DOCUMENTATION

Technical Overview

Jahia v6.61

Jahia's next-generation, open source CMS stems from a widely acknowledged vision of enterprise application convergence – web, document, search, social and portal – unified by the simplicity of web content management.

Jahia Solutions Group SA

9 route des Jeunes,
CH-1227 Les acacias
Genève, Suisse

http://*www.jahia.com*

Summary

1 Introduction 4
2 Overview 5
2.1 What is Jahia? 5
2.2 The closer view 5
2.3 Technical requirements 6
2.4 “Everything is content” 6
2.4.1 Developer and integrator customization 7
2.5 Integrated technologies 7
2.6 Architecture overview 9
2.7 Modules 10
2.8 Jahia’s different actors 11
3 Web layer 14
3.1 Content flow 14
3.2 Templates and views 16
3.3 The Jahia Template Studio 17
3.3.1 Page templates 17
3.3.2 Content templates 17
3.4 REST API 18
3.4.1 Actions 19
3.4.2 Example : native iPhone/iPad application 20
3.5 Mobile rendering 21
3.6 Macros 22
3.7 Filters 22
4 Back-end layer 24
4.1 Workflow 24
4.2 JBoss Drools and event listeners 24
4.3 File repository 25
4.4 Searching and indexing 25

4.4.1 Full text queries using search tag libraries 26
4.4.2 Query languages 26
4.5 Authentication and authorization 27
4.5.1 Single sign-on 28
4.5.2 Roles and permissions 29
4.6 Import / export 29
4.7 Distant publication 29
4.8 Portlets 30
4.8.1 Portlet versus modules 30
5 Performance 33
5.1 Caches 33
5.1.1 Cache types 33
5.1.2 The browser cache layer 34
5.1.3 The front-end HTML cache layer 34
5.1.4 Object cache layer 34
5.1.5 Database caches 35
5.2 Clustering 35
5.2.1 Visitors nodes 36
5.2.2 Authoring nodes 36
5.2.3 Processing node 36
5.3 More resources on performance 36
6 Additional resources 37

1 Introduction

This document contains a technical introduction to Jahia. It is designed to help readers with technical skills such as integrators, developers, testers or others as a starting point into Jahia. It is not meant to be a user's guide or an administration's guide. Please refer to the corresponding documents if that is what you are looking for.

This document has six sections:

- An overview of Jahia: what it is, the different types of actors involved, technical requirements, integrated technologies and frameworks.
- The web layer, which is a description of the layer exposed to the browser, and how it relates to the various components in Jahia; how they may be composed to build powerful web applications.
- The back-end layer, which contains a description of all the different services and technologies available in Jahia. This back-end is used by the web layer but may also be in some cases used directly by integrators, such as in the case of integrating custom workflows.
- A section about performance, and how Jahia addresses the very demanding high-load scenarios.
- Finally, a section that describes the additional resources available to developers and integrators, from online resources to commercial support contracts.

2 Overview

This section presents a global overview of the elements of a Jahia system.

2.1 What is Jahia?

Jahia can be many things to many people. Most projects will use it as a Web Content Management (WCM) solution, or whatever the moniker is at the time of reading, while others will use it as a portal server, a web integration platform, or even a full-fledged content integration solution.

What Jahia really is, is software listening to HTTP requests with the ability to produce responses using HTML, any markup or even binary data that users might need. In its core, it uses a content repository to store and retrieve content, as well as multiple ways of deploying custom logic to operate on the content or to interface with third party systems. That's the million-mile view of Jahia, and should give you a good idea of the flexibility of the system.

2.2 The closer view

If the previous description of Jahia was a bit too abstract, the following description should be more helpful. Jahia is basically composed of the following layers:

- A servlet container (Apache Tomcat, Oracle WebLogic, IBM WebSphere or others)
- A set of filters and servlets that form the "outer layer" of Jahia
- A set of Spring beans that make up the main architecture of Jahia
- A set of modules that extend the basic functionality
- A JCR implementation for storage of content (Apache Jackrabbit 2.x)
- A portal container (Apache Pluto 2.x)
- A scheduler (Quartz)
- A workflow engine (jBPM)
- A rules engine (Drools)

Of course this is a much-simplified view of the elements that Jahia is made of, but it should help identify the type of technologies involved.

2.3 *Technical requirements*

Jahia has the following minimum requirements:

- Oracle JDK 1.5 or more recent 100% compatible 32-bit or 64-bit JDKs
- A servlet API 2.4 / JSP 2.0 container
- 2GB RAM
- Windows, Linux (RedHat, Ubuntu), Mac OS X operating systems

Recommended requirements:

- Oracle 64-bit JDK 6 or above
- Apache Tomcat 6.x
- 4GB RAM
- Ubuntu or RedHat Linux 64-bit kernel

2.4 *"Everything is content"*

Another way of presenting Jahia is what we call "everything is content". Since the very beginning, Jahia has been aggregating all kinds of content on pages, including dynamic content such as portlets. Jahia has always been able to mix applications and content on the same web pages. Jahia 6.5/6.6 takes this notion even further by easily allow the building of "content-based applications", also known as "composite applications" that make it easy to build powerful applications that share a content store as a back-end.

In other words, working with Jahia means manipulating content and defining views as well as rules to be executed when an event is triggered on the content. Any content element stored in Jahia (text, image, PDF documents, portlet references, OpenSocial or Google gadgets) is considered content and therefore shares:

- Common properties (name, UUID, metadata, etc.)
- Common services (editing UI, permissions, versions, etc.)
- Common rendering and handling systems

Content is stored in a hierarchical structure (using Java Content Repository aka JCR standard), but as you will see it is possible to query or operate on it in other ways.

2.4.1 *Developer and integrator customization*

End users may see Jahia as a product, but for developers and integrators it is also a very powerful platform that may be configured and extended to fit a wide variety of needs.

Here is a sample of the different type of customization tasks:

- Integration and personalization
    - Of templates
    - Of default Jahia modules
- Development
    - New modules usable in pages
    - New logic parts (rules, filters, actions, classes)
    - New functionalities that add features to Jahia
- Configuration
    - Of workflows
    - Of roles and permissions
    - Of the user interface

2.5 *Integrated technologies*

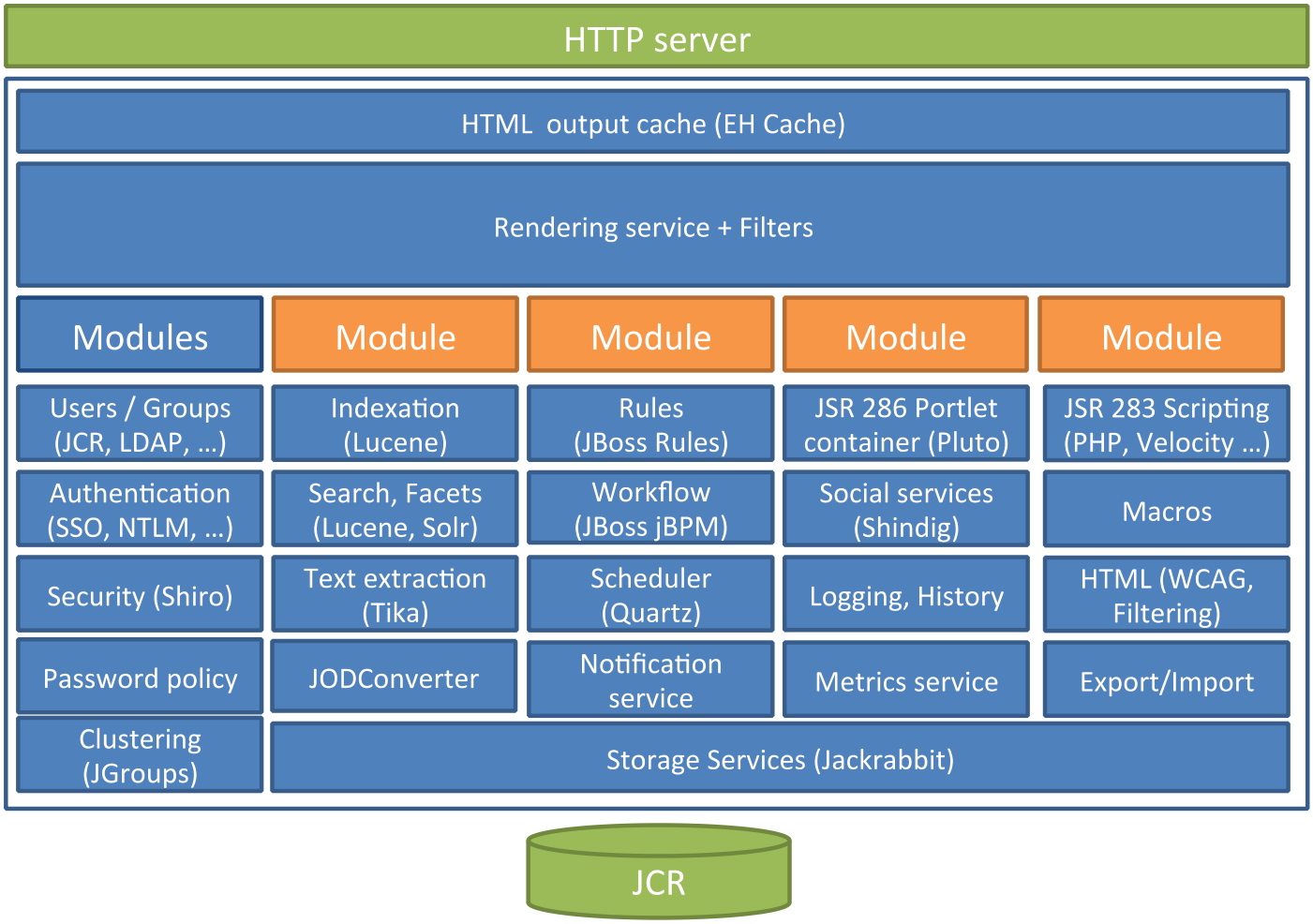
Jahia integrates a lot of different technologies and frameworks, and this section will give you an overview of what's included and how it is used.

- Jahia stores all of its data in a Java Content Repository (JCR) (Apache Jackrabbit 2.x):
    - Two workspaces are used in the JCR, one for the staging content (called "default") and one for the live content (called "live")
    - JCR content is stored in an SQL database (MySQL, PostgreSQL, Oracle, MSSQL, and more). Node data is stored in serialized form for performance reasons
- Jahia integrates in its core:
    - The core (services and frameworks) containing:
        - Apache Lucene as the indexing/search engine
        - Apache Camel as an enterprise integration engine

- Spring Framework as the dependency injection and bean configuration technology (as well as much more)
- Google Web Toolkit with Sencha GXT extensions for the UI in Edit mode and Studio mode
- JQuery and extensions for the contribute and live modes
- JBoss Drools as a rule engine
- JBoss BPM as a workflow engine

- Modules (extensions and templates) that contain:
  - XWiki as the wiki engine
  - Apache Shindig (OpenSocial implementation)
  - LDAP connectors
  - Search Engine Optimization (SEO)
  - Tags and tag clouds

2.6 *Architecture overview*



As you can see, the top layers are basic rendering and communication layers, while the underlying services are more modular. The blue boxes covered what is offered in the core services, either as core modules or framework, while the orange boxes show that modules may not only provide more content definitions, but also custom logic and much more.

Jahia 6.5 introduced modules to the architecture. Prior to this version, extensions to Jahia would be integrated through Spring beans being deployed, but no specific packaging was possible. Since 6.5, it is possible to package modules in WAR files that can then be deployed to extend or complement Jahia functionality. Actually a lot of default functionality in Jahia is built using modules, including, for example, the contribute mode or template sets.

2.7 *Modules*

Modules are a very important part of Jahia, and may be thought of as Jahia's plug-in infrastructure. Basically they are composed of directory and files that are packaged as a WAR file and then copied into Jahia's WEB-INF/var/shared_modules directory for deployment. Upon detection of the new file, Jahia will then deploy the contents into the modules/ directory. Modules may range from very simple ones, such as only defining new views for existing content types, to very complex implementation of new communication layers such as OpenSocial, or implementing back-end LDAP user and group providers.

Template sets (see the Template Studio section below) are also packaged as modules, which make them easy to deploy and update.

Advantages of modules include:

- Re-usability: as they are autonomous, it is easy to move them from development to staging or to production environments. It is also easy to re-use them across projects or share them with others. Also, as it is possible to inherit from existing modules, it makes it nice and easy to extend a default module.
- Maintenance: as they are autonomous blocks, they can focus on a specific use case (like in the case of a forum module), which makes maintenance and evolution easy.
- Reliability: if a module fails, only that part of the system will be unavailable, the rest of the platform will continue to serve requests.
- Separation of concern: as the modules may be integrated at a later time, this makes it easier to split work among members of a team.

A developer will therefore mostly work on modules, either creating new ones or extending the out-of-the-box ones. He may also share his work (or re-use others' contributions) on Jahia's forge (http://www.jahia.org/forge).

A module may contain:

- Content definitions
- View scripts (JSP, JSR-286 compatible languages such as Velocity or Freemarker, or even PHP*)
- Static resources (text file, images, CSS files, Javascript files)

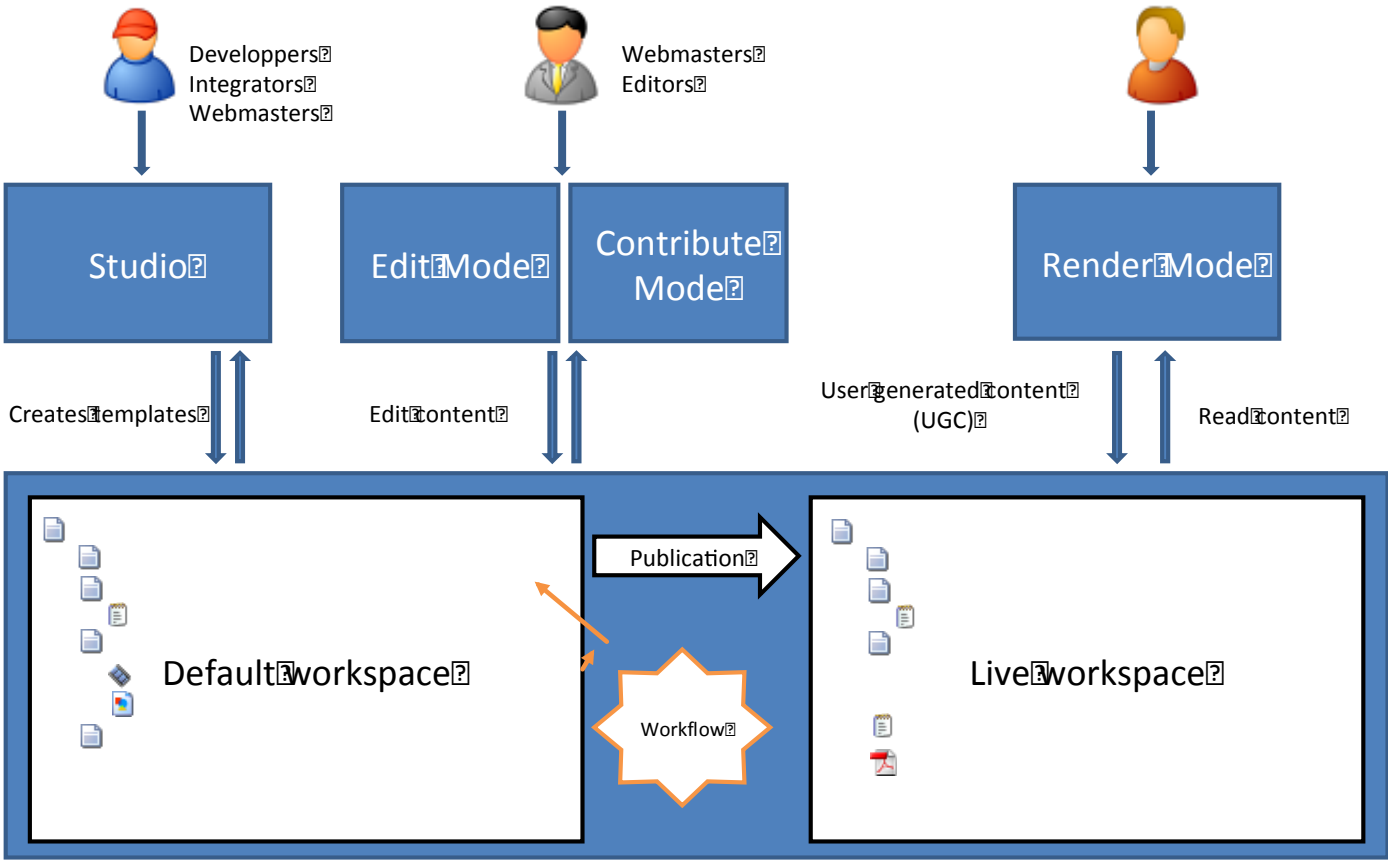
- Resource bundles or other property files
- Java classes or JAR libraries
- Filters
- Permission and role definitions
- Rules
- jBPM workflow definitions
- Tag libraries
- Spring Framework configuration files
- Content import files (in XML format)

Note that none of these files are required, and you may create an empty module, although it won't be very useful.

** Through the integration of Caucho's Quercus PHP engine, which may require a commercial license depending on deployment needs.*

2.8 *Jahia's different actors*

In this section we will present the different types of actors that may interact with a Jahia system, and how they relate to different activities.



Developers, integrators and webmasters will mostly interact with the studio as well as modules to create templates, modules so that the other users may use a system that has been customized to their needs. In this role, this will enable them to “lock” the look and feel of the web site, as well as the content definitions, rules or any other custom logic needed.

Webmasters and/or editors will then use the output of the previous work to populate the site with content, using the edit mode and/or the contribute mode. The Edit Mode is a very powerful content editing interface, mostly targeted at advanced users, while the Contribute Mode is an easy-to-use content editing interface aimed at basic content editors. It should also be noted that integrators are free to customize the Contribute Mode to their requirements, in order to tailor the experience for the editors. Once the editors are happy with the content, they may use the workflow to publish the modifications to the live workspace (or if they are not directly allowed to do so, they may start the review process), at which point it will be available to site visitors.

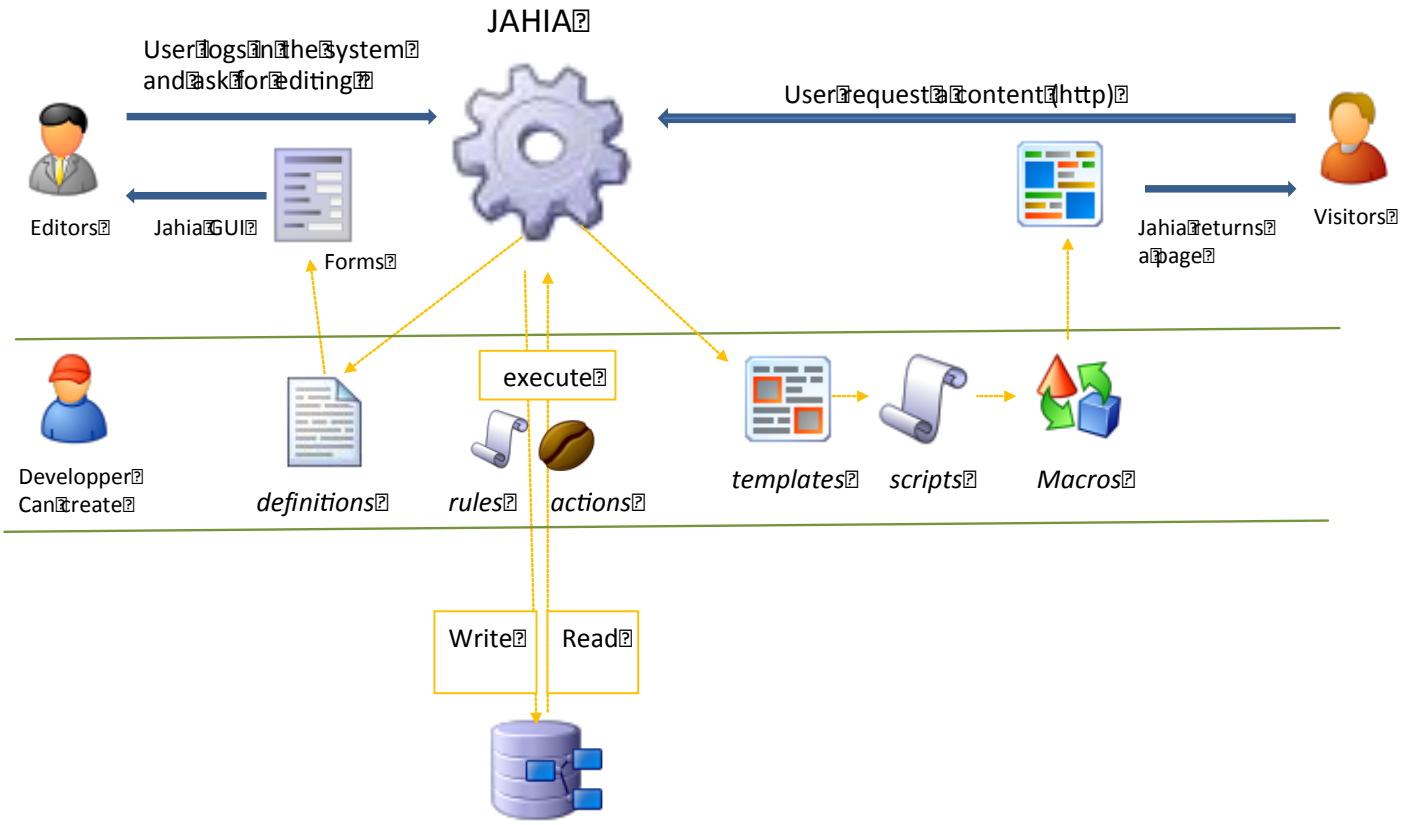
Site visitors may then browse the site, and if allowed, also input user-generated content in modules such as the forum, wiki or another other components deployed in the site.

3 Web layer

This section details the web layer of a Jahia system. This layer is both flexible and powerful, so we will first go over the flow of content, then present how a page is rendered.

3.1 Content flow

In order to understand how Jahia works with content, we have illustrated this with the following diagram:



Starting from the bottom up, the developer can create different types of objects, ranging from content definitions to macros that will be used by Jahia to customize the experience for other users. We will now briefly detail the different types of objects:

- Definitions: content definitions define the type of objects that will be edited in the system as well as their structure. These may range from simple properties to complex sub-tree structures
- Rules: rules define “consequences” (similar to actions) that must be performed when a certain condition is met. They make it possible, for example, to listen to modifications on content objects (such as page creation), to trigger any type of consequence.

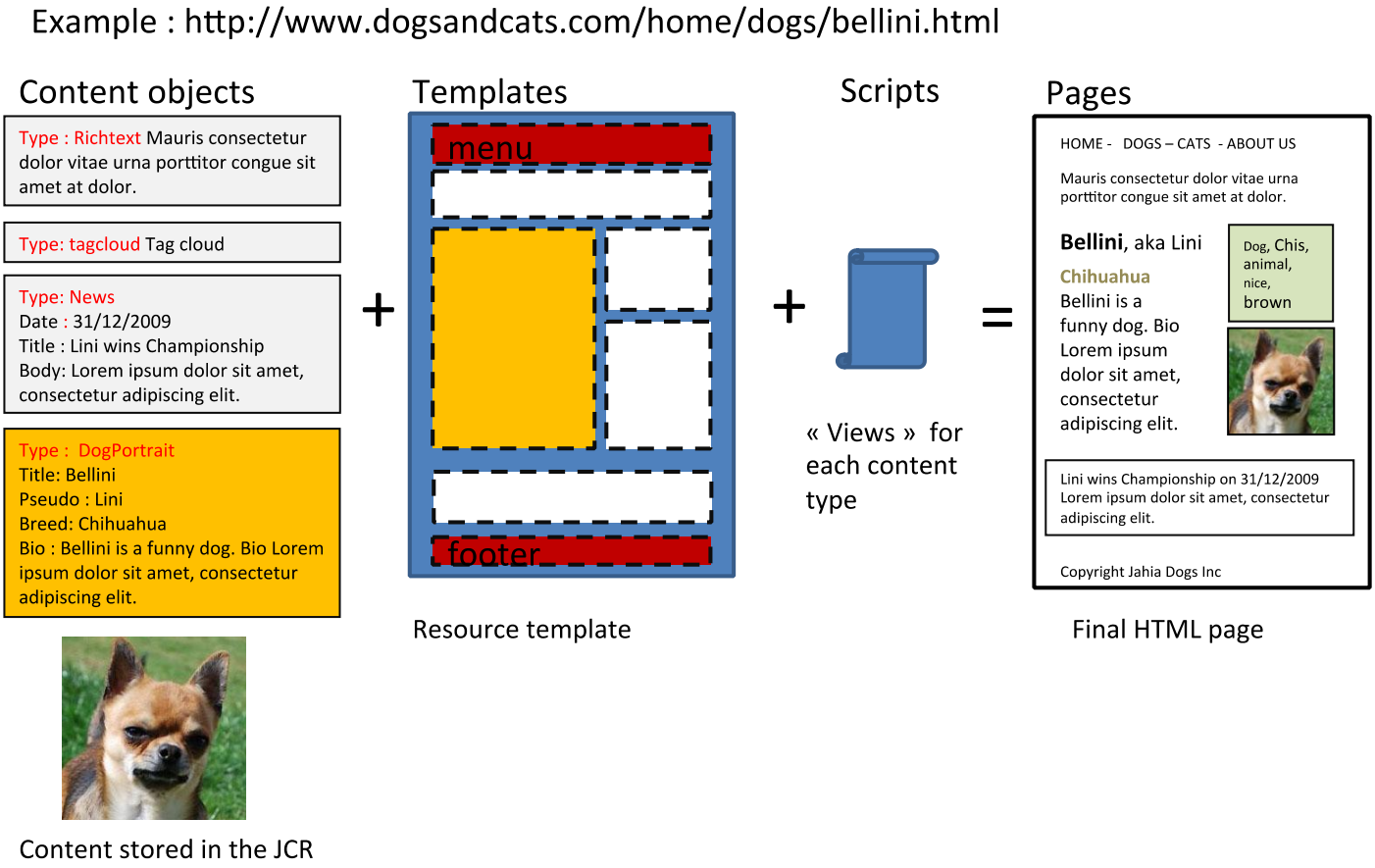
- Actions: actions are similar to method calls, except that they are called from the REST API. Developer may either use existing actions (such as “createUser” or “startWorkflow”) or define their own custom ones to fit their needs. This simple yet powerful extension mechanism makes it possible to perform almost any task in the Jahia back-end from a REST call.
- Templates: templates are defined in the Jahia Template Studio, and they make it easy to define page and content layouts that may be used when creating pages or displaying content elements (such as a news entry). Templates may be packaged in Template Sets, and then be deployed to any web site, or moved from staging to production environments. Template Sets may even contain default content, which is useful to create powerful site factory scenarios.
- Scripts: used to render a specific content object type. The default script type is JSP, but Jahia supports any Java Scripting API (http://www.jcp.org/en/jsr/detail?id=223) compatible script language (for example Velocity, Freemarker, or even PHP). Multiple scripts may be provided for a single node type: these are called “views” in Jahia.
- Macros: macros may also be defined to provide quick substitutions on the final output of a Jahia page. Macros are executed even if a page is retrieved from the HTML cache, so macros can be very useful to quickly customize page output. There are some performance caveats as macros are constantly executed; they must always be very fast to execute.

Editors will then log into the system and start creating sites, pages and other content types that were specified by the developers. They use Jahia’s powerful Edit Mode or the simpler Contribute Mode to submit content and build the site bit by bit. As they enter content, rules, actions are used to perform logic actions upon submission; and then templates, scripts and finally macros are used to output the resulting HTML.

Visitors will either surf anonymously or log into the system, browse the site and interact with any dynamic object types that the editors and developers have made available to them. An example of such a dynamic object could be a forum, or a comment list. The content they contribute is called “User Generated Content” (or UGC). Again, Jahia will use the templates, scripts and macros to render the pages for the visitors, and if they are allowed to enter content, rules, actions and content definitions will again come into play (but not illustrated above to keep the diagram simple).

3.2 *Templates and views*

As presented in the previous section, Jahia 6.5 introduced a new editable template system that makes it easy to customize basic or even complex layouts without any scripting skills. In order to better understand how a page is composed, we will now illustrate this in the following schema:



In the example presented above, we are requesting a content object from the Java Content Repository that is located on the home page called “bellini”. Jahia will therefore first use the URL to find the corresponding content object, and then start looking for different objects that will help render the final page. In this specific example, we are not requesting a page, but a content object directly, which is a little more complex. If we had wanted to render the page, we would have used the following URL: http://www.dogsandcats.com/home.html . Jahia would have then looked for a page template, scanned it to find all the different objects present on the page, and then used views to render each specific object type. In the case of the above example, we have illustrated a more advanced use case, where we directly request a content object.

If all we had for a content object was a view script, when requesting an object by itself, we would probably only get an HTML fragment rendered instead of a complete page (as object views are designed to be re-used within pages). In order to avoid this, Jahia has a mechanism called the “content template” that allows integrators to design a template specific to a content object type that will be used to “decorate” around the object itself, allowing, for example, to render navigation, headers and footers around the object. The rendering of a full HTML page for a single content object then becomes quite similar to the case of rendering an actual content page.

3.3 *The Jahia Template Studio*

As templates are not scripts, but defined inside the content repository, Jahia 6.5 introduced a new tool called the Template Studio to edit them. A template is actually a set of nodes that will define the layout of a page, allowing users with no scripting or HTML experience to easily edit or update existing templates. For advanced users with strong JCR skills, you could even export the template as XML, edit it and re-import it back into Jahia, should you wish to do so. Templates are regrouped in Template Sets, which can then be deployed to a site on the same Jahia installation or packaged as a module and exported as a WAR file to either be deployed on another development instance, or to another Jahia installation for staging or production.

3.3.1 *Page templates*

Page templates are the default template type, and are made available to editors when they create a new page. At that point the editor may specify the template he wishes to use that will define the layout of the page. Building structured templates targeted to the specific site vertical makes it a lot easier for site administrators to make sure that the sites have a coherent structure and look, and will also help make changes later on.

Page templates (and content templates) may also inherit from a parent template, so you may, for example, have a “base” template that has a very free structure, and then inherit from it to build templates that have more rigid structures.

3.3.2 *Content templates*

As explained in the content flow diagram, content templates are used when an URL is requesting a content object that is not a page type, but any other. It is therefore possible to “decorate” a content type by adding

navigation, headers, footers, or any other desired page elements around a rendered content object by defining a template and associating it with a list of types (through the Jahia Template Studio UI) for which it should be used. This is very useful for master/detail views, where the master list of objects would be displayed on a page, and the detail view could be a single content object rendered using a content template.

For example, let's say you have news articles in your definitions, and you would like to display a single news article on a HTML page. You could have a home page that would list the abstract of the last ten news articles, and each of them would have a link to a single news article detail view, with all the details and sub objects attached to it. The home page would be rendered using the page template and the news article detail would be rendered using a content template associated with the news article type.

3.4 *REST API*

Jahia provides a simple yet powerful REST API that allows CRUD (Create, Retrieve, Update and Delete) operations on content. It also offers more advanced operations such as searches, triggering actions (built-in or custom), as well as authentication.

Jahia also uses this API in various modules (such as the entire contribute mode) to implement their functionality, using AJAX technologies such as JQuery.

Access to a content element in the REST API is done through a URL convention that we detail below:

- http://(servername:serverport)/(context)/(servlet)/(workspace)/(lang)/(path).(view).(templateType)
- Example: http://www.domaine.com/jahia/cms/render/defaut/en/sites/ACME/home.html

Where:

- servername: is the hostname that is associated with a site, or simply the server's hostname or IP address if no site is associated with a hostname.
- serverport: is an optional port number if Tomcat is not configured to listen to port 80, or if no Apache server has been setup in front of Tomcat.
- context: is the context in which Jahia is deployed (empty if Jahia is deployed as the ROOT application).

- servlet : the servlet you want to use. Most of the time it will be the "/cms/render" servlet that is the main page render servlet, but you might also need to use "/cms/login", "/cms/logout" or "/cms/find".
- workspace: live or default (=staging).
- lang: is the language code for the language you wish to render or modify.
- path: the path to a node (such as /sites/ACME/home.html in the above example).
- view: an optional view.
- templateType: the type of render desired. Most of the time it will be .html, but you can just as easily request .csv, .xml or .rss. Of course, this requires the corresponding views to have been defined in the system.

We also provide an URL rewriting system as well as a vanity URL system for SEO compatibility. The above explanation is really targeted towards integrators and developers that want to use the REST API.

Now that we have detailed the URL format, let's look at the possible methods that may be called on an URL. The REST API offers basic CRUD (Create, Retrieve, Update and Delete) methods that are mapped to the HTTP standard methods:

- GET will render the node.
- POST will create a new node (or call action).
- PUT will update the node.
- DELETE will remove the node.

It is possible to have more methods using actions.

3.4.1 *Actions*

Actions are an extension point to the default HTTP methods provided in the REST API. Methods may be defined in modules and make it easy to perform additional operations on nodes.

Here is an example of an action being called from an URL:

```
POST
http://localhost:8080/cms/render/default/en/sites/ACME/home.startWorkflow.do
```

All actions use the *.do extension at the end. The above example will start a workflow process on the /sites/ACME/home node. Additional parameters may be specified either using the URL query string, although it is recommended to use HTTP body parameters.

Here are some additional examples of actions provided by Jahia:

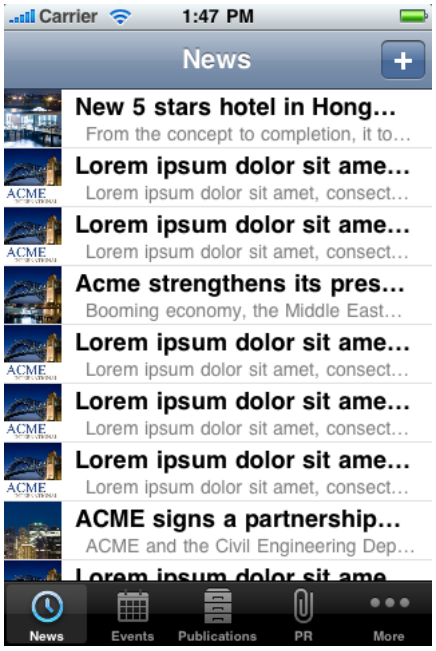
- NewUser (module: userregistration)
- PublishFile (module: docspace)
- MailAction (module: formbuilder)
- GetFeed (module: feedimporter)
- DocumentConverterAction (module: docConverter)

As you can imagine, this makes it easy to use content actions directly from a simple HTML form, without any Javascript (in this example calling the newUser action):

```
   <form method="post" action="<c:url
value='${currentNode.path}.newUser.do' context='${url.base}'/>"
name="newUser" id="newUser">
```

3.4.2 *Example : native iPhone/iPad application*

An interesting example application illustrating the usefulness of the REST API is a prototype of a native iPhone/iPad application that was designed by Jahia. The native application connects to Jahia on startup, retrieves a configuration file that contains different screen properties, notably queries that will be executed with the REST find servlet. The results are generated using JSON output. The native application can then display the content any way it chooses. Having the content in JSON format makes it easy to adapt to multiple screen sizes, as the native prototype is a dual iPhone and iPad application. The application is also capable of creating new content by using POST HTTP requests to create new content objects. All this is quite optimal and integrated with the authentication as it also uses the login and logout servlets.



As you can see in the above example, the native application displays the ACME demo content, with native device rendering. This makes for a strong mobile user experience while having exactly the same content base. It is also possible to cache the data in the native client for offline browsing.

3.5 *Mobile rendering*

An alternative to the native application rendering is to use user agent matching to change the template type so that the rendering may adapt the size and navigation to the size of mobile devices. Jahia makes it possible to configure the user agent matching using regular expression matching and changing the template type dynamically so it becomes possible to switch to alternate scripts to render the content. For example, if we have the following configuration in the WEB-INF/etc/spring/application-renderer.xml file:

```
<bean class="org.jahia.services.render.filter.UserAgentFilter">
     <property name="priority" value="6" />
     <property name="applyOnModes" value="live,preview" />
     <property name="applyOnConfigurations" value="page" />
     <property name="userAgentMatchingRules">
          <map>
               <entry key=".*iPhone.*" value="iphone" />
               <entry key=".*iPod.*" value="iphone" />
              <entry key=".*iPad.*" value="iphone" />
              <entry key=".*Android.*" value="iphone" />
          </map>
     </property>
</bean>
```

This means that if an iPhone or Android user agent is detected, it will first look for an “html-iphone” script directory for a view script, and if it does not exist, it will default to an “html” directory. As you can see this simple yet powerful mechanism makes it possible to do all kinds of user agent matching. You could even use it to render for specific desktop user agents (such as legacy ones).

It is also possible to use integrate with solutions such as the Apache Mobile Filter (http://www.idelfuschini.it/it/apache-mobile-filter-v2x.html) which can expose mobile device capabilities as request attributes if you need more precise control over page rendering. This might also be a good candidate for a filter, or you could integrate WURFL (http://wurfl.sourceforge.net/) as a module.

3.6 *Macros*

As described in the content flow, Jahia also has a macro mechanism, which makes it possible to insert markers in pages (even in free text field) that will be dynamically replaced with the corresponding values. This is useful if you want to use a macro to insert a user’s name, or anything you might think of. Macros may be defined in modules as JSR-223 script files responding to a specific syntax (such as {getConstant} or {username}), and Jahia also offers a few default macros such as:

- Constants stored in a node property display
- Form token generation, in order to prevent multiple submissions of the same form
- Display the current user name

3.7 *Filters*

Macros actually require a marker to be inserted in the page for the content to be inserted, so there may be some cases where you actually want to filter the output and perform transformations in real time. This can be done through the usage of filters. For example, it might be useful to use a filter to transform all email addresses detected and replace them with an obfuscated version that will avoid detection by spam web spiders. Jahia provides different filters out of the box, but you may of course add your own. Here is a non exhaustive list:

- Portlet process action filter (for portlet support)
- Static asset filter (injects Javascript and CSS in the HTML page header)

- Metrics logging filter (gathers statistics on object views)
- User agent dispatcher (for mobile rendering)
- HTML cache
- Email obfuscator
- And a lot more…

As you can see, some filters are quite powerful, while others may be very specific to a certain usage. Filters are an important part of the internal processing of Jahia.

4 Back-end layer

This section presents all the various technologies and frameworks available in Jahia's back-end layer.

4.1 Workflow

Jahia integrates the jBPM 4 workflow engine (http://www.jboss.org/jbpm) that provides support for advanced workflow integrations through the definition of workflows, using the BPMN 2.0 specification. Jahia's UI is integrated with the workflow screens so that the experience is seamless for end-users who will never have to leave the UI to perform workflow operations (unless integrators wish to, of course). It is also compatible with any jBPM compatible GUI workflow definition tools such as the Signavio web modeler tool (http://docs.jboss.com/jbpm/v4/devguide/html_single/#signavio)

4.2 JBoss Drools and event listeners

Often, as modifications of content happen, integrators will face the need to perform a specific action when an event occurs. In previous versions of Jahia, event listeners could be either written in Java classes or in JSP files, or even Groovy scripts. In version 6.5, it was replaced with a much more powerful and easier to use rule system based on JBoss Drools. An example of such a rule is given below:

```
rule "Image update"
    salience 25
#Rebuild thumbnail for an updated image and update height/width
    when
        A file content has been modified
             - the mimetype matches image/.*
    then
        Create an image "thumbnail" of size 150
        Create an image "thumbnail2" of size 350
        Set the property j:width of the node with the width of the
image
        Set the property j:height of the node with the height of the
image
        Log "Image updated " + node.getPath()
end
```

As you can see rules are similar to the English language formulation and this makes it easy for integrators to use and read. Of course the vocabulary of conditions, making it easy to respond to any event with any action.

4.3 *File repository*

Jahia 6 was the first version to include the Java Content Repository as its standard file repository and build services on top of it. Actually, the integration of Jackrabbit is not a strong dependency, as Jahia uses the standard JCR API to offer access to multiple repositories. In Jahia 6 it was already possible to access CIFS/SMB file repositories, and a few other example implementations are also available (among these are FTP, Alfresco, Exo Platform, Nuxeo connectors). On top of the file repository services, different interfaces expose content through various interfaces such as WebDAV, template file browsing and Jahia's AJAX UI. On the other side of the repository, integration with a rules engine is used among other things for image thumbnail generation and metadata extraction. This rule engine is also pluggable and can be extended by integrators to perform other specific logic upon repository events.

4.4 *Searching and indexing*

Jahia comes built-in with strong support for searching and indexing, and does so by combining multiple frameworks in order to offer the following features:

- Full-text searches (Apache Lucene)
- Multi query languages support (Apache Jackrabbit)
- Facets (Apache Solr)
- "Did you mean" (Apache Solr)
- Open search
- Connectors to other search repositories such as Alfresco via EntropySoft connectors (available in our Jahia Unified Content Hub extension)

In order to use the above features, Jahia provides two technologies: full-text search tag libraries and query languages

4.4.1 *Full text queries using search tag libraries*

The full-text query feature is made available through a set of tags that are focused on searching using basic text matching with content object properties or file contents. It produces results as hits that contain information such as the target object that matched the result, an extract of the matching content and the matching score. It also supports search options defined by the JSR-283 standard such as optional words, mandatory or excluded words, search by sentence or word, etc.

Here is an overview of the features available when using full text queries:

- Search on all content within a site.
- Search on multiple sites on the server.
- Search the contents of files in the content repository.
- Search the contents of files in external repository (only with the Jahia United Content Hub extension).
- Highlight searched terms in the results page.
- Order by matching score.
- Exclusion of matching property (through content definition parameters).
- Limit results to improve performance.

Full text queries are a great way to offer an easy to use yet powerful search feature on a Jahia installation, but they are not very useful to perform more targeted queries, such as retrieving a list of the last 10 news entries or similar queries. This is where the query languages become interesting.

4.4.2 *Query languages*

The query language feature is actually a native functionality of a JCR-compliant implementation such as Apache Jackrabbit. It offers different query languages that are functionally equivalent, but differ in implementation and usage scenarios. It allows for complex condition matching and result ordering, as well as in some cases joins on multiple content object types. The result of the query is a list of matching nodes, ordered by the specified properties.

Here is an overview of the features offered by this type of querying:

- Search by content type (news, article, etc.)
- Complex conditions based on properties, or even in the case of SQL-2 or Query Object Model, joins.

- Integration with faceted search
- Integration with front-end HTML caching

The following query languages are available:

- SQL- 2: Similar to the standard SQL database language so easy to use for developers. The queries are parsed and transformed into Java Query Object Model queries and then executed as such. As this is part of JCR v2, it is relatively new and therefore there are some issues with its implementation, notably on join performance. For most simple queries it will do fine, but for more complex ones it is recommended to use XPath until this language implementation matures.
- JQOM (Java Query Object Model): this is a Java object representation of a JCR v2 query. It is possible to build these using Jahia's provided query tag libraries, or to build them directly from Java code. SQL-2 and JQOM queries are quite similar, except that JQOM avoid the parsing stage, so they are a bit faster. In practice, it is quite seldom that JQOM is used, but it might be interesting in some cases.
- XPath: although it has been marked as deprecated in JCR v2, it is still available in Apache Jackrabbit and is by far the most optimized query language. It is not as easy to use as SQL-2, but it is very useful to build very fast queries;therefore, often worth the extra effort in designing the query. There are some tricks to know how to search for multi-language content, as it is not handled transparently, in the case of the other two implementations. But even Jahia uses it internally for speed in the cases where SQL-2 performance is not fast enough.

Jahia also comes built-in with modules that use queries to provide their functionality. An example of this includes the "last news retrieval" feature in the news module. Also available is a generic "query" module that will ask for the query when added to a content page. This makes it easy for editors to be able to build custom queries at content creation time, without requiring any assistance from the integrators (of course this module should not be made available if this possibility is to be restricted).

4.5 *Authentication and authorization*

One of Jahia's strengths has always been its powerful authentication and authorization sub-system. It allows for modular yet precise controls of permissions on a wide-variety of objects or actions. Permissions

may be very granular or as coarse as desired, which makes it a great tool for deployment in small to large enterprises.

4.5.1 *Single sign-on*

Jahia integrates with the following SSO frameworks:

- Central Authentication Service (CAS) SSO, http://www.jasig.org/cas
- Java EE container authentication support
- Pluggable authentication pipeline that can be easily implemented to add support for more SSO solutions

The last framework is useful in the case of integration with non-standard SSO technologies or custom-built ones. One possible example would be the case of a mobile service provider that uses phone numbers as authentication logins. Interfacing with a custom database will integrate into Jahia's back-end, exposing user and group information directly to Jahia's UI and permissions.

While it is possible to integrate with Kerberos http://web.mit.edu/kerberos/ (the authentication valve is present in the distribution) this integration is not officially part of the tested and supported stack for Jahia 6.6.0 version

Please get in touch with the company to know the usage conditions.

Once the user is properly identified, the authorization sub-system is composed of:

- Access control lists on content objects
- Roles the user may participate in
- Permissions on any user actions for a specific role

In order to be able to set access control lists, user and group services are provided, and are of course also pluggable. By default Jahia comes with it's own user and group provider service, as well as a connector to LDAP repositories, but it is also possible to develop custom services to plugin to either a custom database or a remote service. Jahia is also capable of storing properties and user information for external users and groups inside it's own services, making it possible to store personalization data in Jahia. It should also be noted that all these service implementations are available at the same time, so there is no need to replace one with the other.

4.5.2 *Roles and permissions*

New to Jahia 6.5 is the introduction of full-fledged roles. Roles are basically a collection of permissions, regrouped under a logical name. For example an “editor” role regroups permissions for editing content and starting workflow processes. Jahia comes with default roles built-in, as well as with a powerful UI to modify the default assignments (only available in the Enterprise Edition). Integrators may of course define their own roles and permissions, as well as change the default assignments. It is also possible to add permissions in modules and automatically assign them to existing roles upon deployment.

Roles can then be assigned to users and/or groups at any location in the content repository. For example, you may define a role “editor” to a specific group in a specific section of the website. They will be able to act as that role only in that specific location in the content repository, and nowhere else. This makes it easy to delegate responsibilities in order to collaborate on content editing, reviewing and overall content management. It is of course recommended to re-use roles through the various sites and sections, as a minimal set of roles will be good both for site management and authorization performance (as HTML caching is also using roles to determine which content is viewable or not).

4.6 *Import / export*

Jahia’s import/export feature is an extremely powerful mechanism for migrating content in various different ways between Jahia sites, or even between Jahia installations. It uses the JSR-170 (or JCR) XML format as a basis for content export, along with other specific files such as file hierarchies for binary data export. All these different files are compressed in a ZIP file that may be used by the import sub-system. This makes it possible to export a complete Jahia installation, a set of sites, a single site or even a sub-section of a site using the same import/export technology. Using this feature, users can migrate content between sites, or even between sections of sites, or also use it to export content to non-Jahia systems or import from non-Jahia systems. The same system is also used for migrations from previous versions of Jahia to the newest available version.

4.7 *Distant publication*

Jahia may be deployed in multiple instances to cover scenarios where a Jahia instance is located inside a firewall, and a public instance is located inside a DMZ zone accessible from the public web. To publish data

from the inside to the outside, Jahia has a feature called distant publication (also known as remote publication), which makes it easy to automate the process of migrating data from an authoring server to a browsing one. Note that this is still compatible with user-generated content such as a deployed forum on the public instances, meaning that remote publication will not touch user generated content created on the public instance.

4.8 *Portlets*

Jahia includes an embedded portal server, which is based on the Apache Pluto reference implementation of the JCR Portlet API specification. The goal of this implementation is to offer support for integrators who need to embed portlets on content pages. This means that any portlet API compliant application may be integrated with Jahia in a few simple steps.

4.8.1 *Portlet versus modules*

In order to differentiate portlets from modules, we offer the following table that summarizes the differences:

	Portlet	**Module**
Classification	Older technology, extension to traditional Web server model using well defined approach based on strict isolation mode.	Using newer, loosely defined "Web 2.0" techniques, integration at both server or client level.
Philosophy/Approach	Approaches aggregation by splitting role of Web server into two phases: markup generation and aggregation of markup fragments.	Uses APIs provided by different modules as well as content server to aggregate and reuse the content.

Content dependencies	Aggregates presentation-oriented markup fragments (HTML, WML, VoiceXML, etc.).	Can operate on pure content and also on presentation-oriented content (e.g., HTML, JSON, etc.).
Location dependencies	Traditionally content aggregation takes place on the server.	Content aggregation can take place either on the server-side or on the client-side, but usually happens on the server.
Aggregation style	"Salad bar" style: Aggregated content is presented 'side-by-side' without overlaps.	"Melting Pot" style: Individual content may be combined in any manner, resulting in arbitrarily structured hybrid rendering and editing.
Event model	Read and update event models are defined through a specific portlet API.	CRUD operations are based on JCR architectural principles, and on the client REST interfaces allow content interactions.
Relevant standards	Portlet behavior is governed by standards JSR 168, JSR 286 and WSRP, although portal page layout and portal functionality are undefined and vendor-specific.	Base standards are JCR API, REST, JSON and XML. Defacto standards include JQuery as a Javascript framework.
Portability	Portlets developed with the portlet API are in theory portable to any portlet container.	Modules are Jahia specific.
Repositories	Portlet repositories have been a pipe dream for a long time, but despite multiple efforts they have	Modules are available on Jahia's forge, developers and integrators are encouraged and free to post

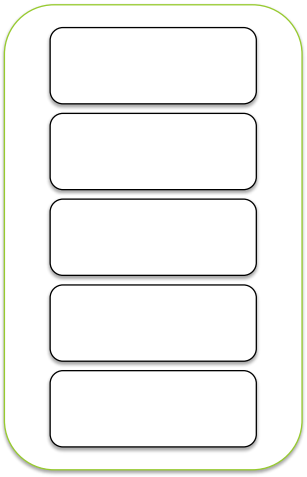
	never taken off and they stay usually specific to a portlet container implementation. .	them there, or anywhere else they wish.
Performance	A page will be fully dependent of the rendering speed of each portlet to achieve good performance, which may be difficult if closed source portlets are present in the system.	Modules have built-in support for page caching if they re-use Jahia-stored content, which is generally the case.

In general, integrators looking for a powerful and rapid integration solutions will probably want to use modules. The main use case for portlet usage is the integration of legacy applications that are only available as portlets. In the case of other systems (servlets, web services) it is preferred to use modules as the integration will be more robust, easier to re-use and deploy and to maintain,of course.

5 Performance

High performance on high-traffic web sites is often tricky to achieve. In this section we will present the technologies available in Jahia that will help you handle large loads as well as scale out.

5.1 Caches



Caches are essential to high performing web systems such as Jahia in order to be able to avoid recreating dynamic content under large system loads. Jahia uses a multi-layered caching subsystem.

5.1.1 Cache types

The cache types all use the same cache service that is responsible for providing cache implementations. Jahia now standardizes on the EHCache (http://ehcache.org/) implementation, which can range from very simple setups all the way to distributed TerraCotta (http://www.terracotta.org/) or BigMemory (http://www.terracotta.org/bigmemory) cache instances.

Jahia uses multiple cache layers to optimize the performance of page delivery:

- the browser cache
- front-end HTML caches
- object caches
- database caches

Each of theses cache layers plays a different role in making sure values are only computed once.

5.1.2 *The browser cache layer*

While not integrated in Jahia but in the browser, the browser cache plays a critical role in guaranteeing good performance for the end-user. For example, Jahia's usage of the GWT framework makes it possible for AJAX source code to be aggressively cached in the browser cache, therefore making sure we don't reload script code that hasn't changed. Jahia also properly manages the browser cache to make sure it doesn't cache page content that has changed. It also controls expiration times for cached content, so that the browser doesn't request content that is rarely changed.

5.1.3 *The front-end HTML cache layer*

Historically, Jahia has had many front-end HTML cache layer implementation. The first was the full-page HTML cache. While very efficient when a page was already available in the cache, it didn't degrade very well for pages that had a fragment of the HTML that changed from page to page, or from user to user (for example by displaying the user name on the page). In Jahia 5 we introduced the ESI cache server, which added the ability to cache fragments of HTML. This technology required a separate cache server that executed in a separate virtual machine to perform it's magic. While much better than the full-page cache for dynamic page rendering, the ESI caching system suffered from problems with inter-server communication, which was very tricky to get to work efficiently. Also, integrating the ESI cache required good knowledge of the fragment-caching model when developing templates, which was an additional burden on integrators. Jahia 6 takes the best of both worlds, by combining the sheer efficiency of the embedded full-page cache with the fragment handling of the ESI cache server. This new cache implementation is called the "module cache" and integrates fragment caching at a module level, making the interaction with templates very natural. Template developers usually don't have to add any markup in order to have their fragments correctly cached. Even when they need to control the fragment generation, this is much easier to do than in previous versions of Jahia. The "Skeleton Cache" is also an HTML front-end cache that basically caches everything "around" the fragments, and by regrouping both cache sub-systems we obtain the equivalent in terms of performance to the full-page HTML cache that existed in previous versions of Jahia while retaining the flexibility of a fragment cache.

5.1.4 *Object cache layer*

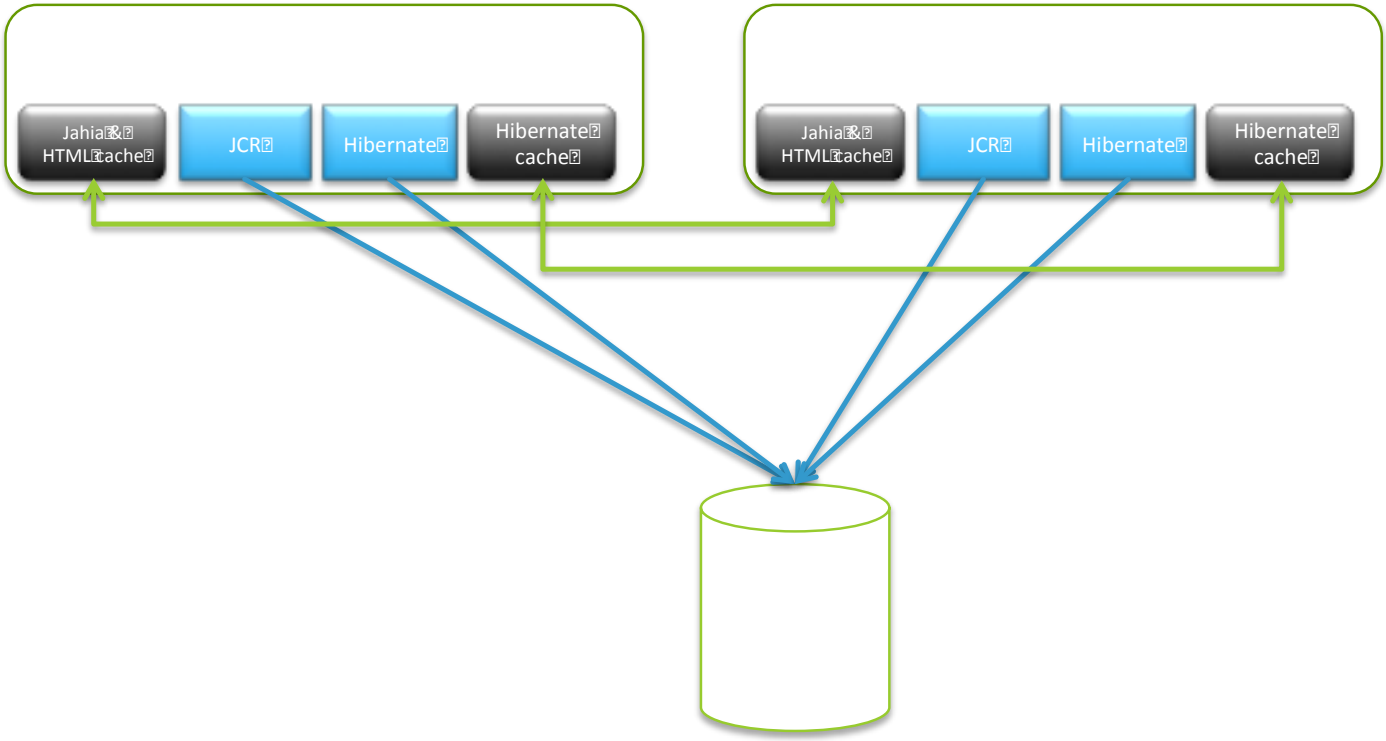
The next layer below the front-end HTML cache sub-systems is the object cache layer. This layer handles some Java objects that cannot be optimally cached by the underlying layers. In previous versions of Jahia this layer had a lot of different caches, but in the most recent versions it has been reduced to the strict

minimum based on performance testing. It serves as a layer on top of the database caches in order to avoid reconstructing objects for each model request. This is all handled internally by Jahia and it is only important to interact with these caches if integrators are directly calling back-end APIs that don't automatically update the caches (a good example of this are the LDAP user and group caches).

5.1.5 *Database caches*

The last layer of caches is the database cache layer that makes sure that only minimal interaction with the database happens. This cache is important because database communication requires object (de-) serialization as well as network communication, so the overhead of database query execution may be quite substantial. The Hibernate ORM and Jackrabbit frameworks handle this layer transparently, so normally developers and integrators will not need to deal with it.

5.2 *Clustering*



Deploying Jahia in a cluster is a very powerful way of distributing CPU and memory load to handle larger traffic sites. A typical Jahia cluster installation is illustrated in the above graph. Jahia nodes communicate with each other through cache and database layers, but also access shared resources: a shared file system and the database. The file system is used for the binary content if the server is configured to store it there, or in the database if the default configuration is used. The database stores everything else. It is therefore

very important to have a high-performance database installation, as Jahia will depend on it to scale. Jahia can also differentiate nodes in a cluster setup in order to offer more specialized processing. We will review here quickly the different node types.

5.2.1 *Visitors nodes*

Jahia "visitors" nodes are specialized Jahia nodes that only serve as content publishing nodes. They also interact with portlets or application modules to render pages and input user generated content. Using this node specialization allows the separation of visitors load from authoring and background processing loads.

5.2.2 *Authoring nodes*

Jahia "authoring" nodes are cluster nodes that can be used to either browse or edit Jahia content. This is the most common usage of Jahia nodes, and therefore it is interesting to have multiple instances of these nodes in order to distribute the load.

5.2.3 *Processing node*

In Jahia, long-running tasks such as workflow validation operations, copy & pasting, content import and indexing are executed as background tasks, and only executed on the processing node. This way, while these long operations are executed, other nodes are still able to process content browsing and editing requests. Note that for the moment it is only allowed to have one processing node. This node is designed to be fault-tolerant, so in case it fails during processing, it can simply be restarted and it will resume operations where it left off.

5.3 *More resources on performance*

As Jahia constantly strives to improve on performance, make sure to check our website for additional resources on performance, as well as our "Configuration and Fine Tuning Guide" that contains best practices of deployment and configuration to properly setup Jahia for high loads.

6 Additional resources

On our website (www.jahia.org), you will find the following resources to help you continue your experience of Jahia:

- Forum: this is the community forum, where Jahia users can exchange questions and answers. It is highly recommend to check the forums for any questions you may have as they could have been already addressed previously.
- Commercial support: Jahia also offers commercial support contracts to fit your needs. These may range from standard basic support all the way to custom assistance contracts. Please check http://www.jahia.com for details on our commercial offerings.
- Documentation: on our www.jahia.org website you will also find our online documentation ranging from end users guides to integrators documentation and API. Make sure to check back often, as we will be updating them as new releases come out.
- Videos: also available on our www.jahia.org are tutorial videos that will show you how to accomplish certain tasks or illustrate some specific functionality.

Jahia Solutions Group SA

9 route des Jeunes,
CH-1227 Les acacias
Genève, Suisse

http://*www.jahia.com*