Daisy the Open Source CMS

Steven NOELS

Managing Partner, Outerthought
Zwijnaarde, Belgium
stevenn@outerthought.org

Abstract

Daisy is an open source content management framework, and consists of a stand-alone repository server and several client applications, the most notable being the Daisy Wiki application. Daisy has a strict two-tier separation between repository server and clients, which communicate using an HTTP- and XML-based interface.

This whitepaper highlights some of Daisy's innovative concepts, and explains where Daisy is different from the multitude of other CMS applications out there. These distinct Daisy concepts make Daisy an ideal candidate for managing diverse sets of information, for both website content management, software documentation and for intranet knowledge sharing.

1 Distinct Daisy Concepts

1.1 A Big Bag of Documents

A Daisy repository can be envisioned as a big bag of documents: no more, no less. Documents can be grouped into Collections, and can be queried upon using their associated metadata, but the repository model itself has no concept of hierarchy. Often, a content repository imposes a hierarchical approach to managing repository contents, using the all-too-familiar "folders" concept. Looking at the typical use of content management systems, the repository hierarchy will then reflect either the organisational structure of a company, or the navigation hierarchy of the website(s) published out of the repository: there's no middle ground between both approaches. This makes reuse of information, and sharing documents across department walls harder if not impossible. When the company organisation changes, the document repository will need to reflect these changes as well.

Contrasting this hierarchical approach with current internet trends, we now find websites where information is tagged rather than stored into a strict hierarchy. With Daisy, documents are stored into a large bag, and labelled using metadata, which is then used to create virtual views on the repository. Documents can emerge in several different places of a website (or not): there is no direct link between the classification of documents in a Daisy repository, and the navigational or hierarchical structure of a website published out of a Daisy repository. With Daisy, there are no strong ties between the structure of the organisation working on the repository content, and the repository structure itself: information is available for liberal reuse and sharing.

This enables flexible reuse of repository content, and postpones the decision of how to orderly publish content out of the repository until setting up a website: this website becomes a virtual view on the repository content. Also, documents can easily be added or removed from a virtual view by simply changing their metadata labels: content classification sits closely to the affected content rather than in some elaborate and disconnected hierarchical scheme. Of course, a Daisy website offers all the structural navigation tools one expects to efficiently access the content: this is managed in a separately maintained navigation document. This way, multiple Daisy websites can provide different structural views on top of a non-structured repository.

1.2 Sensible Use of XML

All data coming out of the repository consists of XML documents with a well-defined grammar. The rigid structure of XML isn't imposed on the content parts themselves however: these can consist of either squeaky-clean XHTML, or plain binary data (images, office document formats, ...). When retrieving a Daisy document from the repository, the resulting XML document is an envelope containing an XML representation of the metadata field values, and pointers to the relevant binary content parts. These parts can be obtained directly from the repository API across HTTP, and it is a concern of the publishing (or editing) front-end application to treat them in any specific way (like inlining images into a web page).

Daisy doesn't force users to encode all textual data into XML, and the Daisy repository is specifically tuned towards management of semi-structured data, a format which encompasses a vast range of possible applications, like website content management, but also many typical office document structures, software/project documentation and quality control-related SOPs. All textual data is however preferably stored in XHTML, which is actively weeded out of browser- or layout-specific tags, making sure documents can be repurposed across multiple media, like print or mobile applications.

The use of XHTML as a base encoding format for textual data also enables the use of an in-browser WYSIWYG editor, working across common browser families without requiring any specific plugins, or falling back onto an awkward forms-based approach to edit textual data. The editing component used in the Daisy Wiki uses cross-browser JavaScript code, has a modest footprint, yet enables users to edit tabular information, insert images, choose between a number of paragraph styles and more. There's no learning curve associated with the use of the rich-text editor, and its features will apply to many, even sophisticated types of documents.

Using a semi-structured approach, and the productive use of XML to communicate between repository and repository clients, Daisy adds few constraints to the type of information which can be managed by the repository, yet its ReST-based interface uses XML to provide a strong contract between content management application components.

1.3 Typed Documents

Daisy documents adhere to a Document Type, just as XML documents typically adhere to an XML grammar (such as XML Schemas). Document Types consist of both Field and Part definitions, with Fields being strongly-typed metadata labels. Using the Document Type definitions, the Wiki editor is automatically configured and/or adjusted: once a Document Type has been defined (or edited), the editor automatically provides editing functionality for the required Fields and Parts.

Unlike other content management systems, no development effort is required to create and maintain repository editing screens for the Daisy Wiki: this is done automatically.

All document metadata is stored with the document, but not inside it. This means Daisy can rely on common and proven implementations for easy, fast and robust indexing and querying of metadata, such as relational database technology. The content parts are full-text indexed (where applicable), so both metadata and document content can be searched, and are stored in the most applicable storage layer.

The metadata fields are centrally defined and managed, so that field definitions can be reused across multiple document types. Fields values must adhere to a set of built-in types (such as dates, boolean, strings), and can be restricted using selection lists.

Daisy Document Types are similar to XML grammars: they help the document editor to create consistent documents, following a predefined structure. The Daisy Wiki goes beyond this, and also uses the Document Type to automatically configure the editor. By providing a strict set of field datatypes, Daisy is able to maintain "classic" database indexes making sure search and retrieval are done in a performant way.

1.4 Two-tier Approach to Content Management

Daisy makes a clear distinction between the repository server and the content management application, which acts as a client of the repository. All repository access is done through a ReST-like, HTTP/XML-based API (Application Programming Interface), enabling external applications - which could be implemented in any programming and/or scripting language which support the trivial HTTP protocol - to access the repository server and content as well.

While this not only caters for automated repository operations, it also ensures that the repository model is kept clean and doesn't impose any particular front-end UI design: it is possible to create command-line applications connecting to the repository to query, access, add and retrieve information from the repository.

Other CMS systems often mix the publishing, content management (i.e. editing) and repository functions into a single, monolithic (web) application, or require a proprietary client application to connect to the repository. While the Daisy Wiki does both editing and publishing in a single comprehensive web interface, it is strictly separated from the core repository services, and a blueprint publish-only application is shipped as well (which could then be used as an outward-facing front-end of a Daisy-based CMS solution).

To make client applications aware of changes in the repository (like documents being added or modified, or the creation of new user accounts), all relevant change information is published (wrapped in XML messages) onto a standards-based JMS message bus. This enables client applications to act upon repository changes (e.g. flush document schema caches) when applicable, and makes easy integration with external applications possible as well.

Few CMS systems have a clear separation between the CMS application and the repository. Achieving this separation has been a major goal of the Daisy design: its two-tier approach, combined with the Wiki-style of information management, is unique in the market.

1.5 Functional API

The HTTP/XML-based repository API aims to provide a reasonably high-level repository access interface. The Daisy repository API offers a set of clear-cut and easy-to-use operations to client applications.

First of all, the API has no concept of "client sessions", which means it scales well even with large numbers of client applications, as each transaction with the repository is atomic and has no side-effects outside the scope of the individual request.

Secondly, the API consists of a set of high-level operations, which make immediate sense to the novel Daisy developer. Contrasting with other repository APIs, where repository content is described as abstractly-named "nodes", the Daisy API talks about Documents, Fields, Parts, Comments: easy to grasp concepts without over-generalisation.

Thirdly, the API consists of simple HTTP calls with optional query parameters and XML document payloads. It's the kind of HTTP interface we've come to expect since Google, Amazon and Flickr.

To make things even easier for the Java developer, the entire HTTP/XML API is wrapped into an higher-level Java API, making the request/response specific behavior of the Daisy API totally invisible. Even better, the same API also lives in the back-end, and is capable of the same operations without the overhead of network traffic, associated with a remote API. Code can be moved between the front- and the back-end tier without changes since the API remains the same.

Also, Daisy doesn't have a multi-level API: all the available features of the repository (like querying, read/write operations, but also locking information and document comments) can be accessed, modified or obtained using a single, comprehensive, and well-documented API, using a protocol onto which the Web has been built upon, and for which exists support in any programming language out there.

1.6 Active Documents and Navigation

A website typically consists of a mix of both static and dynamic information. Web pages should be able to present both static content, and information retrieved using repository queries or external aggregated content.

Daisy provides the novel concept of in-page queries: simple queries inserted inside a document, which are dynamically executed when a document is displayed. This can be used to embed a list of news items inside a homepage, or to build dynamic navigation documents. The navigation structure of a Daisy-based website is managed in separate navigation tree documents, which can contain queries as well. Of course, query results are filtered against the access control rules, so users will never see a link to a document for which they have no read access.

The combination of queries inside navigation trees, and the automated filtering against access rules, caters for complex navigational hierarchies of websites, which easily grow as the site's content grows as well.

The Daisy Wiki and blueprint "publish only" application are built on top of Apache Cocoon, a renowned web application framework which is ideally suited for dynamic content aggregation and multi-modal/-lingual content rendering. Daisy provides extension hooks to easily access the full power of Cocoon, e.g. to aggregate external content (like RSS feeds) into website pages.

1.7 Wiki-style Repository IDE

Daisy ships with a Wiki-like front-end application, which doubles as a comprehensive repository IDE. All the repository functions are accessible from within the Daisy Wiki application, plus the Wiki itself is an ideal starter application for building a knowledge-sharing intranet. With the repository IDE being a simple web application, even power users don't need access to specific repository client software, reducing the burden on IT support departments: if users can read the Wiki, they also can edit it (barring access restrictions of course).

Using the Wiki, information can be added, edited and retrieved from the repository. The Wiki offers access to the version history of all repository content. It allows checking up on document referrers: document links are fully managed by the repository making sure links and cross-references are consistent.

The Wiki IDE user interface is easy to learn: users will feel immediately at home to browse and edit information.

Alongside the Wiki, other (web or commandline) applications can be built which provide optimized access or specific functions: Daisy is a content management framework designed to develop any kind of content management application, thanks to its two-tier design and the clear-cut repository API.

1.8 Browse and Query vs Navigate

In Daisy, publishing websites are a concern orthogonal to content management. Website structure (or rather: navigation) is contained in navigation tree descriptions, of which multiple can exist. Navigation trees can be made dynamic by inserting queries into the tree, whose results are automatically converted to navigation nodes.

This, combined with the in-page queries, caters for a different approach towards information discovery: the plenitude of information contained inside the repository can be shown piece-wise, dynamically linked and retrievable, rather than by using elaborate, unscalable and difficult to maintain hierarchical classification systems.

With a query language as familiar as SQL (or at least a simplified and slightly adapted version of it), people are encouraged to explore the repository at their heart's content.

1.9 Centralized Access Control

Access control configuration can sometimes be confusing to administer, due to the fact that access control rules are often dispersed throughout the repository, attached to the hierarchical repository structure or the individual documents. Daisy solves this by storing all access control information into a single, central location. These rules follow a simple logic: first of all, you need to define which documents a given rule applies upon. For this, you'll use a subset of the query language, so you're free to use any sort of constraint or document selection mechanism you can invent. Secondly, you'll define the users (or roles) for which the rule is applicable, and lastly, you'll define the type of operation you're allowing or disallowing.

Access control checks are pervasive: query results are filtered against ACL rules, as are navigation tree nodes. This way, users are never presented with links to documents they have no access to, apart from direct links inside document text.

2 Find out more

Daisy is an open source project, licensed under the commercially-friendly Apache License (version 2.0). Its source code and release distribution are made available from the community website: http://cocoondev.org/daisy/.

3 Biography

Steven Noels is co-founder of Outerthought, a geek-level technical Open Source Java & XML competence support center. Prior to that, he held several key positions as technical SGML/XML consultant and tools specialist, working for some of the leading XML and internet companies in Belgium. Steven has been actively advocating and using Java, XML and open source technologies since many years. He maintains a weblog at http://blogs.cocoondev.org/stevenn/. Steven is a Member of the Apache Software Foundation.