

2.2 Goals

The following prerequisites are used for the design of the CMS container

The CMS container is focused on the technical side of the CMS projects. Most of the functionality is already implemented once and could be assimilated in the CMS container. The CMS container has to become a solid base for CMS projects from a development point of view.

The code should re-use other products and should use other open source components as much as possible. The CMS container is going to depend on MMBase as content engine and on Pluto as templating mechanism. The build process should allow easy upgrade of both products and should enable the possibility to fix issues in these dependent products.

Every build iteration is going to add some functionality which is not always required for a customer project. The CMSCMS container should be highly configurable and that means that the core of the CMSCMS container has to deal with this. The CMS container will provide hooks where possible for future add-ons or customizations. Basic functionality for these hooks will be implemented

The CMSCMS container will be a barebone application with a lot of optional and configurable components. How easy a release of a customer CMS is made will highly depend on the way the CMS container build environment will deal with all these choices and dependencies.

To conclude, the CMS container has to deliver:

- A lean and extensible core which defines rules which every CMSCMS application should adhere to.
- A way to define a component which can be packaged, deployed and distributed
- A build system which manages dependencies and versions.

As a side note, the CMSCMS container is based on other products and API's and has to accept the design decision and shortcomings of these products and API's.

FINALIST
never stop developing

{3} Introduction

This document describes the functional and technical decisions which are made in the design of the CMS container and how it integrates with 3rd party API's. The outline of this document is as follows:

Overview of a CMS application
Architectural overview
Content repository
Site management
Build system
Component definition

The CMS container is built on top of MMBase MMBase. This document assumes that the reader understands the concepts and base functionalities of MMBase. See the appendix for more information on MMBase.

FINALIST
never stop developing

{4} Overview

Every web based CMS consists of at least two separated applications. These are an editorial site (including a staging environment) and one or more front-end sites (live environment). The editorial site is used for the maintenance and configuration of the other sites. Robust user interaction is the key requirement of this site. The editorial site maintains the content, site structure and templating of the front-end sites.
Some of the functionalities which the CMS container should provide are:

- Content repository
- Screen management and templating
- Navigation
- Workflow

4.1 Content repository

The only thing which really matters in a CMS is storing, retrieving and modifying of content instances are stored in a repository. The content repository consist of a hierarchical structure of contentchannels where content instances are stored in.

4.2 Screen management and templating

Most of the CMS projects have their own way of modelling templates in the front-end. This is one of the reasons that none of the projects is easy to generalize.
JSR-168, the portlet container specification, generalizes a templating model. A portlet container maintains portlets and provides them with the required runtime environment. A portlet container contains portlets and manages their lifecycle. A portlet container receives requests from the portal to execute requests on the portlets hosted by it. A portlet container is not responsible for aggregating the results produced by the portlets. A portal implementation, which uses a portlet container, can organize the portlets on a screen and will aggregate the results of the portlets based on the layout of a screen.
As a side note, the portlet spec is more focused on enclosing multiple customer applications on one screen than on presenting more or less read-only sides with little user interactivity . Custom projects usually have a small amount of functional application-like components.

4.3 Navigation

The navigation, just like templating, is implemented for every project in a different way. Most sites have a tree-like navigation structure. Every menu on the front-end site displays a part of the tree. Some common used menus are tabs, left/right tree menus, footers or crumb paths.

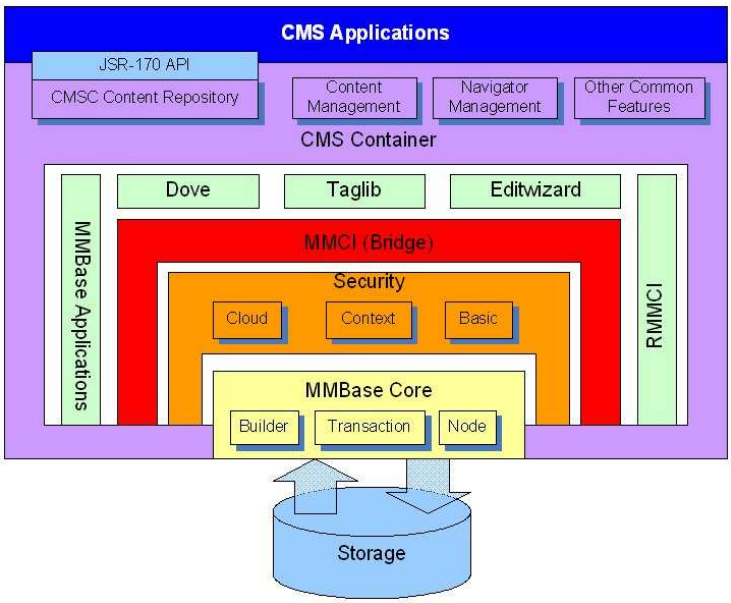
4.4 Workflow

The reason to introduce a workflow is to assign responsibilities on different stages in the publication process to the users of the CMS. In the current projects with workflow, the security is based on roles on contentchannels. The role on the nearest contentchannel, where the content instance is created in, is used for the rights of the user. Every content instance should go through a couple of stages before it is visible on the front-end(s).

The workflow on the content repository could be the same as described above. Workflow on the site management part of the CMS container will be dependend on how site management is implemented.

{5} Architectural overview

Readers familiar with the MMBase documentation will recognize some parts from the below figure. The figure shows clearly how the CMS container and custom CMS application are related to each other..



As shown, the CMS container is a normal MMBase application which collaborates with several MMBase extensions and implements some features. The difference with a normal MMBase application is that it is not an end product. It still requires a CMS application to put everything together before an editor can use it.
MMBase can host many types of content applications. The CMS container is focused on repository-based content applications. In other words, the CMS container is a Kernel Architecture for Repository-based MMBase Applications.

The most important part of a repository-based application is that it has a content repository. The primary functional objective of the CMS container is to define a standard content repository which CMS applications should adhere to. A standard content repository consists of an object model and application code supporting it. Optional components of the CMS container can implement features based on the rules defined for this content repository. The content repository should be complete enough to support many content standards like JSR--170 (Java Content Repository), web metadata and possibly the Dublin core.

A content repository is nice, but not enough for a web-based CMS application. A front-end site which shows the content is also required. Many MMBase applications have their own site management. The CMS container is not different. The preferred choice of the CMS container is JSR-168 (Portlet Container). Many MMBase applications integrate their content repository with site management. The

CMS container makes a clear distinction between the two and keeps the dependency as loose as possible. The advantage is that other CMS applications can use the repository and choose another site management implementation if they want. In the future, the CMS container could use an different more advanced portal implementation if the current developed version does not meet requirements. The drawback is that there are 2 parts in the CMS container which canb look the same, but are very different.

The CMS container will host, next to the core parts (repository and site management), several optional components which are frequently used in custom CMS projects. Examples of these components are: newsletter, agenda, faq, poll, banners, chat, forum, image gallery, etc,
The components hosted by the CMS container will be maintained in the same way as the core parts and will be supported in every release of the CMS container.

FINALIST
never stop developing

{6} Content Repository

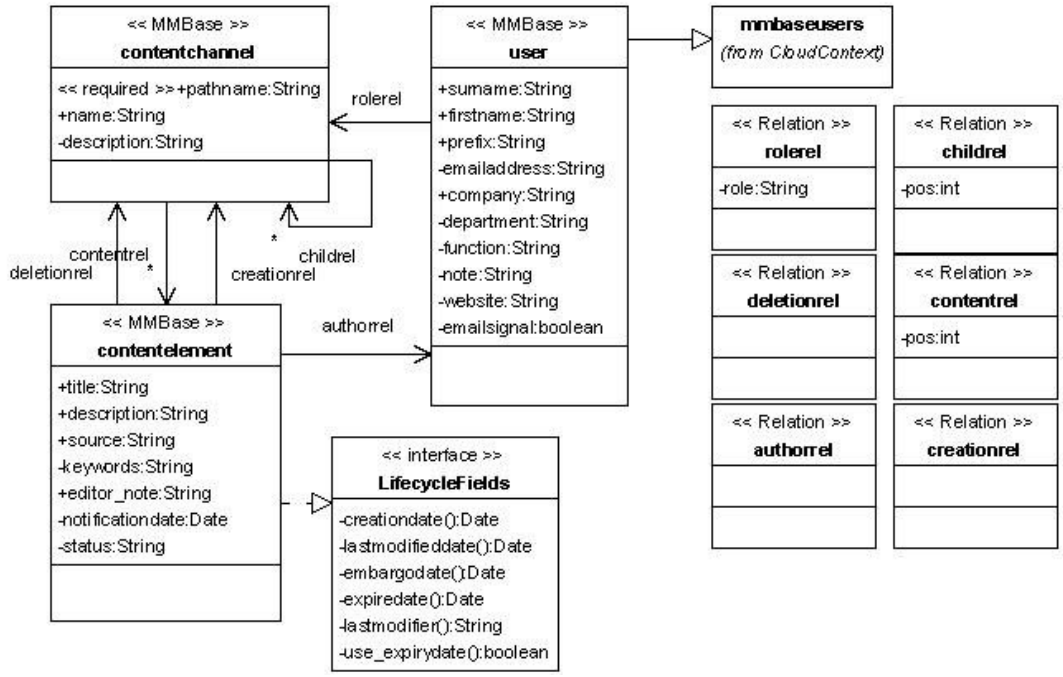
The content repository is the heart of the CMS container. A content repository adds content services to a data repository. The services a content repository implements are searching, fine grained access control, content categorization and content event monitoring. The CMS container uses the MMBase core as the content repository engine. MMBase provides a way to define a repository object model and has basic features for content services such as storage, query, security and notification

The repository consists of two object types. The first type is contentelement which is a base object type for all content instances and has all attributes which are shared by all content instances. The second type is contentchannel which is a hierarchical categorization tree. Contentelements are linked to this tree.

This repository model makes a major design decision compared to the MMBase cloud concept. MMBase advocates a network model of related objects. Every object can be an entry point to other objects This repository model narrows it to a tree of related objects. There is one root contentchannel as entry point to the objects.

There are several reasons to choose for this limitation on the MMBase cloud model.

- Many existing MMBase applications use a tree structure to access content. The CMS projects which inspired the CMS container all have a tree model.
- Users are familiar with tree structures. The file system of operating systems are also represented as a tree structure.
- The java content repository standard (jsr-170) categorizes content in a tree structure. (See appendix)



The above figure shows the full object model which is used to implement the services of the repository.

As mentioned before, the repository is a tree structure of contentchannels. Every contentchannel has a (parent/child) relation to one parent contentchannel. A parent contentchannel can have multiple child contentchannels. A contentchannel has a name, description and pathname. These fields are persisted in MMBase. A contentchannel also has some virtual fields which are derived from the persisted pathname field. The virtual fields are path and level. A contentchannel is in a tree structure and can be accessed through a path relative from the root contentchannel. Every pathname of a parent contentchannel is a fragment of the full path. The value of the level field represents how deep the contentchannel is in the tree, The root contentchannel is on level one.

The object type contentelement has all attributes which can be used to store metadata of a content instance. A content instance has an embargo- and expiry date which can be used by a front-end site to determine if a content instance should be visible. The other lifecycle fields are maintained by MMBase and will be updated when the content instance is persisted.

There are three relations between a contentchannel and contentelement. All are used to implement a different rule in the repository.
The creationrel relation defines in which contentchannel the content instance is created in. The contentchannel relates with the user node with a creationrel to indicate the owner of the content instance.with the creationrel is the owner of the content instance. A content instance can only have one creationrel relation.
A content instance is linked to a contentchannel through a contentrel relation. Content instances are ordered in the contentchannel based on the pos field of the contentrel relation. A content instance can

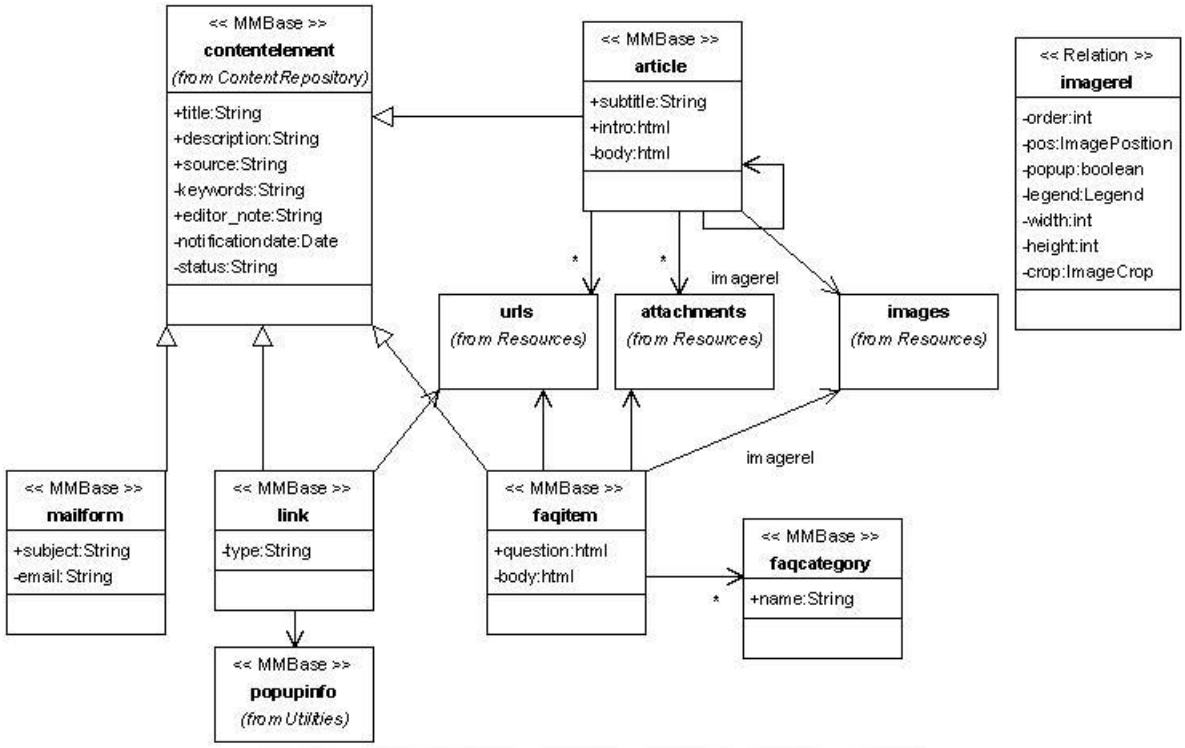
be linked to multiple contentchannels. Since content instance can be shared over multiple contentchannels, the contentchannels can be used as a meta datastore as well. In this case, the tree should not be regarded as a central structure but as a metadata repository. As an example a product could be shared between a business and resellers branch of the repository. In this use case, content elements will generally be related to many contentchannels.
Next to the root contentchannel is a special contentchannel called trash which has all content instances linked to it which are not linked (through a contentrel) to any other contentchannel in the tree. The creationrel relation of these content instances is related to the trash contentchannel. The deletionrel is used to remember the last creationrel in the repository tree.

Every content instance has an author. This is represented in the object model through the authorrel relation to a user. A user can log into the editor application to modify the content repository.

The authorization functionality on the content repository is quite simple and straightforward. It basically consists of a relation with a special role between users and the repository tree. The relation points to a contentchannel in the tree and privileges cascade downwards. Privileges lower in the tree override those that were set higher in the hierarchy.

Obviously, a group structure is missing in the authorization at the moment .Privileges are now directly assigned to users. It would be better to allow assignment of roles to groups of users.

6.1 Custom content model



A CMS application can define its own content model. The only requirement is that some of the object types extend from the contentelement object type to be able to store them in the repository.
The repository divides the objects, which are stored, in two groups based on the fact that they extend contentelement or not. Objects which extend contentelement are primary content for the repository. Objects which do not extend contentelement are secondary content. To be more exact, a content block is stored in the repository. A content block consists of 1 primary content object and related secondary content objects. In the figure above an article with images, urls and attachments is a content block.

The content repository does not limit the number of object types extending from contentelement. It is up to the CMS application how many object types are created. A CMS application has to decide how generic or specific the cloud will be. A very specific cloud will have several article object types like pressrelease, newsitem, infoitem, wheaterreport, etc. The generic cloud will only have an article object type. The number of object types in a very specific cloud can explode when many domains are covered. Consider the following when defining the cloud

- How many domains will be covered and how often is a new domain added. In a specific cloud , every domain will add new object types.
- A generic cloud will have less edit screens and templates in the front end, but they will be more complex.
- A generic cloud does not describe a domain exactly. The information lost by not defining object types has to be stored in metadata of the generic object type.

FINALIST
never stop developing

{7} Site management

Site management in the CMS container has a very flexible and complex model At a first glance, it is a very overdesigned model to display content on a web page. Many existing CMS projects don't have such a model and they work just fine. The current model implements a portal environment. To understand the reasons for this model, it is important to understand the way portals work and in particular the jsr-168 compliant ones.

What is a portal? The jsr-168 specs defines it as follows:

A portal is a web based application that –commonly- provides personalization, single sign on, content aggregation from different sources and hosts the presentation layer of Information Systems. Aggregation is the action of integrating content from different sources within a web page. A portal may have sophisticated personalization features to provide customized content to users. Portal pages may have different set of portlets creating content for different users.

Not all features of a portal are of interest for the CMS container. The one which is of interest is that a portal hosts the presentation layer of information systems. Jsr-168 does not define what an information system is. A portlet on a portal page does know how the information system should be presented..

A portlet is a web component, managed by a portlet container, that processes requests and generates dynamic content. Portlets are used by portals as pluggable user interface components that provide a presentation layer to Information Systems.

A portlet container runs portlets and provides them with the required runtime environment. A portlet container contains portlets and manages their lifecycle. It also provides persistent storage for portlet preferences. A portlet container receives requests from the portal to execute requests on the portlets hosted by it. A portlet container is not responsible for aggregating the content produced by the portlets. It is the responsibility of the portal to handle the aggregation.

The following is a typical sequence of events, initiated when users access their portal page:

- A client (e.g., a web browser) makes an HTTP request to the portal
- The request is received by the portal
- The portal determines if the request contains an action targeted to any of the portlets associated with the portal page
- If there is an action targeted to a portlet, the portal requests the portlet container to invoke the portlet to process the action
- A portal invokes portlets, through the portlet container, to obtain content fragments that can be included in the resulting portal page
- The portal aggregates the output of the portlets in the portal page and sends the portal page back to the client

For detailed information on portlets read the jsr-168 specification.
http://jcp.org/aboutJava/communityprocess/final/jsr168/

The CMS container does not define what can be shown on a web page. This is up to the CMS application to define. The CMS container should only provide a presentation framework and this is what a portal environment does.
The CMS container has to define how a CMS application can define how information is shown on a web page. This is exactly what the portlet specification (jsr-168) does. A portlet is a limited piece of end-user functionality, consisting of view and logic, that can be independently installed in a portal. This is in line with the goal of the CMS container to define a component.

By having the CMS container adhere to the JSR-168 standard opens several options for reusability.

1. Portlets developed for other jsr-168 portals should be reusable inside the CMS container
2. Portlets developed for the CMS container should be reusable in other portal products.
3. The portal implementation of the CMS container will be very limited. Migrating to a feature-rich jsr-168 portal product should be relatively easy.

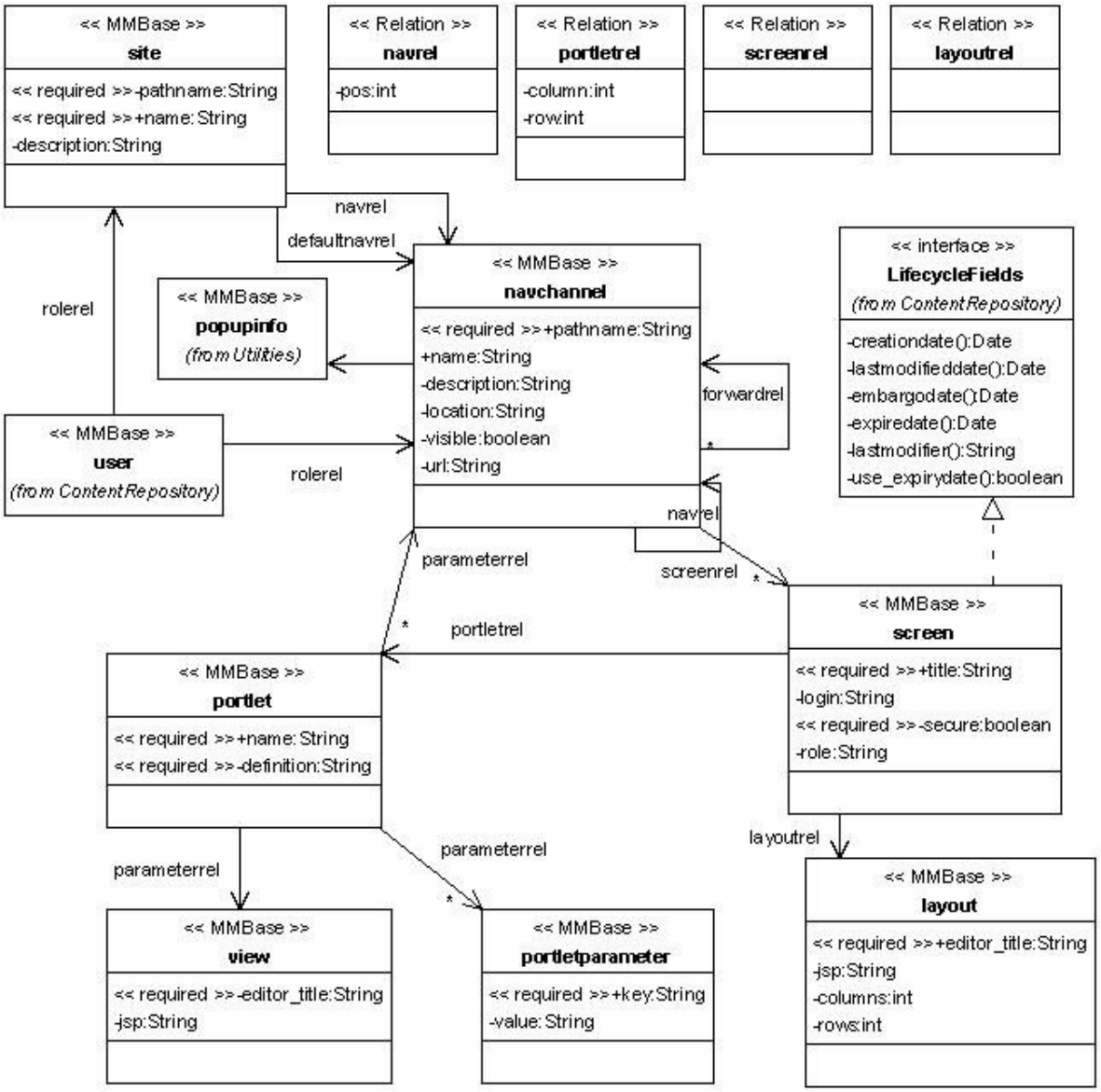
Another feature not yet mentioned is that portlets can be represented in different modes. A portlet is default in view mode, but can be switched to edit or help mode. A portal can define custom modes next to the standard three modes. This opens up the possibility to switch portlets in an advanced edit mode to show a rich text editor in a web page.

JSR-168 is a first version of a specification and is lacking some things. Portal implementers will implement their own version for these things which will make it hard to migrate from one product to the other. JSR-286 is the second version and will address some of the current gaps and will align with WSRP 2.0 (Web Services Remote Portlet)
The largest gap in the current specification is inter-portlet communication. It is not possible to tell in a standard way to other portlets that something has changed which might be important for them. An example of this is an shopping cart overview portlet which has to react on the action that a product select portlet adds a product to the shopping cart.

It should be noted that most CMS driven sites are rather limited in user interactivity and the portal approach will in some cases be overdesign since most of the portlets will simply be read-only. Furthermore, every portal needs to redefine the composition of templates and portlets.

7.1 Cms container portal

The portal implementation of the CMS container is built on top of Pluto. Pluto is the reference implementation of the portlet container specification. The example portal implementation which is provided by Pluto is changed to work with objects in MMBase. The example portal implements the minimal requirements for a portal to test the reference implementation. This is enough at the moment to provide a presentation framework. The CMS container portal does not have the intention to become a feature rich portal implementation like Liferay, Jetspeed or JBoss Portal.



The above object model is currently used by the CMS container portal. It is enough to model all the required features of the templating mechanism, but it is still far from an editor friendly presentation layer.

The site objects are the entry points for the portal implementation to access portal page information. A site object is the root of a navigation tree. All navchannels below it represent the sitemap. The navrel relation is the parent/child relation of the tree. The navchannels are ordered in the sitemap through the pos field in the navrel relation. The defaultnavrel relation between a site and navchannel is used to mark one of the navchannels as the home channel for the site.
Friendly urls are created based on the pathname field of the site and navchannel objects. A pathname is a fragment of the full path. A virtual path field exists in the site and navchannel objects.

A navchannel is a menu item in the site and can be visible in the menu. A menu item (navchannel) can point to a screen in the site or an external url. The screen or url is shown in a popup when a popupinfo object is related to this navchannel.

Screen objects are portal pages for the CMS container portal. Screens aggregate the portlets on a screen. A screen has an embargo- and expiry date which can be used by a front-end site to determine if a screen should be visible. The other lifecycle fields are maintained by MMBase and will be updated when the screen is persisted.
A layout object defines where the portlets on a screen are displayed. The current layout assumes a grid of columns and rows, but this does not have to be the case.

Portlet objects represent portlet instances on a portal page. A portlet requires a unique name on the page. Every portlet instance has a portlet definition which is defined in the portlet.xml (jsr-168). The portlet.xml defines which java class file is used and which modes are allowed.
Portlet instances can use preferences (key-value pars) to persist their state. These preferences are stored in portletparameters in the model. The CMS container portal has some special preferences. These preferences are links to other objects in MMBase and are modelled through a parameterrel relation to these object types (navchannel, contentchannel and contentelement).
A view object is related to a portlet instance to tell how the content should be displayed in the portal page. A portlet instance will process user actions and will dispatch the request to the view to generate the mark-up fragment (html, xml, wml, etc).

The authorization functionality on site management is the same as for the content repository. It is quite simple and straightforward. It basically consists of a relation with a special role between users and the navigation tree. The relation points to a site ore navchannel in the tree and privileges cascade downwards. Privileges lower in the tree override those that were set higher in the hierarchy.

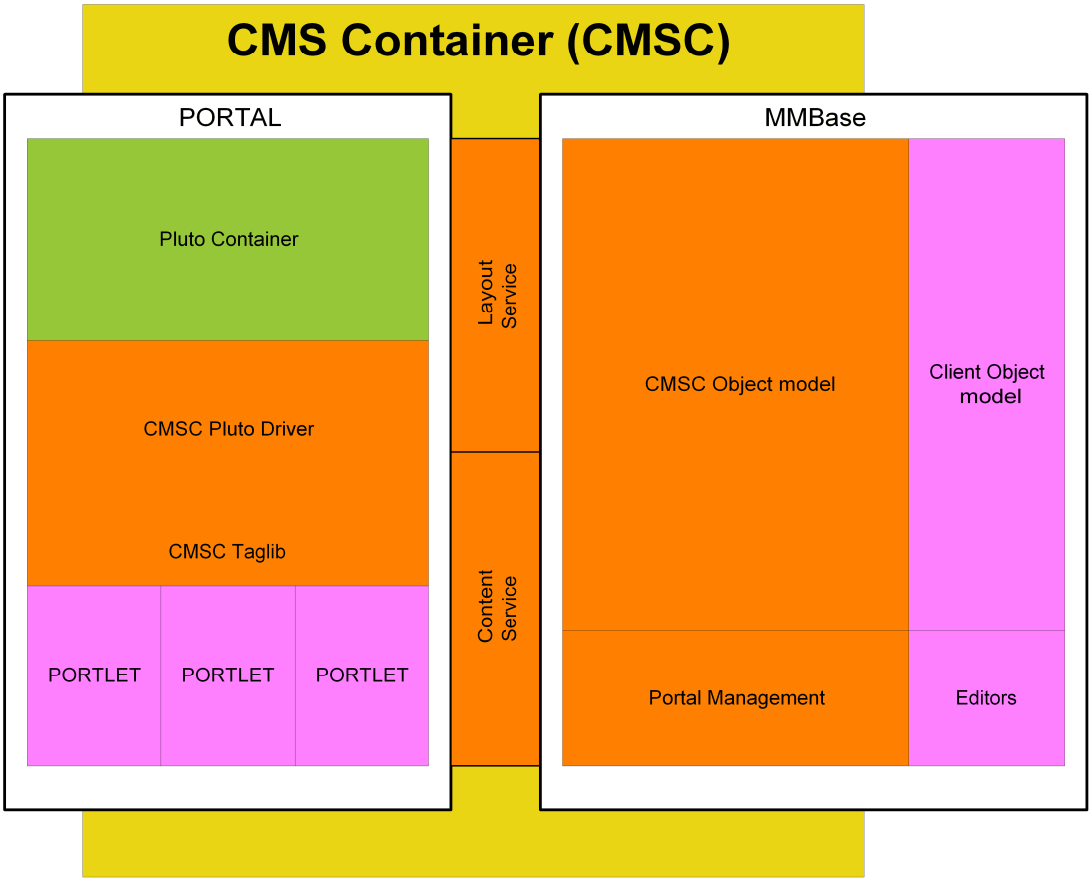
As mentioned earlier the model is not finished and lacks some features in regard of user friendly edit screens. It currently lacks the following features:

- no possibility to specify a default layout for a part of the site hierarchy.
- no possibility to specify which layouts are allowed for a part of the site hierarchy
- no possibility to specify a default layout for a portlet
- no possibility to specify which portlet layouts (views) are allowed for a subtree of the site hierarchy
- no possibility to define which views can be used by which portlets.
- no possibility to define which portlets, or which portlet views, can be used to show which content elements

{8} Architecture revisited

The previous chapters describe in detail which functionality the CMS container has. A more detailed figure of the architecture of the CMS container can be drawn. The below figure shows the core components of the CMS container and where a CMS application can add custom functionality.

The CMS container consists of two parts. The portal which host the presentation layer and MMBase which host the object models. The CMS application can add its own object model and editors. Custom portlets can use the custom object model to display content.



FINALIST
never stop developing

{9} Cms container reusable component

By having the CMS container adhere to the JSR-168 standard, the reusable components become plain JSR-168 portlets consisting of view and functionality, object model and back-end editors.

This approach also defines the scope of the reusable component, being a limited piece of end-user functionality that can be independently installed orthogonally on the host system. Examples are a poll, a search function, a forum. In fact all editable content is presented through portlets, mostly portlets that are native to the CMS container.

In the context of the CMS container dependencies (MMBase and JSR-168) and the rules defined earlier a component could consist of the following resources.

- an MMBase application for the object model. An MMBase application includes an application.xml, associated builders and default data.
- portlet definitions
- views (JSP's)
- layouts (JSP's)
- editwizards
- editors
- configuration resources
- java sources
- taglib
- documentation
- examples

Component source layout currently in use

- config (MMBase configuration)
    - applications (MMBase application)
- resources (configuration)
- src
    - tld (taglib)
    - webapp (templates)
        - editors (editors)
            - config (editwizards)
        - WEB-INF
            - portlet.xml (portlet definitions)
            - templates
                - view (views)
                - layout (layouts)
    - java (java sources)
    - test (test sources)
- xdocs (documentation)

It should be clear that workflow, versioning or authorisation by this definition can not be defined as CMS container reusable components. This means that these features will always be present. Implementers should however be able to configure these functions in such way that they seem to be absent. E.g. by defining only one step in the workflow, the workflow seems to be absent. Or by defining only one security group that bundles all roles, it seems authorisation is completely absent.

The reusable component feature will heavily rely on the applications framework within MMBase. This framework currently lacks the possibility to uninstall MMBase 'applications'. This means that as long as this feature is not implemented in MMBase, the CMS container can not provide proper 'uninstall' functionality.

FINALIST
never stop developing

{10} Build system

The CMS container uses Maven as the build tool. Maven has defined some standards which the CMS container can benefit from:

- Standard way of working with dependencies and versions
- Standard project directory layout
- Standard way of packaging and distribution of artefacts.

Every project in maven is described by a Project Object Model. Every project delivers an artefact which can be installed in a maven repository. A maven repository is a special structured directory or web server. A project can define dependencies to other artefacts. Maven searches for dependencies in remote repositories and downloads them to its local repository.

Maven supports some artefact types by default. Some of these are: jar, war, ear, rar, zip. A maven plugin can define its own artefact type by following some naming conventions. The maven build of MMBase has defined an mmbase-module artefact which can easily be used in the CMS container build.

An mmbase-module artefact has four special directories.

- config - mmbase config files
- templates - jsp, images, etc.
- examples - examples for usage
- WEB-INF - jar, taglibs, etc

Another useful feature of the mmbase-module plugin is that it can extract the artefact into a war file structure.

Maven has support for a multiproject build. This means that maven can create multiple artefacts and install them into the local repository. The multiproject build will resolve the order in which projects should be build based on the dependencies defined inside each project. Dependencies which are not built by one of the projects will be downloaded from remote repositories.

The core of the CMS container is a multiproject build itself and delivers several artefacts at the moment

- Contentrepository – implementation of the content repository and editors
- Sitemanagement – implementation of site model and editors
- Portal – JSR-168 portal implementation
- edit-webapp – edit environment which host the editors of the repository and site
- utilities – Utilities used in other core CMS container artefacts
- basicmodel – custom example object model and editwizards
- documentation –uml diagrams

MMBase has a multiproject build and delivers several artefacts. The CMS container uses the following artefacts as dependencies.

- mmbase

- mmbase-cloudcontext
- mmbase-taglib
- mmbase-dove
- mmbase-editwizard

A CMS application project will define dependencies to MMBase and the CMS container artefacts and maybe some other optional components (eg lucene from mmapps.sourceforge.net)
Building the maven (multi)projects in the right order will create a deployment for a specific CMS application.
A full clean deployment for a CMS application (CMScImpl) will have the following build sequence

- MMBase – checkout of 'all' module from cvs.mmbase.org
- Cmsc – core artefacts from the CMS container
- Optional components – eg mmapps lucenemodule
- Cms application – CMScImpl demo application

A developer can decide not to do a clean build and only run the parts he/she changed. Maven will then use the artefacts (in binary form) from a remote repository. Somebody else has deployed the artefacts to the remote repository. In an ideal situation the developer only has to build the CMS application (last step).

To conclude, Maven provides many of the features the CMS container requires for the build system.

FINALIST
never stop developing

{11} Appendix A: MMBase Explained

11.1 Overview

- What is MMBase?
- History
- Architectural overview
- Data and Object models
- Core features
- Extensions

11.2 What is MMBase?

MMBase is a general purpose object oriented content management system.

Content Management System: Software that enables one to add and/or manipulate content
Object Oriented: A popular buzzword that can mean different things depending on how it is being used. The term object-oriented is generally used to describe a system that deals primarily with different types of objects, and where the actions you can take depend on what type of object you are manipulating.

MMBase is a so called web content management system (CMS). This means that MMBase can be used to manage information with only one necessary tool: a web browser. MMBase makes it possible to manage information independent from time and place. Information can be created, stored, changed, manipulated and deleted with MMBase, without knowledge of technology. This of course only if you have the right authorisation to do so.

MMBase is an object-oriented system. This means that information pages normally consists of several objects, for example: picture, text, location, audio, etcetera. Each of these objects can be re-used many times, on many different channels. Nevertheless, an object is stored only once in the database. So if the object changes, the changes will immediately be visible on all devices. Most CMS are not object oriented, but document oriented. These systems are fine if you want to store documents 'as they are', but they are poor in avoiding redundancy of information.

MMGlossary: Multi Media Base (MMBase)

- dummies: a tool by which you can build intelligent web sites.
- managers: open source content management system, with strong multi-media features.
- information analysts: content management system by which text, images and other content can be managed in an object oriented way.
- techies: java based web-application, which acts as an object oriented interface to relational databases.

11.3 History

- 1995 - internal development at the Dutch Public Broadcasting organization VPRO. This is the year Java was released.
- May 2000 - first Open Source version 1.4
- End of 2000 - a handful of MMBase websites mainly hosted by broadcasters
- March 2005 - several hundreds of MMBase websites (Even Chinese).

The origin of MMBase was developed in the heart of the Dutch Public Broadcasting organization VPRO (www.vpro.nl). After 5 years of internal development, the VPRO decided to make MMBase Open Source in May of the year 2000. Soon other broadcasters also started to use MMBase as well and a year later there where already a handful MMBase websites.

After the first year the take-up got into another gear and at this moment there are -as far as we know- several hundreds MMBase websites. And the number of users that consider MMBase to be strategically important for their organization is growing fast. From an easy to use CMS just for the presentation of some web pages, the system is now turning in to an enterprise system.

MMBase is build in Java, a high-level object-oriented programming language developed by Sun Microsystems. Java is an object-oriented language similar to C++, but simplified to eliminate language features that cause common programming errors. Java is a general purpose programming language with a number of features that make the language well suited for use on the World Wide Web.

In the beginning, MMBase was a full web server with his own templating language (SCAN) and its own servlet implementation. The current code base still contains some hacks from back then. Many components have been replaced by standard J2EE solutions, but it doesn't like to give up its own features.

11.4 Content cloud

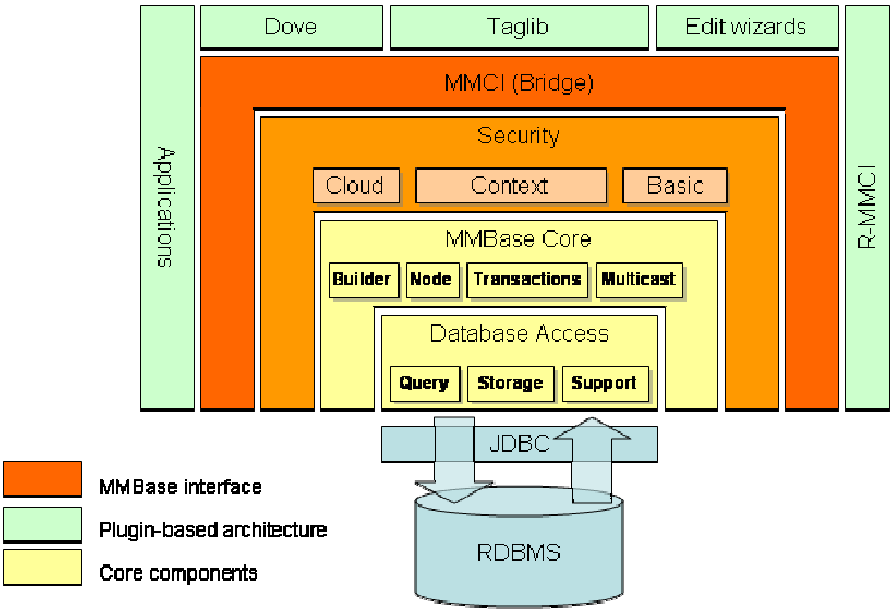
- information is stored in objects
- objects are related to other objects
- objects belong to an object type
- allowed relations between object types
- object types and allowed relations are just pieces of information and can be stored.

MMBase models its information as objects. MMBase creates objects that are a representation of object types and MMBase maintains objects that specify the possible relations between these object types. This structure results in a web of object pieces and their relations, also known in MMBase as "the content cloud".

The content cloud offers optimal flexibility on use and reuse of content, while in the meantime it doesn't impose any constraints on the layout of the site that refers to the content in the cloud. For

example, you could easily create a large single content cloud and publish this cloud, or parts of it, to several different websites, each site presenting their own unique layout.

11.5 Architectural overview



The MMBase architecture can be subdivided into the following layers.

- Database Access MMBase Core
- Security
- Bridge
- Plugins

11.6 Database access

Database access layer provides the mapping from relational to object oriented when necessary

MMBase only requires a configured database connection

- Table creation
- Constraint generation
- Insert, Select, Update and Delete

Database statements for system administrators
Create, Drop, Backup and Restore database

MMBase is connected to a database via the JDBC interface. Ones you've created a database and connected it to MMBase, MMBase will do the rest for you. It will take care of the creation of tables and will generate and take care of link tables. So the only SQL statements which you have to learn to being able to work with MMBase are: creating a database, making a backup of a database, restoring a database and dropping a database. (This is only really true for well supported databases)

The database access layer generates the SQL queries. MMBase itself uses a query framework which is not tightly coupled with SQL. Query generation for other persistency devices then sql database might be easy to implement.

11.7 Database support

Last couple of releases supported the following relational databases:

- Informix (VPRO and EO, cloud with millions of objects)
- PostgreSQL (Kennisnet, Gemeente Amsterdam)
- MySQL (Too many to choose)
- Hsql (Default distribution)
- Oracle (Vodafone is the only serious implementation)

11.8 MMBase Core

Tasks:

- Configuration management
- Starting Subsystems
- Initialization of Object model
- Loading Modules

11.9 Security

The security models available in MMBase are:

- No security
- Cloud Security
- Context Security
- Context Cloud Security

MMBase offers an elaborated security layer for authorization and authentication. Authorization tells what a user is allowed to do. Authentication tells how a user can log on to an MMBase website. The security models available in MMBase are:

No security

Although trivial, it is sometimes handy to have an MMBase installation with no authentication, no authorization or no security at all. It won't come as a surprise that MMBase supports this functionality

Cloud Security

MMBaseusers are administrated in the MMBase object cloud. This security model offers a basic ranking system in Anonymous, Users and Admin. An advantage of the Cloud Security are that you can do your user administration via the web interface and that you are able to integrate the MMBase users into your object model.

Context Security

MMBase users are administrated in XML-configuration files on the file system. Users are part of Groups and Ranks. Each object in MMBase belongs to a context. The Context Security model offers an fine-grained mechanisms to specify rights of users and groups in different contexts. The model provides the possibility to define basic workflow schemes. The Context Security is especially fit if the rights of users has to be specified on object level or if you need basic workflow functionality. drawback of the context security model is that the user administration can not be carried out via the web interface, but one has to use the XML-configuration files

Context Cloud Security

The Context Cloud Security provides the same functionality as the Context Security. In contrast to the Context Security, the Context Cloud security stores the authentication and authorization information in the object cloud. This implementation removed the drawback of the context security, because the user administration can be carried out via the web interface.

11.10 Bridge

Also know as the MMBase Connection Interface (MMCI)

- A man in the middle
- Set of Java interfaces
- Its functionality should not change often.

MMBase has its own templating/scripting language (SCAN). MMBase had to deal with another one when Java Server Pages and tag libraries were introduced. The SCAN language was tightly integrated with the rest of the system. Even some parts of the database layer knew what html was. The developers of the MMBase taglib decided to create an layer around the core which the taglib could use to interact with the system. This layer has become the bridge/MMCI.

The bridge is a set of Java interfaces which are to be used to interact with MMBase. When you understand these interfaces you will understand most of the core concepts of MMBase. The interfaces are kept very simple, but contains the functionality you need for day-to-day usage.

The interfaces have not been changed for a long time. Many applications which upgrade to newer versions of MMBase only require small changes to work again. The core has had major changes, but the bridge has successfully hidden these changes.

11.11 Extensions (plugins)

Widely used extensions

- Taglib
- JSP-editors
- R-MMCI
- Editwizards

There exist a growing list of extensions which add functionality to the MMBase Core. The most important extensions are listed below.

Taglib

The taglib provides a set of tags which facilitate communication between MMBase and JSP-templates.

JSP-editors

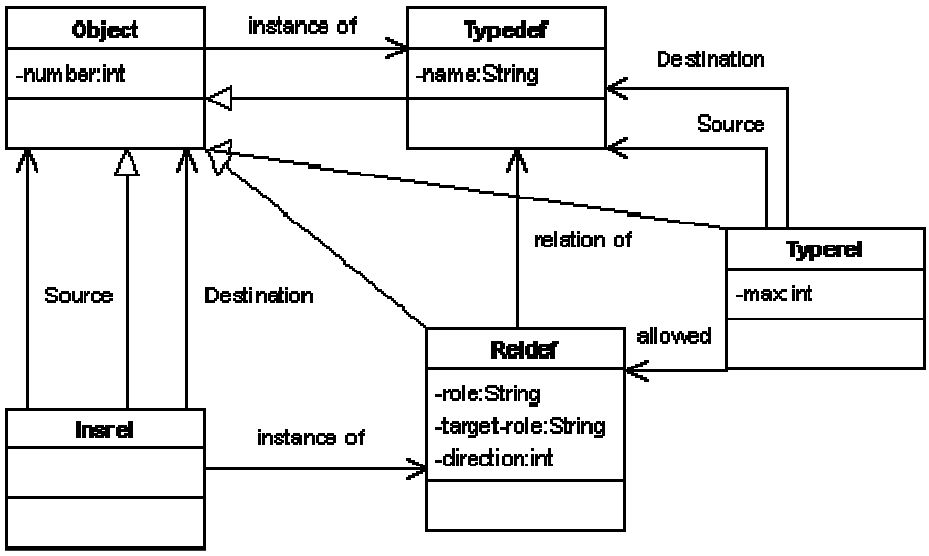
One widely used tool build with the taglib are the JSP-editors for MMBase. Several editor versions are available and some of them are distributed with the MMBase core

R-MMCI

The Remote MMBase Cloud Interface makes it possible to communicate with different MMBase installation at the same time. This powerful plugin makes it possible to share and exchange data between MMBase installations irrespective of their location.

Editwizards
The editwizards provide the functionality to define forms to edit the MMBase object cloud. Editors who do not frequently use MMBase might get lost in their MMBase object cloud, when using the default JSP editors to enter or change content, With the editwizards it is possible to build task oriented forms, which guide editors through their work.

11.12 Core Data Model



Just admire the UML diagram and learn it by heart. All information in MMBase is stored in this model. Even the object types from this model are stored in it.

- object: instances of persisted information
- insrel: instances of relations between persisted information
- typedef: definition of persisted information (object type)
- reldef: definition of relation between persisted information
- typerel: defines which relation definition is allowed between object types

11.13 Builders / Applications

Builders

- Name, maintainer and version
- Classfile
- Names and descriptions
- Properties
- Fields (gui and db)

Applications

- Needed builders
- Needed reldefs
- Allowed relations

MMBase only requires a configured database connection, but it has to know what it has to create. MMBase reads builder and application xml files from the configuration. The builder xml defines an object type. The application xml defines a user object model.

Builder

A builder defines the following for an object type:

- Name: The name of the builder, which should currently be equal to the name of the file
- Maintainer: The maintainer of the builder. This is mandatory, and should contain the name of the organization maintaining the builder. This is generally a domain name. For the core builders this value is mmbase.org
- Version: The version number of the builder
- Extends: The name of a builder that this builder extends. If you do not specify a builder, you should fully describe the builder. If you do specify a parent builder, you need only describe new or changed tags or attributes. The default builder that is extended is object.
- Classfile: The java class used by this builder to administrate nodes of that builder type. Specify this tag if you have custom behaviour defined for an object type.
- Names and descriptions: These are used to supply gui (graphical user interface) names for the builder. The information is generally used in generic editors, as well as the default value for use in the editwizards. You can include gui names and descriptions for different languages. the system picks the correct name depending on its currently specified locale.
- Properties: A list of name-value pairs, used by the java class specified in the classfile as configuration parameters. You normally only specify properties for objects with specific behaviour (such as Images).
- Fields: A list of fields where the object can store its information in. The field definition is divided into two sections. A gui section for names and types to use for display and a db section for the attributes to persist the information (size, datatype, key, nullable, etc.).

Application

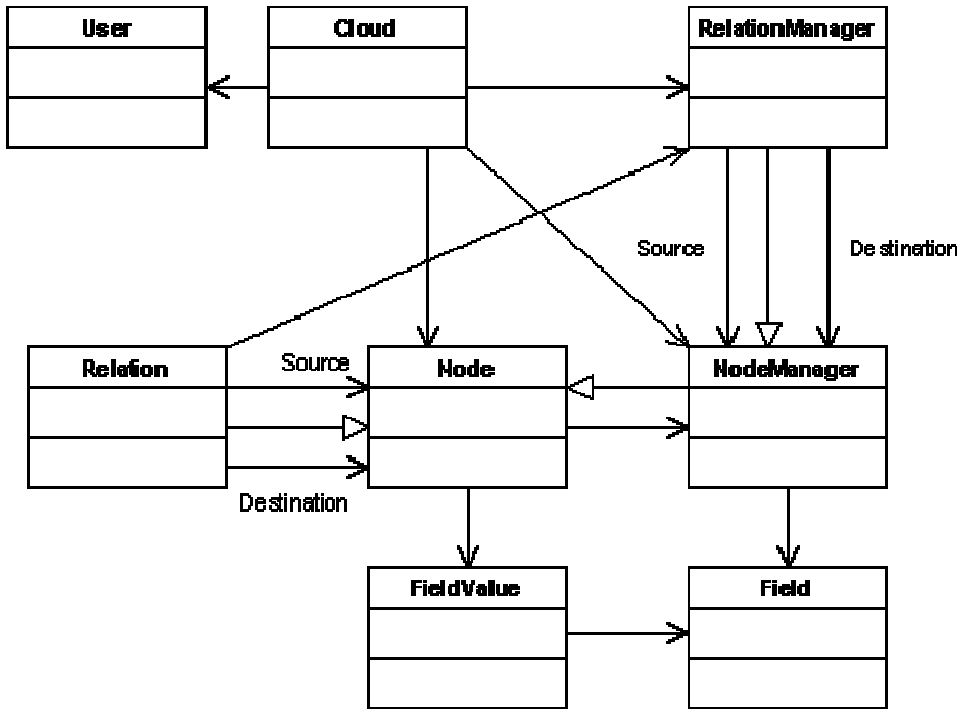
An application defines the following for an user object model:

- Name: The name of the applications, which should currently be equal to the name of the file
- Maintainer: The maintainer of the application. This is mandatory, and should contain the name of the organization maintaining the application. This is generally a domain name. For the core applications this value is mmbase.org
- Version: The version number of the application
- Needed builders:: The needed builder list contains a list of all the builders that are necessary to run this application. Both the builders of regular objects and the builders used to create relations have to be listed here.
- Needed reldefs: The neededreldef list contains a list of all relation roles that have to be defined for this application. When you use the MMBase editors to have a look at the RelationDefinition objects in your MMBase installation, you will see how closely the "reldef"

descriptions follow these objects. The most import attributes of every reldef below neededreldeflist are `source' and `builder'. The function of a `reldef' is to connect those two. Another word for 'source' is 'role' (as it is called often for example in taglib attributes). The `builder' refers to the actual database table in which the relations can be stored, and therefore defines which fields the relations of this type have.

- Allowed relations: The allowedrelation list contains a list of what type of relation (what roles) should be used to relate two object of specified type. When you use the MMBase editors to have a look at the RelationType objects in your MMBase installation, you will see how closely the "relation" descriptions follow these objects. The `from' and `to' attributes both refer to builder names (and could also have been called sourcebuilder' and `destinationbuilder' or `sourcenodetype' and `destinationnodetype'). The `type' attribute then actually defines the role for this allowed relation, so these correspond to the `source' attributes of the reldef.

11.14 Bridge Object Model



Core / Bridge Mapping

Core	Bridge
Object	Node
Typedef	NodeManager
Insrel	Relation

Core	Bridge
Reldef	RelationManager
Typerel	RelationManager

Only two things are real different from the Core Data Model.

- There is a cloud and user object. The cloud object is used as a viewport over the objects in the core for that particular user. The security layer of MMBase will filter out all the objects the user is not allowed to see.
- The relation definition and allowed relation is mapped in the relationmanager.

The field objects are not really modelled in the data model. They are modelled in the builder xml files which belong to a typedef node. The Core *Object* Model has a fielddef object which is wrapped by the bridge field object.

11.15 Core Typedefs (Basic application)

- Daymarks
- MMServers
- OAlias
- SyncNodes
- Versions

Resources application

- attachments
- urls
- images and icaches
- posrel

Daymarks
Daymarkers are used to calculate the age of MMBase objects. Every day a daymarker is added to the daymarks table. Such an entry consists of a daycount (number of days from 1970), and a count (current object number of that day).

MMServers
MMServers stands for MMBase servers. It is possible to run multiple MMBase servers on one database instance. Every mmserver node represent a real MMBase server (think of it as a machine where one instance of MMBase is running). On start-up MMBase queries the mmservers table and looks if he is listed in the list of mmservers, If not MMBase create a new node containing information about itself (name/host/os/jdk). The mmservers builder has extra behaviour, it can communicate with other servers(using multicast). The basic functionality it provides however is sending information

about changes of node to other mmservers (Listen !! I just have changed node 123). This mechanism makes it possible to keep nodes caches in sync but also makes it possible to split tasks between machines. You could for example have a server that encodes video. when a change to a certain node is made the servers can start encoding the videos.

OAlias
The OAlias builder is an optional core builder used to associate aliases with nodes. Each OAlias object contains a name field (the alias), and a destination field (the number of the object referenced). MMBase will return the object when the alias is requested.

SyncNodes
Syncnodes are used to keep track of imported nodes that belong to a other systems. syncnodes can be used in later stage to "re-import" nodes or create new relations

Versions
The version objects are used to keep track of installed builder and application versions. MMBase will check the versions and will install the new builders and applications.

Resources Application
The resources application is used in most projects, MMBase has several servlets which serve the binary objects and can manipulate or analyse them for metadata.

11.16 Extending builders

Builder class methods:

- init() - builder loading. (read properties)
- insert() - new node create.
- commit() - node committed
- getValue() - implement virtual fields

The builder class methods are called when lifecycle operations are performed on the builder or node (which the builder manages).

Below are some methods which can be used inside the builder

- public String getInitParameter(String name)
- public void setDefaults(MMObjectNode node)
- public int getObjectType()
- public MMObjectNode getNewNode(String owner)
- public void removeNode(MMObjectNode node)
- public MMObjectNode getNode(String key)
- public MMObjectNode getNode(int number)
- public MMObjectNode getHardNode(String key)
- public MMObjectNode getHardNode(int number)

- public boolean checkAddTmpField(String field)
- public List getNodes(NodeSearchQuery query)

11.17 Modules

An piece of (MMBase) functionality which is configured by a XML file in ‹config›/modules/.

MMBase loads modules at start-up. Modules can be used to implement services like resource management (database connections, email queues, etc.) or maintenance tasks.

A special module is the mmbaseroot module which is MMBase itself. This module will be loaded first and initializes the subsystems like configuration management and storage layer.

11.18 Observers

Observers belong to the category of "very cool but unknown features of MMBase".

An observer can register itself as a change listener on a builder. The builder will signal the observer when an object is changed.

The Observer interface has two methods:

- nodeLocalChanged()
- nodeRemoteChanged()

Only a few people in the community know that they exist. A lot of people create hacks by extending builders to implement the observer functionality. The core of MMBase relies on this functionality, because the caches are cleared by it.

The nodeLocalChanged() is called when the node is changed in the local MMBase server. The nodeRemoteChanged() is called when a remote MMBase server changed the node in the shared database.

11.19 Taglib

A taglib is just an extension to HTML which give a developer the possibility to enhance their web pages with server-side functionality

- MMBase tags (cloud, node, field, ...)
- security tags (maycreate, maywrite, maydelete, ...)
- utility tags (compare, formatter, isempty, ...)
- context/variable tags (import, present, remove, ...)
- list tags (list, related, first, last, ...)
- tree tags (tree, grow, depth, ...)

11.20 JSP-editors

There are several editors for MMBase implemented with the taglibs which use the node/field metadata to display the input forms and list of relations.

- Scan-look editors
- MMEditors
- My_editors
- preditors

11.21 R-MMCI (Remote MMBase Cloud Interface)

RMMCI is a set of java files that makes it possible to connect to an MMBase cloud using the MMCI (MMBase Cloud Interface) without the need to start your own MMBase Cloud instance.

Developers of the MMBase community have created an interface called MMBase Cloud Interface (MMCI). It is possible to access most of the features of MMBase with the MMCI. The interface is there to provide abstraction between how MMBase is implemented (database/Java code) and the outside world. This Local MMCI implementation uses core MMBase classes to implement the MMCI and is widely used.

RMMCI uses the decorator pattern to intercept every call to the MMCI, sends the call over a network (using RMI) and at server side RMMCI calls the Local implementation.

11.22 Editwizards

The editwizards allow designers to easily create wizard-like web-entry forms for content entry and maintenance.

- Wizard definition schemas
- Object and relation actions (CRUD)
- Input validation
- Highly customizable front end based on xsl and css

The editwizards extension can be divided into 2 layers. The first layer is the server side engine which consists of java and jsp files. The engine communicates with MMBase and generates the data for the second layer. The second layer is the front end, which runs on the server and in the client browser.

The engine communicates with MMBase through the dove interface. The dove interface adds a protocol for communication with, and passing commands to, the MMBase system using an XML format. The output from the engine to the front end is an XML stream with the data to build the html pages. The engine is not only a pass-through for data. It also keeps track of stacks with opened wizards for every user and parses the wizard definition schema files.

The editwizards generates five types of pages. Three of them are used in normal operation. These are the list, wizard and searchlist pages. Most of the times, you go from a list page to a wizard page and then you add related objects with the searchlist page to the wizard. The ones left over are the debug and error (exception) pages. The debug page shows editwizard engine internal data which could be handy to use when changing the front end xsl's. Hopefully, you will never see the error page. If you look in some directories then you will see files with the same names as the types of the pages.

11.23 References

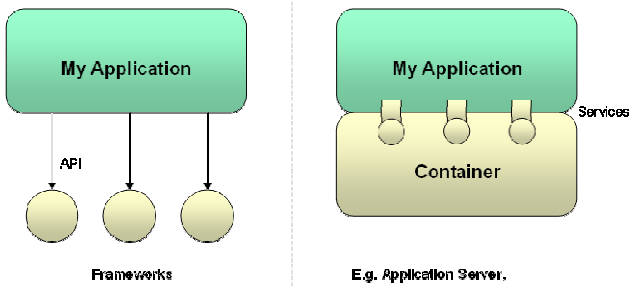
- MMBase site: http://www.mmbase.org
- Documentation: http://www.mmbase.org/mmdocs
- Bugtracker: http://www.mmbase.org/bugs
- Mailing lists: http://lists.mmbase.org
- CVS: http://cvs.mmbase.org

{12} Append B: Maven explained

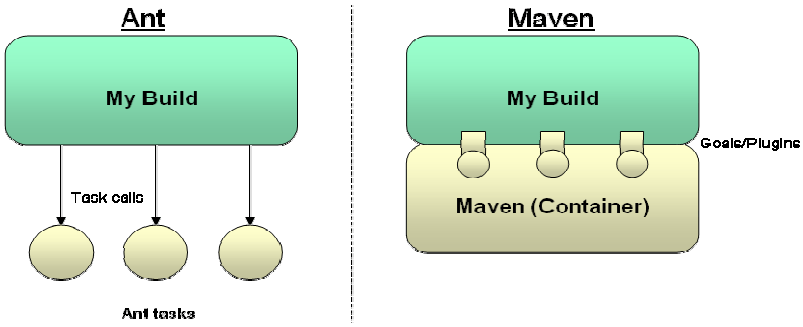
12.1 What is Maven?

Maven is a Java project management and project comprehension tool. Maven is based on the concept of a project object model (POM) in that all the artefacts produced by Maven are a result of consulting a well defined model for your project. Builds, documentation, source metrics, and source cross-references are all controlled by your POM

Developing a build system is almost the same as with an application. An tradition application was composed of several API's and frameworks. Nowadays, an application is developed inside a container which provides several services.



A build system with ant is composed out of tasks calling each other. Maven takes the container approach and provides some services to a build..



Advantages of a Container approach

- Ability to reuse existing services
- Easy to start with - configure and extend
- Well-defined and shared structure/architecture
- Easier to maintain

Note: It makes sense to use a custom approach when flexibility is a must. This is not usually the case for builds

Maven can be summarized in 3 concepts.

- POM (Project Object Model)
- Plugins - Made of Goals = Actions
- Repositories

Maven files

- project.xml - POM
- maven.xml - Custom goals (Goal, preGoal, postGoal)
- project.properties - Project Configuration
- build.properties - System/User Configuration

12.2 POM (Project Object Model):

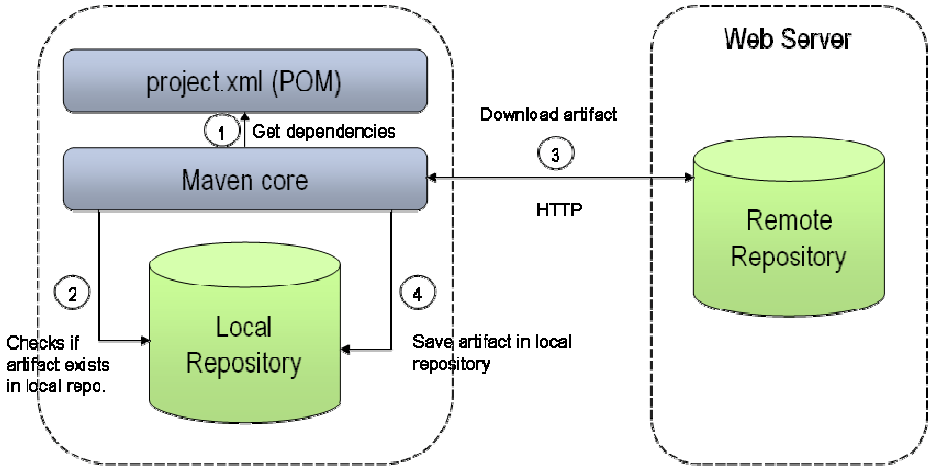
In short, the pom has the following information

- Verbose project description
- Company/Community information
- Mailing list and CVS/Subversion server access
- Build information
    - Source code and unit test code location
    - Resources needed for a build
- External dependencies
- Project team:
    - Developers
    - Contributors
- Versioning
- Reports

12.3 Repositories

This concept is very important to understand. This is where the CMS container is going to rely on.
A maven repository is just an directory of webserver with a special layout
The maven system will do the following steps to resolve dependent artefact (jar, war, ear, zip).

1. Get the dependencies from the POM
2. Search for these dependencies in the repository on the local filesystem
3. When not present local then connect to remote repositories and download the artefacts
4. Store the artefact in the local repository for future usage.



12.4 Goals

- Compares to "targets" in Ant and Make
- Adding preconditions and postconditions to an existing goal
- Goals written in Jelly (jakarta.apache.org/jelly)
  - Executable XML
- Custom goals for each project
  - maven.xml

12.5 Plugins

- Functionality implemented as plugins
  - Java compilation, jar-creation, junit-testing,
- Downloaded on demand
  - May be listed as project dependencies
- Do not reinvent the wheel
  - Uses Ant for building
  - Uses Checkstyle for code convention conformity checks
  - Uses Junit for unit testing
  - Uses Javadoc for code documentation
  - ....

- Ant
- Appserver
- Artifact Plugin
- Ashkelon
- AspectJ
- AspectWerkz
- Abbot
- Caller
- Castor
- Changelog
- Changes
- Checkstyle
- Clean
- Clover
- Console
- Cruise Control
- Dashboard
- Developer Activity
- Distribution
- DocBook
- EAR
- Eclipse
- EJB
- FAQ
- File Activity
- Genapp
- Gump
- Hibernate
- Html2XDoc
- IDEA
- J2EE
- Jalopy
- Jar
- Java
- Javacc
- Javadoc
- JBoss
- JBuilder
- JCoverage
- JDEE
- JDepend
- JDeveloper
- JDiff
- JellyDoc
- Jetty
- JIRA
- JNLP
- JUnit
- JUnitDoclet
- JXR
- Latex
- Latka
- License
- LinkCheck
- MultiChanges
- Multiproject
- Native
- NSIS
- PDF
- Plugin
- PMD
- POM
- Release
- Repository
- RAR
- Source Control Management
- Shell
- Simian
- Site
- Struts
- Tasklist
- Test
- TJDO
- Touchstone
- Touchstone Partner
- Uberjar
- VDoclet
- WAR
- Webserver
- Wizard
- XDoc

... and more ...

12.6 Extending Maven

- First level of customization
  - Edit project descriptor and project.properties
  - Project.xml, project.properties, build.properties
  - Zen of Maven: parameterize, not rewrite
- Second level of customization
  - Create a maven.xml file
  - Set pre and post conditions of existing goals
  - Write new goals
- Third level of customization
  - Create a plugin and share it

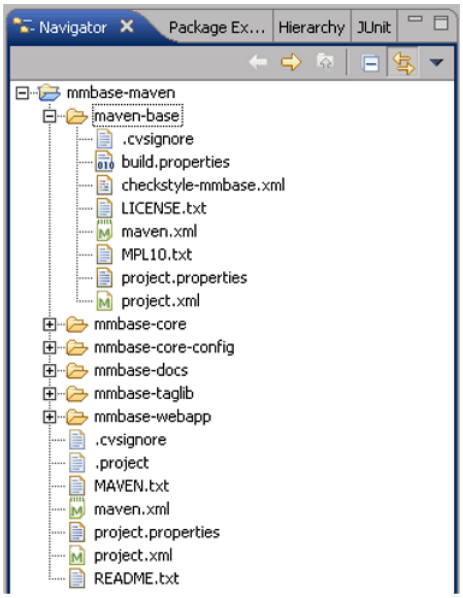
12.7 Multiproject support (subproject)

- Different levels of granularity for project management
- Developer-level: runs build on 'local' subproject during development

Maven has:

- POM inheritance mechanism
  - ‹extend›${basedir}/../project.xml‹/extend›
  - Define common properties in master project
  - Add or override subproject specific properties
- Multiproject plugin
  - Triggers build of all dependent projects
  - Auto-detection of the required build order
- One main artefact per (sub)project

12.8 Multiproject example



Use a common directory to share build-related resources

- Shared Maven POM (project.xml)
- Shared custom Maven goals (maven.xml)
- Shared Maven properties (project.properties)
- Checkstyle files
- Licenses

12.9 Continuous Integration

- Catch integration problems at the earliest possible time
- Most bugs caught within the day they were introduced
- Reduces bug search scope
- Reduces risk at an early stage.

Maven has:

- Automated and highly reusable build script
- Code maintained in a central location (CVS/Subversion support)
- Unit test run included out of the box
- CI configuration generation

12.10 SNAPSHOT versions

In Maven SNAPSHOTs are artefacts approximate the latest build of a particular project.
SNAPSHOT actually means search for latest artefact in all repositories
Developers only have to work on their subproject in multi project development, because Continuous Integration can deploy the latest successful builds in the remote repository
A developer can choose to work with his current snapshot through the -o (offline) option

12.11 Deployments/Releases

Source Control Management (SCM) plugin provides basic release functionality

- Preparing a release
    1. Verify that there are no uncommitted changes in the checkout
    2. Prompt for a desired version and tag name

3. Modify project.xml and changes.xml to set the released version and commit the changes
4. Tag the entire source tree with the new tag name

Create your own procedures for multi projects and implement them in a plugin.

12.12 References

- Maven site: http://maven.apache.org
- Documentation: http://maven.apache.org/guides/

{13} Appendix C: JSR-170 Java Content Repository

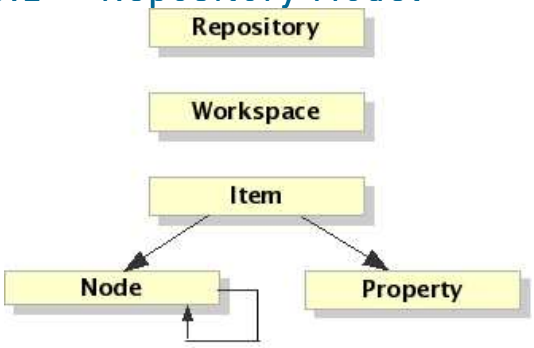
13.1 What is JSR-170 specification

JSR-170 is a Java Specification Requests for Content Repository. Because more and more vendors offer various proprietary content repositories, the need for a common programmatic interface to these repositories has become apparent. The aim of the JSR-170 specification is to provide such an interface and, in doing so, lay the foundations for a true industry-wide content infrastructure. Application developers and custom solution integrators will be able to avoid the costs associated with learning the particular API of each repository vendor. Instead, programmers will be able to develop content-based application logic independently of the underlying repository architecture or physical storage. Customers will also benefit by being able to exchange their underlying repositories without touching any of the applications built on top of them.

JSR-170 specification defines a standard, implementation independent, way to access content bi-directionally on a granular level within a content repository. It proposes that content repositories have a dedicated, standard way of interaction with applications that deal with content. The API focus on transactional read/write access, binary content (stream operations), textual content, full-text searching, filtering, observation, versioning, handling of hard and soft structured content.

This specification has been split into two compliance levels as well as a set of optional features. Level 1 defines a read-only repository. This includes functionality for the reading of repository content, introspection of content-type definitions, basic support for namespaces, export of content to XML and searching. This functionality should meet the needs of presentation templates and basic portal applications comprising a large portion of the existing code-base of content-related applications. Level 1 is also designed to be easy to implement on top of any existing content repository. Level 2 additionally defines methods for writing content, assignment of types to content, further support for namespaces, and importing content from XML. Finally, a number of independently optional features are defined that a compliant repository may support. These are transactions, versioning, observation, access control, locking and additional support for searching.

13.2 Repository Model



The JSR-170 defined a generic repository concept model. A content repository consists of one or more workspaces, each of which contains a tree of items. An item is either a node or a property. Each node may have zero or more child nodes and zero or more child properties. There is a single root node per

workspace, which has no parent. All other nodes have one parent. Properties have one parent (a node) and cannot have children; they are the leaves of the tree. All of the actual content in the repository is stored within the values of the properties.

13.3 Repository Features

This specification is divided into two compliance levels and a set of additional optional features which repositories of either level may support. Level 1 provides for read functions and level 2 adds additional write functions. The functional division is as follows:
Level 1 includes:

* Retrieval and traversal of nodes and properties
* Reading the values of properties
* Transient namespace remapping
* Export to XML/SAX
* Query facility with XPath syntax
* Discovery of available node types
* Discovery of access control permissions

Level 2 adds:

* Adding and removing nodes and properties
* Writing the values of properties
* Persistent namespace changes
* Import from XML/SAX
* Assigning node types to nodes

Optional: Any combination of the following features may be added to an implementation of either level.

* Transactions
* Versioning
* Observation (Events)
* Locking
* SQL syntax for query