CQ5 WCM Architect Guide

Day

CQ5 WCM Architect Guide

Contents

1 Introduction

1.1 Introduction

Day Management AG's CQ5 platform allows you to build compelling content-centric applications that combine Web Content Management, Workflow Management, Digital Asset Management and Social Collaboration.

The product has been completely redesigned from Communiqué 4, allowing Day Management AG to use new architecture and technologies, thus increasing functionality while reducing complexity. Extensive use of standards helps reduce project risk, and protect your investment in the long-term.

The Graphical User Interface has been completely re-engineered using AJAX and the latest browser technologies. This leads to an unparalleled user experience. With the help of the Apache Sling framework (a core part of the CQ5 platform) it becomes fast and efficient to develop *content-enabled vertical applications* (CEVA) that are both JCR-based and REST-style.

The Java Content Repository (JCR) is fully leveraged with many important features; including search, versioning, access control and observation.

The CQ5 architecture allows the move to a *Data First* philosophy as:

- structure Is an expensive overhead
- for the underlying storage, any data structure is (mostly) irrelevant
- see Chapter 5,

This vastly improves flexibility and efficiency.

All these changes ensure that CQ:

- is easily scalable
- is highly reliable
- makes it easier to reuse, or extend, elements
- is faster
- allows for rapid development
- is already feature-rich when taken straight out-of-the-box

1.2 Purpose of this Document

To provide information about the architecture of CQ, together with the concepts and technologies used.

1.3 Target Audience

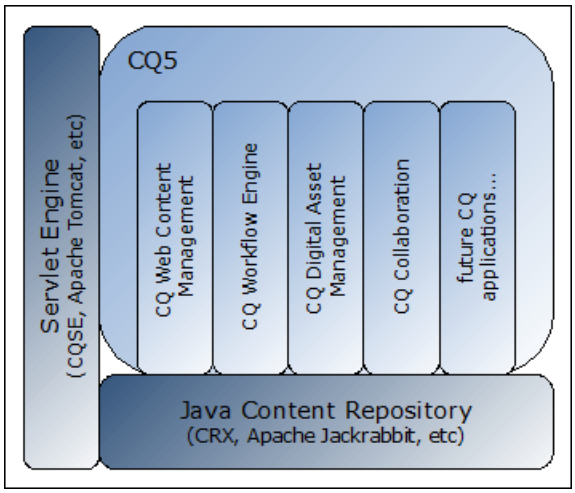
- System Architect
- Project Manager
- Technical Manager
- Developer
- System Administrator

2 CQ - The Concepts

2.1 An Overview of CQ5

The following diagram illustrates the interrelationship between CQ and other operational elements; which may be products from Day Management AG, or their third-party equivalents:

Figure 2.1. CQ5 Overview



Servlet Engine

The Servlet Engine acts as the server within which each CQ (and CRX if used) instance runs as a web application.

Any Servlet Engine supporting the Servlet API 2.4 can be used.

Java Content Repository (JCR)

A Java Content Repository uses the JSR-170 API to access the content repository using Java, independent of the physical implementation. **JCR** is the **J**ava **C**ontent **R**epository standard, also known as **JSR-170** after its Java Specification Request.

A repository effectively consists of two parts:

- A Web application that offers the JSR-170 compliant API and temporary data storage (in the form of the session).
- A Persistence Manager with persistent data storage, such as the file system or a database.

Content Repository Extreme (**CRX**) is Day Management AG's own repository product. See the CRX documentation for more details; including direct access using WebDAV, CIFS, File Vault etc.

CQ5

The common foundation of the CQ5 platform provides a basis for the interoperability and seamless integration of all CQ applications. This is available to both:

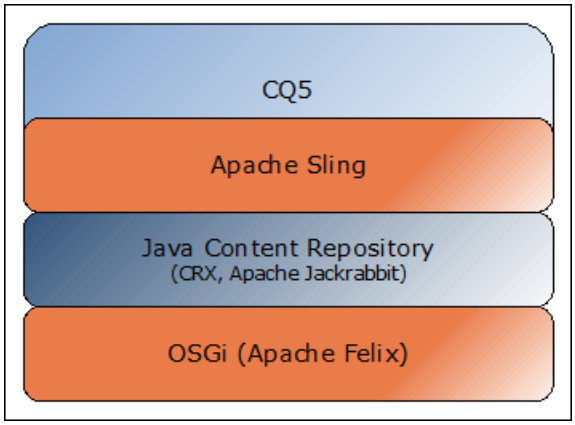
- the applications that are integral to CQ itself
- any customized applications developed for the CQ5 platform.

CQ WCM (Web Content Management) is the first application developed to exploit the advantages of CQ5, other Day products will be migrated in the near future (although they already interact with CQ WCM). These include Digital Asset Management (CQ DAM) and Social Collaboration.

2.2 The Technology Stack that CQ5 is based on

CQ5 is based on new technologies including:

Figure 2.2. CQ5 Technology Stack



Apache Sling

Apache Sling is a web application framework for content-centric applications, using a Java Content Repository, such as Apache Jackrabbit or CRX, to store and manage content.

Sling:

- is based on **REST** principles to provide easy development of content-oriented applications.
- is embedded within CQ5.
- is used to process HTTP rendering and data-storage requests which assemble, render and send the content to a client (i.e. the *new delivery*).
- maps Content objects to Components (which render them and process incoming data).
- comes with both server-side and AJAX scripting support.
- can be used with a range of scripting languages, including JSP, ESP and Ruby.
- started as an internal project of Day Management AG .
- has been contributed to the Apache Software Foundation.

Note

See http://incubator.apache.org/projects/sling.html for more information.

OSGi (Apache Felix)

CQ5 is built within an application framework which is based on the OSGi Service Platform Release 4.

OSGi technology

- “is the dynamic module system for Java™.”
- comes under the classification Universal Middleware.
- “provides the standardized primitives that allow applications to be constructed from small, reusable and collaborative components. These components can be composed into an application and deployed.”

- OSGi bundles can contain compiled Java code, scripts, content that is to be loaded in the repository, and configuration or additional files, as needed.
- allows the bundles to be loaded, and installed, during normal operations. In the case of CQ5, this is managed by the Sling Management Console.

Apache Felix has been used to implement this framework.

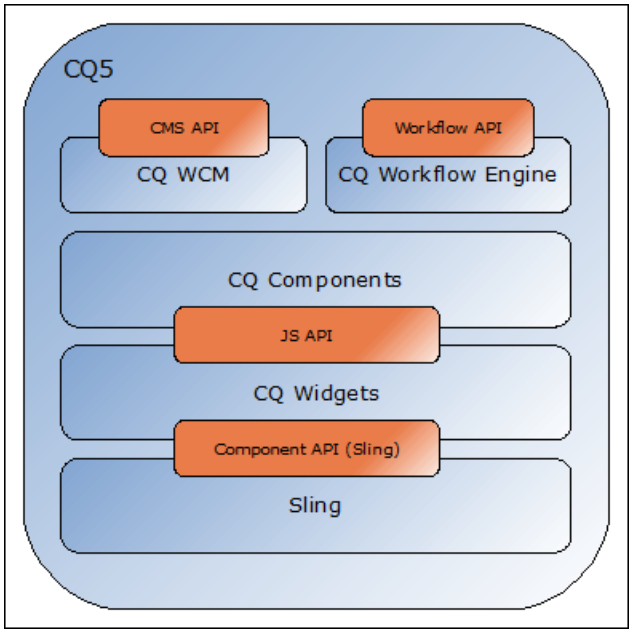
- "Apache Felix is a open-source project to implement the OSGi R4 Service Platform, which includes the OSGi framework and standard services, as well as providing and supporting other interesting OSGi-related technologies."

Java Content Repository (JSR-170 API)
: A JCR uses the JSR-170 API to access the content repository using Java, independent of the physical implementation.

2.3 Inside CQ5

CQ5 forms a stable platform for content-centric applications such as CQ WCM:

Figure 2.3. CQ5 Internal Layers



CQ WCM
: Web Content Management within the CQ5 platform allows you to generate and publish pages to your website..

CQ Workflow Engine
: The CQ Workflow Engine is a powerful and easy to use process engine that can be used by all applications running on the CQ5 platform. A Java API and RESTful HTTP interface is also provided for access by applications outside CQ5.

  Within CQ WCM workflows can be used to control the process of generating and publishing content, which are often subject to organizational processes, including steps such as approval and sign-off by various participants.

CQ Components
: Components provide the logic (code) to render content. They include both templates and specific components such as Text with Image, Column Control and Subtitle amongst others. Components are based on a combination of widgets, replacing the CFC from Communiqué 4.

CQ Widgets
: Widgets are the basic elements used to implement a specific user function, often the editing of a piece of content; they include buttons, radio-boxes, dialogs, etc.

Apache Sling
: The Component Framework (Sling) provides the underlying mechanisms for rendering content.

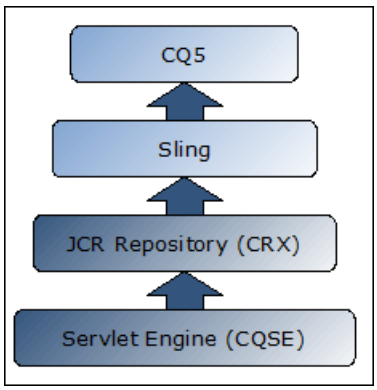
3 CQ5 - The Physical Architecture

3.1 Server Startup Sequence

CQ5 has drastically reduced startup time as compared to Communiqué 4. For CQ WCM (straight out-of-the-box) the startup time is now in the order of 5 seconds.

The elements required for CQ WCM are started in the following sequence (bottom-up):

Figure 3.1. Server Startup Sequence



3.2 CQ WCM Environments and how they interact

3.2.1 Author and Publish Environments

A CQ WCM installation usually comprises of multiple instances, used for different purposes. Content, including code and other resources held in the repository, can be replicated (copied from destination to source) using a HTTP, or HTTPS, connection.

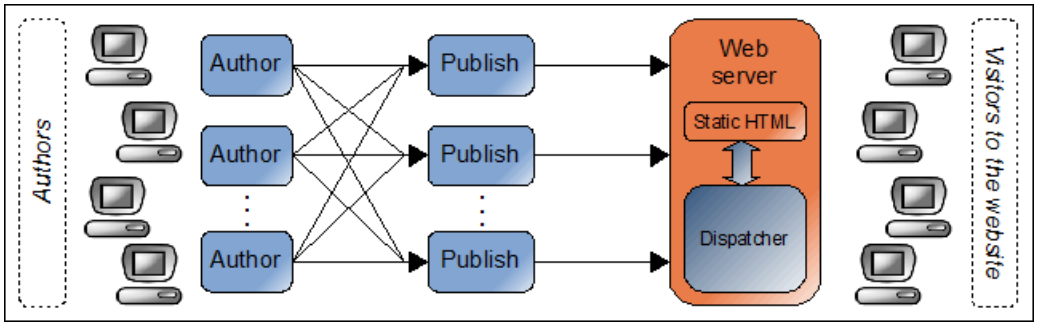
A production environment often consists of two different types of instances:

author
: This is the environment where you, and your colleagues, will input your content and administrate the system.

publish
: This environment holds the content you have made available to visitors to your website; be it public, or within your intranet.

The following diagram gives an overview of typical configurations possible for the various CQ WCM environment. It shows how content flows from the authoring environments until it is available to be accessed by visitors to your website. It also highlights the fact that to increase performance and availability it is common to combine several authoring and publishing environments to service a website.

Figure 3.2. CQ WCM Environments



Note

This diagram covers a range of possible configurations, with multiple environments of either sort. Depending on your configuration, each author environment can propagate content to one, or more, publish environments.

Author
: This is the environment where you and your colleagues will:

- administrate the entire system
- input your content
- configure the layout and design of your content
- activate your content to the publish environment

It is accessed using the **siteadmin**. Access to the content and functionality is controlled by authorization permissions assigned to your user account.

Publish
: This holds the content which you have made available to visitors to your website.

The content is dynamic, real-time and can be personalized for each individual user.

Static Web Server
: For performance optimization it is possible to convert your dynamically published content (excluding any personalized parts) to static HTML, serviced by a static web server. Static web servers are very simple, but fast. Examples include Apache, and IIS.

The Dispatcher can then be used in conjunction with the web server to realize an environment that is both fast and dynamic and with moderate hardware requirements.

Dispatcher
: The Dispatcher helps realize an environment that is both fast and dynamic. It works as part of a static HTML server, such as Apache, with the aim of:

- storing (or "caching") as much of the site content as is possible, in the form of a static website.
- accessing the layout engine to retrieve dynamic content as and when necessary, but as little as possible.

Which means that:

- **static content** is handled with exactly the same speed and ease as on a static web server; additionally you can use the administration and security tools available for your static web server(s).

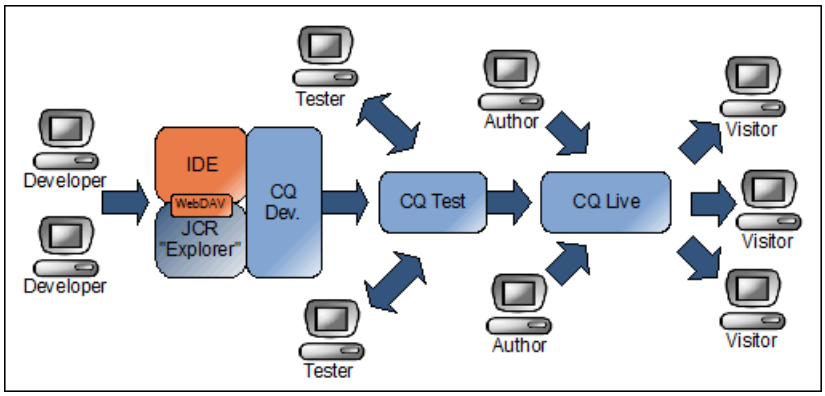
- **dynamic content** is generated as needed, without slowing the system down any more than absolutely necessary.

The Dispatcher contains mechanisms to generate, and update, static HTML based on the content of the dynamic site. You can specify in detail which documents are stored as static files and which are always generated dynamically.

3.2.2 Environment levels used within the full development cycle

As with any software, the projects developed using CQ WCM are subject to the development cycle. Therefore various environment levels are required to cover the development and testing phases. Depending on your requirements, each level may be configured for either only an author, or an author and publish environment.

Figure 3.3. Development and Test Environment Levels



Note

Multiple instances of each environment level can exist.

Development

Prior to authors registering their content in CQ WCM, the developers are responsible for developing and customizing the proposed website. They:

- develop and customize components
- realize the design within the website
- develop the necessary scripts to implement the required functionality of the website

The major development tools used are:

- an Integrated Development Environment.

  Day provides an Eclipse-based development environment, CQDE (the Communiqué Development Environment). It is also possible to use other IDEs, such as Eclipse or IntelliJ, for which plug-ins have developed to simplify their use for CQ and to integrate them with the repository.

- a method of direct access to the Java Content Repository.

  In the case of CRX, the Content Explorer, Content Loader and other in-built tools are used.

- WebDAV or CIFS, which simplify access to the repository.

Depending on the scale of your system, the development environment can have both **author** and **publish** environments, or the test environment will be used for such functionality.

Test
: After development, it is usual to have a Testing environment where you can access the new system, to test both design and functionality.

  This will often comprise of both an **author** and **publish** environment.

  Day provides a basis for automated GUI tests, together with some reference test scripts.

Live / Production
: As discussed previously, the Live (or Production) environment comprises both:

  - an **author**ing environment for the input of content
  - a **publish** environment for content made available to visitors to the website

3.3 Performance and Availability

3.3.1 Caching

There are two basic approaches to web publishing:

Static Web Servers
: Are very simple, but fast.

  A static web server, such as Apache or IIS, serves static HTML files to visitors of your website. Static pages are created once, so the same content will be delivered for each request. If a visitor requests a file (e.g. a HTML page), the file is usually taken directly from memory, at worst it is read from the local drive.

  This process is very simple, and thus extremely efficient, but does not cater for personalization or dynamic content.

Content Management Servers
: Provide dynamic, real-time, intelligent content, but require much more computation time and other resources.

  CQ is such a server, whereby an advanced layout engine processes the request from a visitor. The engine reads content from a repository which, combined with styles, formats and access rights, transforms the content into a document that is tailored to the visitor's needs and rights. This allows you to create richer, dynamic content, which increases the flexibility and functionality of your website.

  However, the layout engine requires more processing power than a static server, so this setup may be prone to slowdown if many visitors use the system.

The Dispatcher is used to combine the 2 philosophies, on the principle of:

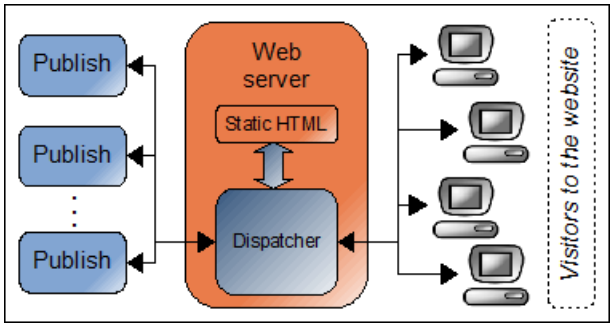
- if the cached document is newer, the Dispatcher returns it
- if older, the Dispatcher retrieves the current version from the appropriate CQ WCM instance (see also Section 3.3.2, “Load Balancing”)

This allows you to take full advantage of a static web server, while retaining the capability to process dynamic content as and when necessary.

3.3.2 Load Balancing

The Dispatcher keeps internal statistics about how fast the web servers are processing documents. Based on this data, the Dispatcher estimates which server will provide the quickest response time when answering a request, and so it assigns the necessary request, and computational time, to that server.

Figure 3.4. Load Balancing



You gain:

Increased processing power
: In practice this means that the Dispatcher shares document requests between several web servers. Because each server now has fewer documents to process, you have faster response times. The Dispatcher keeps internal statistics for each document category, so it can estimate the server load and distribute the queries efficiently.

Increased fail-safe coverage
: If the Dispatcher does not receive responses from a web server, it will automatically relay requests to the other server(s). Thus, if a server becomes unavailable, the only effect is a slowdown of the site, proportionate to the computational power lost. However, all services will continue.

Increased flexibility
: You can also manage different websites on the same static web server.

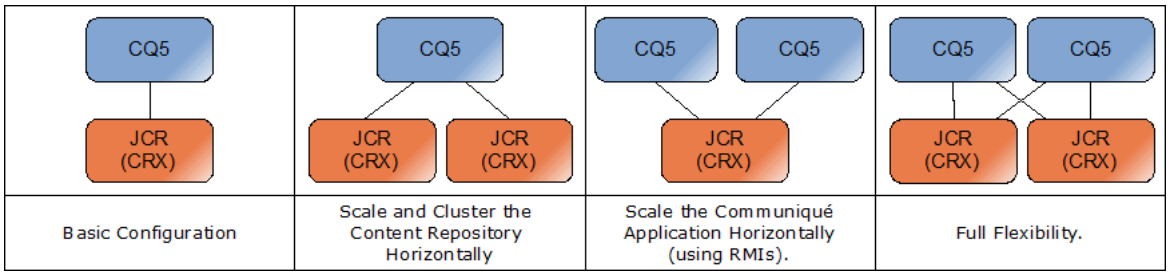
3.3.3 High availability CQ5

To increase availability, and performance, of your Production environment, it is common to combine multiple instances of the CQ WCM author and publish environments.

This concept can be extended to include multiple instances of your content repository. After you have installed a high-availability JCR solution, you can integrate it to ensure that your CQ WCM solution is protected against hardware and software failure. CRX Clusters are supported for various persistence managers. See the CRX Clustering documentation for further information.

There are various high-availability configurations possible (dependent on whether it is a publish or author environment) the basic principles of which include:

Figure 3.5. High-availability CQ WCM and JCR (CRX)



For information on setting up a high-availability **CRX** solution, please refer to the CRX Configuration Guide

4 Accessing CQ

4.1 Authentication and Authorization

4.1.1 Authentication

Authentication is the process of identifying, and verifying, a user.

The process of authentication and login can be broken down as follows:

1. Authentication information is extracted from the request. In CQ this is done by an *authentication handler*.
2. The authentication information is then checked to determine whether it is sufficient and/or correct. In CQ this is performed by the *login modules*.
3. The appropriate response is initiated.

For CQ, initial authentication uses a standard HTML-login form in conjunction with the *Authorization Header Authentication Handler*. The HTML-form must have fields for the user name and password (the same field names must then be used by the Authorization Header Authentication Handler).

You can also use a similar form for controlled access to various areas of your website.

4.1.1.1 LDAP, Single Sign On and Portals

The various authentication methods can be realized by using different login modules.

For example, CQ can interact with a LDAP server that stores user information centrally, eliminating the need for duplication. This central server is then used to verify login information which enables you to realize Single Sign On, both with other in-house applications and external Portals. See CRX - LDAP Authentication for further information.

4.1.2 Authorization

Authorization determines whether a user is allowed to take action on specific areas within the system. For example, a user can be authorized to read or update a specific page.

Authorization is managed using a series of entities:

User
: A user accesses a system using their user account. A user models either a human user or an external system connected to the system. The user account holds the details needed for accessing CQ; a key purpose of an account is to provide the information for the authentication and login processes - allowing a user to log in.

Groups
: A group is a collection of users and/or other groups. A change in the permissions/privileges assigned to a group is automatically applied to all users in that group. A user does not have to belong to any group, but often belongs to several.

Action
: Actions are performed on a resource. For example, a user can read, edit or delete a page, amongst other actions.

Permissions
: A permission allows a user to perform an action on a given resource within the repository. Permissions are stored, and can be seen, at resource level within the repository.

Privileges
: Privileges allow access to functionality available within the application; for example, replication of a specific path, or the ability to update the page hierarchy (including creating new pages).

Resources
: Resources define the functionality to be accessed.

4.2 CQ and the Web Accessibility Guidelines

CQ has been developed to maximize compliance with the Web Accessibility Guidelines.

Web accessibility means that “people with disabilities can perceive, understand, navigate, and interact with the Web, and that they can contribute to the Web”.

This can include measures such as providing textual alternatives to images (or any non-text item). These can then be used to help people with sight impairment by outputting the text on a braille keypad, or through a voice synthesizer. Such measures can also benefit people with slow internet connections, or indeed any internet user - when the measures offer the user more information.

Such mechanisms must be carefully planned and designed to ensure that they provide the information required for the user to successfully understand and use the content. For example, a text describing an image should not only state what the image is of, but the context in which it is being displayed - a picture of the Earth as seen from outer space can be used to illustrate many themes - space travel, global warming, the oceans, the continents and so on. .

There is a lot of information available about the guidelines on the Internet, including the Web Accessibility Initiative.

Certain aspects are integral to CQ, whereas other aspects must be realized during your project development.

Please see the Day US 508 Compliance Assessment for the official status with regard to Section 508 of the US government.

5 Data Modelling

5.1 Data Modeling - David Nuescheler's Model

5.1.1 Source

The following details are ideas and comments expressed by David Nuescheler.

David is co-founder and CTO of Day Software AG, a leading provider of global content management and content infrastructure software. He also leads the development of JSR-170, the Java Content Repository (JCR) application programming interface (API), the technology standard for content management.

Further updates can also be seen on http://wiki.apache.org/jackrabbit/DavidsModel.

5.1.2 Introduction from David

In various discussions I found that developers are somewhat at unease with the features and functionalities presented by JCR when it comes to content modeling. There is no guide and very little experience yet on how to model content in a repository and why one content model is better than the other.

While in the relational world the software industry has a lot of experience on how to model data, we are still at the early stages for the content repository space.

I would like to start filling this void by expressing my personal opinions on how content should be modeled, hoping that this could some day graduate into something more meaningful to the developers community, which is not just "my opinion" but something that is more generally applicable. So consider this my quickly evolving first stab at it.

Important

Disclaimer: These guidelines express my personal, sometimes controversial views. I am looking forward to debate these guidelines and refine them.

5.1.3 Seven Simple Rules

5.1.3.1 Rule #1: Data First, Structure Later. Maybe.

5.1.3.1.1 Explanation

I recommend not to worry about a declared data structure in an ERD sense. Initially.

Learn to love nt:unstructured (& friends) in development.

I think Stefano pretty much sums this one up.

My bottom-line: Structure is expensive and in many cases it is entirely unnecessary to explicitly declare structure to the underlying storage.

There is an implicit contract about structure that your application inherently uses. Let's say I store the modification date of a blog post in a `lastModified` property. My App will automatically know to read the modification date from that same property again, there is really no need to declare that explicitly.

Further data constraints like mandatory or type and value constraints should only be applied where required for data integrity reasons.

5.1.3.1.2 Example

The above example of using a "lastModified" Date property on for example "blog post" node, really does not mean that there is a need for a special nodetype. I would definitely use "nt:unstructured" for my blog post nodes at least initially. Since in my blogging application all I am going to do is to display the lastModified date anyway (possibly "order by" it) I barely care if it is a Date at all. Since I implicitly trust my blog-writing application to put a "date" there anyway, there really is no need to declare the presence of a "lastModified" date in the form a of nodetype.

5.1.3.1.3 Discussion

http://www.nabble.com/DM-Rule-#1:-Data-First,-Structure-Later.-Maybe.-tf4039967.html

5.1.3.2 Rule #2: Drive the content hierarchy, don't let it happen.

5.1.3.2.1 Explanation

The content hierarchy is a very valuable asset. So don't just let it happen, design it. If you don't have a "good", human-readable name for a node, that's probably that you should reconsider. Arbitrary numbers are hardly ever a "good name".

While it may be extremely easy to quickly put an existing relational model into a hierarchical model, one should put some thought in that process.

In my experience if one thinks of access control and containment usually good drivers for the content hierarchy. Think of it as if it was your file system. Maybe even use files and folders to model it on your local disk.

Personally I prefer hierarchy conventions over the nodetyping system in a lot of cases initially, and introduce the typing later.

5.1.3.2.2 Example

I would model a simple blogging system as follows. Please note that initially I don't even care about the respective nodetypes that I use at this point.

```
/content/myblog
/content/myblog/posts
/content/myblog/posts/what_i_learned_today
/content/myblog/posts/iphone_shipping

/content/myblog/comments/iphone_shipping/i_like_it_too
/content/myblog/comments/iphone_shipping/i_like_it_too/i_hate_it
```

I think one of the things that become apparent is that we all understand the structure of the content based on the example without any further explanations.

What may be unexpected initially is why I wouldn't store the "comments" with the "post", which is due to access control which I would like to be applied in a reasonably hierarchical way.

Using the above content model I can easily allow the "anonymous" user to "create" comments, but keep the anonymous user on a read-only basis for the rest of the workspace.

5.1.3.2.3 Discussion

http://www.nabble.com/DM-Rule-#2:-Drive-the-content-hierarchy,-don't-let-it-happen.-tf4039994.html

5.1.3.3 Rule #3: Workspaces are for clone(), merge() and update().

5.1.3.3.1 Explanation

If you don't use clone(), merge() or update() methods in your application a single workspace is probably the way to go.

"Corresponding nodes" is a concept defined in the JCR spec. Essentially, it boils down to nodes that represent the same content, in different so-called workspaces.

JCR introduces the very abstract concept of Workspaces which leaves a lot of developers unclear on what to do with them. I would like to propose to put your use of workspaces to the following to test.

If you have a considerable overlap of "corresponding" nodes (essentially the nodes with the same UUID) in multiple workspaces you probably put workspaces to good use.

If there is no overlap of nodes with the same UUID you are probably abusing workspaces.

Workspaces should not be used for access control. Visibility of content for a particular group of users is not a good argument to separate things into different workspaces. JCR features "Access Control" in the content repository to provide for that.

Workspaces are the boundary for references and query.

5.1.3.3.2 Example

Use workspaces for things like:

- v1.2 of your project vs. a v1.3 of your project
- a "development", "QA" and a "published" state of content

Do not use workspaces for things like:

- user home directories
- distinct content for different target audiences like public, private, local, ...
- mail-inboxes for different users

5.1.3.3.3 Discussion

http://www.nabble.com/DM-Rule-#3:-Workspaces-are-for-corresponding-nodes.-tf4040010.html

5.1.3.4 Rule #4: Beware of Same Name Siblings.

5.1.3.4.1 Explanation

While Same Name Siblings (SNS) have been introduced into the spec to allow compatibility with data structures that are designed for and expressed through XML and therefore are extremely valuable to JCR, SNS come with a substantial overhead and complexity for the repository.

Any path into the content repository that contains an SNS in one of its path segments becomes much less stable, if an SNS is removed or reordered, it has an impact on the paths of all the other SNS and their children.

For import of XML or interaction with existing XML SNS maybe necessary and useful but I have never used SNS, and never will in my "green field" data models.

5.1.3.4.2 Example

Use

```
/content/myblog/posts/what_i_learned_today
/content/myblog/posts/iphone_shipping
```

instead of

```
/content/blog[1]/post[1]
/content/blog[1]/post[2]
```

5.1.3.4.3 Discussion

http://www.nabble.com/DM-Rule-#4:-Beware-of-Same-Name-Siblings.-tf4040024.html

5.1.3.5 Rule #5: References considered harmful.

5.1.3.5.1 Explanation

References imply referential integrity. I find it important to understand that references do not just add additional cost for the repository managing the referential integrity, but they also are costly from a content flexibility perspective.

Personally I make sure I only ever use references when I really cannot deal with a dangling reference and otherwise use a path, a name or a string UUID to refer to another node.

5.1.3.5.2 Example

Let's assume I allow "references" from a document (a) to another document (b). If I model this relation using reference properties this means that the two documents are linked on a repository level. I cannot export/import document (a) individually, since the reference property's target may not exist. Other operations like merge, update, restore or clone are affected as well.

So I would either model those references as "weak-references" (in JCR v1.0 his essentially boils down to string properties that contain the uuid of the target node) or simply use a path. Sometimes the path is more meaningful to begin with.

I think there are use cases where a system really can't work if a reference is dangling, but I just can't come up with a good "real" yet simple example from my direct experience.

5.1.3.5.3 Discussion

http://www.nabble.com/DM-Rule-#5:-References-considered-harmful.-tf4040042.html

5.1.3.6 Rule #6: Files are Files are Files.

5.1.3.6.1 Explanation

If a content model exposes something that even remotely *smells* like a file or a folder I try to use (or extend from) nt:file, nt:folder and nt:resource.

In my experience a lot of generic applications allow interaction with nt:folder and nt:files implicitly and know how to handle and display those event if they are enriched with additional meta-information. For example a direct interaction with file server implementations like CIFS or WebDAV sitting on top of JCR become implicit.

I think as good rule of thumb one could use the following: If you need to store the filename and the mime-type then nt:file/nt:resource is a very good match. If you could have multiple "files" an nt:folder is a good place to store them.

If you need to add meta information for your resource, let's say an "author" or a "description" property, extend nt:resource not the nt:file. I rarely extend nt:file and frequently extend nt:resource.

5.1.3.6.2 Example

Let's assume that someone would like to upload an image to a blog entry at:

```
/content/myblog/posts/iphone_shipping
```

and maybe the initial gut reaction would be to add a binary property containing the picture.

While there certainly are good use cases to use just a binary property (let's say the name is irrelevant and the mime-type is implicit) in this case I would recommend the following structure for my blog example.

```
/content/myblog/posts/iphone_shipping/attachments [nt:folder]
/content/myblog/posts/iphone_shipping/attachments/front.jpg [nt:file]
/content/myblog/posts/iphone_shipping/attachments/front.jpg/jcr:content [nt:resource]
```

5.1.3.6.3 Discussion

http://www.nabble.com/DM-Rule-#6:-Files-are-Files-are-Files.-tf4040063.html

5.1.3.7 Rule #7: IDs are evil.

5.1.3.7.1 Explanation

In relational databases IDs are a necessary means to express relations, so people tend to use them in content models as well. Mostly for the wrong reasons through.

If your content model is full of properties that end in "Id" you probably are not leveraging the hierarchy properly.

It is true that some nodes need a stable identification throughout their live cycle. Much fewer than you might think though. mix:referenceable provides such a mechanism built into the repository, so there really is no need to come up with an additional means of identifying a node in a stable fashion.

Keep also in mind that items can be identified by path, and as much as "symlinks" make way more sense for most users than hardlinks in a unix filesystem, a path makes a sense for most applications to refer to a target node.

More importantly, it is **mix**:referenceable which means that it can be applied to a node at the point in time when you actually need to reference it.

So let's say just because you would like to be able to potentially reference a node of type "Document" does not mean that your "Document" nodetype has to extend from mix:referenceable in a static fashion since it can be added to any instance of the "Document" dynamically.

5.1.3.7.2 Example

use:

```
/content/myblog/posts/iphone_shipping/attachments/front.jpg
```

instead of:

```
[Blog]
- blogId
- author

[Post]
- postId
- blogId
- title
- text
- date

[Attachment]
- attachmentId
- postId
- filename
```

```
+ resource (nt:resource)
```

5.1.3.7.3 Discussion

http://www.nabble.com/DM-Rule-#7:-IDs-are-evil.-tf4040076.html

Appendix A. Copyright, Licenses and Formatting Conventions

For all copyright statements and license agreements see Copyright, Licenses and Disclaimers.

A.1 Formatting Conventions

The following tables detail formatting conventions used within this guide:

Table A.1. Formatting Conventions - Text

Style	Description	Example
Cross-reference	Cross-reference to external documents.	See the *Microsoft Manual of Style for Technical Publications*.
GUI Item	User interface items.	Click **Save**.
Keyboard shortcut	Keyboard shortcuts.	Press **Ctrl+A**.
Mouse Button	Mouse buttons.	**Secondary-mouse** button (usually the **right-mouse** button).
Link	Link to anchor-points within the current document and/or external sources.	http://www.day.com
`Code`	Example of programming code.	`if (weather == sunny) smile;`
User Input	Example of text, or commands, that you type.	**ls *.xml**
<Variable User Input>	Example of variable text - you type the actual value needed.	`ls <cq-installation-dir>`
[Optional Parameter]	An optional parameter.	`ls [<option>] [<filename>]`
`Computer Output`	Logging and error messages.	`ls: cannot access error.log:`

Table A.2. Formatting Conventions - Actions

When you see this...	It means do this...
Ctrl+A	Hold down the **Ctrl** key, then press the **A** key.
Right-click	Press the **right-mouse** button (or the **left-mouse** button if your mouse has been configured for left-handed use).
Drag	Hold down the left mouse button while moving the item, then release the mouse button at the new location (or the right mouse button if your mouse has been configured for left-handed use).