Nuxeo CPS: an open source framework for the development of enterprise content management and collaboration applications

Julien Anguenot, Stéfane Fermigier,
Florent Guillaume, Lennart Regebro, Tarek Ziadé
Nuxeo, France
{ja,sf,fg,lr,tz}@nuxeo.com

Jean-Marc Orliaguet
Chalmers University of Technology
Göteborg, Sweden
jmo@ita.chalmers.se

Abstract – **Nuxeo CPS (version 3) is an open source, GPL-licensed, framework designed for building content management, collaboration and business processing applications. It addresses the needs of web applications developers to easily create rich document types, with workflow-based access control, portal-like interfaces for accessing documents and information, and rich services that match the features of best of breed non-free (proprietary) offerings in the domain, including: single-sourcing, versioning, indexing, metadata management, localization and translation, subscriptions, notifications, commenting, theming (user interface customization), directories management (users and groups directories for instance) that can be external (LDAP, SQL, etc...), content scheduling and staging, syndication... Higher level components, including mail, calendaring and discussion applications, have been developed on top of the base framework, as well as custom business applications featuring complex document types and workflows.**
CPS3 has been developed by Nuxeo SAS, an innovative French company specialized in open source software and services, and a community of contributors, since 2003, and is now in use for majors projects in the Administration, in France and other countries, and in several multinational corporations. It is based on, and extends in substantial ways, the Zope application server and its Content Management Framework.

I. CPS3 PROJECT MOTIVATION

Zope is an innovative application server started in 1996 and fully open sourced under the ZPL in 1998 by Zope Corporation, an American company. Zope provides an object database, an application server with an object broker that converts HTTP (or other protocols) requests to objects and methods invocation, a security model, a templating and a scripting language, a basic component model for extending the application via dedicated plugins also called "Products", etc. [ZB].

The Content Management Framework (CMF) [CMF], released by Zope Corp. in 2001, provides a component architecture and some basic services to write web content management applications: membership services, indexing, document types, and a powerful document-centric workflow engine called DCWorkflow (which was release after the CMF was released, and was first an add-on to the CMF before being integrated into it).

Nuxeo had worked on several customers projects in 2001 and 2002 using the CMF and some basic extensions, but it appeared quickly that only simple content management – mostly web publishing applications – but no truly collaborative applications with thousands of users, could be created with the CMF and basic extensions, and that some major extensions, refactorings, and new model implementation were needed to compete, in terms both of functional scope and performance under heavy load, in the so-called "ECM" (Enterprise Content Management) and collaboration applications markets dominated by proprietary software companies like Documentum, OpenText and Interwoven.

The CPS3 project was thus launched in 2003 to address these goals, after the company had restructured itself with a R&D department, and considerable resources have been devoted to develop the new framework. First customers projects developed on top of CPS3 were in production by the end of 2003, and stable releases of the software have been issued regularly since 2004, drastically improving the framework.

II. EVENT-DRIVEN FRAMEWORK

In an enterprise content-management system (ECMS), many components have to be tied together to provide a high level of functionality. Some of the components are native to the framework, and some are added later as optional modules.

Many of these components have to cooperate with each other to provide their intended functionality. A simple example would be a workflow component and a mail notification component that have to work together to send an email to some selected users when a document changes state in the workflow (for instance, upon creation of a new document or publication).

Whereas the existing Zope ECMS [CMF, Plone] use explicit knowledge by each component of the other components that may interact with them, CPS chose an event-driven framework for these interactions. In the example above, the workflow component would send an event when the document changes state, and the mail notification component would listen to that event, and dispatch a mail according to the policy configured for it.

This provides a greater flexibility to the developer and to the integrator. The developer can choose to send an event whenever his module does a significant

action, but without having direct knowledge of what other modules may or may not act on this event. He can also choose to register his module as a listener for some kind of events, and decide what has to be done when an event is received. The integrator can choose what modules she installs, and knows they'll work to gether if it makes sense for them. Furthermore, the policy of the interaction (to whom the mail should be sent, in our example) can be clearly specified in the configuration of the listener.

III. REPOSITORY AND PROXIES

Zope's object database (ZODB) lends itself naturally to hierarchical object structures, and this is how most ECMS are implemented in Zope and CMF, including earlier versions of CPS. The object orientation and the orthogonal persistence of the ZODB leads to a rapid development process, removes many of the troubles you get when using relational databases, and is more natural for document databases and document repre sentation. But relational databases facilitate multiple orthogonal views on data in a way that hierarchical object databases don't.

In CPS, unlike the classical approach outlined by the default component Zope CMF, documents are stored in a central repository (see [BER] and [JCR] for back ground information about repositories) and “proxies” pointing to the documents are spread throughout the website structure. The user only sees proxies, and for her they look like normal documents, but the proxies are just a kind of pointer, a link to the real informa tion. In the repository, documents are stored with an additional version number.

Because the proxies have additional metadata, a number of advanced use cases can be fulfilled:

- Multi-publication: A single document can be published in different parts of the site. This is because a document can be pointed to by several unrelated proxies.
- Versioning, check-in/checkout: Documents can be present in several different versions in the repository, and a proxy can point to a specific version, so different proxies can point to different versions of a single document. This makes it possible to have a “validated” version somewhere, and a “working” version somewhere else; documents don't need to be removed from publication while they are edited. The versioning model chosen by the default CPS implementation is based on a simplified version of WebDAV's DeltaV protocol [DeltaV].
- Multilinguism: A proxy can point to several documents, the one actually displayed being chosen according to the user's preferred language.
- Multiple workflow states: Because the proxy objects are the ones that follow the workflow, it is possible to have the same version of a document be published in some part of the site and still be pending in another part where the local reviewer has not had time to do his work. It is achieved by applying a different workflow on each proxy object representing the needed process in the context of the proxy object.
- Multiple storages: Because the repository is the central point of access for storage, it is possible to implement and plug different storages on the repository to store data into other databases like XML or SQL databases or even plain file system. The repository service unifies all these backends and provides a single view of the storage space (this feature is commonly use to set up one different database per year, for example).
- Placeful security: Only the proxy objects are aware about the security that is applied to them according to the context. (i.e : parent folder) and controlled by the workflow applied on it in a placeful manner.

All these advanced features come nearly for free once you implement a proper indirection, the proxies, between the hierarchical tree and the data.

IV. DOCUMENT TYPES

A. Schemas, widgets and layouts

Document types are created in CPS using the ab straction of schemas and layouts.

A *schema* describes the structure of a document as a set of typed *fields*. It is its semantic. The fields may be strings, integers, lists of strings, files, or any arbitrary complex data type. A default set of fields is provided.

A *layout* specifies how a schema is to be rendered and edited, using a list of *widgets* that each specify the user interface and display constraints for one or more fields. New field and widget types can easily be added by the programmers using dedicated registries.

The “layout” abstraction is important, because then a field is not tied to a specific widget for its display. This allows changes to the View or Controller (in the MVC paradigm) completely independently of the Model. Different layouts can be used for the same schema depending on the circumstances, which makes it possible to have different “edit” and “view” pages, or have several “view” pages that display a subset of all the information available in the document – this is typically used for the Dublin Core part of the metada ta for instance.

Schemas and layouts are a fundamental innovation in CPS3, in that they allow the quick development or the customization of rich document types. They are now the foundation for most of the CPS user interface, including directories and portlets (see below).

Although schemas are not a new concept in ECMS, CPS3 provided the first schema system for Zope where schemas were stored in the object database. This en abled us to create CPSTypeMaker, an innovative tool used to automatically create and modify document types on the portal. It provides a complete WYSIWYG

editor that generates and edits CPS schemas and lay outs definitions used by CPSDocument to create in stances of a given type. This editor brings CPS's pow erful document development to a higher lever, making it usable by non-developers that wish to easily extend or create document types on the portal as well as en abling rapid-prototyping. Furthermore, the definitions generated by the tool are more reliable and less bug-prone than human-made definitions.

B. *Flexible documents*

In addition to obeying a schema and a layout ac cording to their type, documents can have a “flexible” part that can be changed dynamically by the user. The user is able to manipulate a given instance of a docu ment and add new fields and widgets to it. Using this, the user changes the schema of his document easily, for instance by adding several images or different text zones, and he can also change the way a given field is displayed, by changing the configuration of the widget associated to it, or by changing the widget completely.

Not all sites want this flexibility of course, and this behavior can be turned on and off per document type, and, inside a document type, per schema.

These flexible documents types (and actually all document types based on schemas and layouts) are CMF content types based on CPS's “Flexible Type In formation”. This object describes for a given content type what schemas to use, what layouts, which of them are flexible, and additional information like “clusters” that help in having several different views for a single document (main page, metadata page, short view, etc.).

C. *Vocabularies*

Vocabularies store a correspondence between keys and a human-readable label. They can be ordered, and may get their data from an external source or be computed. A central repository of vocabularies is pro vided by CPS and users can control it through the CPS user interface. Local vocabularies are available as well to provide placeful restricted vocabularies to granted principals. They are typically used with widgets within the forms.

D. *Data Structure / Data Model*

When interacting with documents defined through schemas, several abstractions are used. The data as it is seen by the user, and the data as it is manipulated by the code, use different abstractions.

A *DataStructure* holds the user-oriented version of the data, designed to be displayed, either computed from the real data before display or returned by the browser from input fields.

The *DataModel* is a transient object that holds infor mation about a particular document's data structure in the native format for that data. For instance, in the DataModel, an integer is really represented as a python integer, whereas the corresponding DataStruc ture, designed for display, holds a string representa tion of it. The DataModel also includes the underlying schemas and the storage options. It acts as a single point of access to the document structure. It is the one responsible for validating access to the data according to right restrictions (ACLs) specified on the fields. It could also be used as a cache of the document's data.

The actual storage into objects from the DataModel, or the extraction of data from an object is performed by a storage adapter.

E. *Storage adapters*

A *storage adapter* is used by a DataModel to get/set the data from/to somewhere. It is is parameterized on one hand by some context, for instance an object, and on the other hand by the schema that drives it.

A storage adapter can also be linked to no object, for instance for during object creation.

The most basic implementation (AttributeStor ageAdapter) is simply using attributes on an object. A variation on it (MetadataStorageAdapter) has knowl edge of the Dublin Core API of the CMF, and uses ac cessors so that the “title” field is not directly stored in the “title” attribute but is get/set using the Title() and setTitle() methods.

More complex storage adapters, used in directories for instance, store data in an SQL table, or in LDAP.

V. DIRECTORIES

A. *The directory abstraction*

It has been recognized early in the development of CPS that not all data manipulated by an ECMS can be considered as a “document” from the user or infras tructure's point of view. For instance, this is the case for the information about the users themselves, the way they are organized into groups. These “non-docu ment” data have several common features:

- they are not workflowed or versioned,
- they are stored in an external storage (LDAP, SQL),
- they have to be searched efficiently,
- they still obey a clear schema.

The “directory” abstraction of CPS covers these use cases. A directory contains entries that obey a schema and are displayed using a layout, thus reusing all the framework described above for documents (indeed, when displayed and edited by a user, there may be no difference at all between a document and a directory entry).

In CPS, typical directories include:

- members
- groups
- roles

B. *Directory backends*

A number of backends have been written to cover the common use cases of data storage or data aggregation for directories.

The standard data storage methods are:

- ZODB storage (inside a BTree)
- LDAP storage
- SQL storage

Additional backends provide “meta-directories”, which can aggregate different underlying directories and make them appear as a new one. CPS provides two of these:

- Meta Directory
- Stacking Directory

The *Meta Directory* builds each of its entries by taking some fields from some underlying directory. For instance, the name and password fields from an entry can come from an LDAP backend, while the login time can come from a ZODB backend.

The *Stacking Directory* does a union of all the underlying directories, so that you can have for instance members coming from LDAP and members coming from an SQL table appear inside a single homogenous directory.

C. User management

Using the directory abstraction, it becomes much easier to manage users because the directories expose a single unified API. The standard user management components (called “user folders” in Zope) can be replaced by a simple one that specifies which directory to use to store users, and how to identify and authenticate them. This is the role of CPSUserFolder.

Using meta-directories, an arbitrary number of user sources can be aggregated and used seamlessly by the rest of the framework. This makes it possible to adapt to complex organization's needs. For instance some of the users may come from a read-only corporate LDAP directory, with their last login date and picture stored in an associated directory in the ZODB, and some other users may be stored in several SQL tables on different servers.

CPSUserFolder can also take over the role of CMF's MembershipTool and MemberDataTool, by delegating all their functions to directories.

D. Vocabularies and directories

The vocabularies objects used by some widgets can be interfaced to directories, using the DirectoryVocabulary.

This type of vocabulary specifies which directory it refers to. The directory in turn has the specification of what field to use as primary key (id), and what field to use as user-visible representation (title). Using this, the vocabulary can synthesize something that can be used in a menu or a list of checkboxes.

This bridging is very useful to provide access to standardized table-like data like list of physical items whose full definition is stored in a directory.

VI. WORKFLOW-DRIVEN MODEL

Most of CPS behavior related to documents, security, users is workflow-driven. CPSWorkflow [TCH] extends heavily the basic document-based DCWorkflow by providing:

- local workflow configurations (documents may follow different workflows depending on where they are located in the website's arborescent structure) see the proxy and repository section in this document,
- specific workflow behaviours (transition flags) that address the needs of multi-publishing, versioning, grouping related document in the same workflow, etc.
  Logic is hooked on the transitions coping with specific logic rules defined by the user or still possible actions allowed to the user in a given context, etc... Here, this is specific business logic that is implemented.
- state behaviours protecting the object following the workflow since all the transition are shared in between the states. (Used only in the workflow stack transition flags.)
- “stack” workflows that can be used for dynamic workflow chain allocation that will take care of dynamic local role distribution to principals according to the policy of the stack definition associated to the stack object.

The workflow stacks are the latest and most complex of these extensions. A stack provides dynamic behavior (in the sense that they can be different for documents following the same workflow) of who is allowed to work on a document, and along what rules. A typical use case is when, during the validation of a document, the reviewer decides that he has to postpone his decision until the document has been approved by specific users. The reviewer then adds the users to a stack; these users can validate or not the document and when the stack is empty the document returns to the reviewer for his final decision.

Note that while the term “stack” is used to describe them, these objects can actually implement any data policy they want. They could be simple sets, or associated to more complex algorithms.

Several stacks associated to several stack definitions can be associated to a given workflow and thus to a given object. The worklow tool knows about their existences and how to synthesize the permissions / roles maps provided by these stacks. It knows as well how to perform diff through the process.

As well, the stack object are under versionning within the workflow history associated to an object so that we can track the changes of the stack themselves them during the process.

With the stack workflow extension it gives to CPS an hybrid workflow model extending the basic document centric model provided by DCWorkflow. It keeps the use of the workflow really straight forward for the users and the developers but still provides

powerful and dynamic smart data structures hooked on the workflow that allows the possibility of having dynamic work items on a given state with a user de fined policy for following a transition, changing state, etc... It can be object such as tasks for instance. Reg istries are available to extend the basic set of work flow stack elements to allow the user to implement and hook its own business logic on its custom work flows.

Moreover, the transition flags can be seen as activi ties embedding business logic controlled by the transi tions.

VII. USER INTERFACE

A. CPSSkins

CPS was first developed with a portal user interface based on “boxes”: graphical elements of information that either the webmaster, the users locally responsi ble for content management, or registered end-users, can use to customize the content and presentation of information displayed for themselves or for other users of the portal.

CPSSkins [CPSSkins] was developed at Chalmers University of Technology, then integrated in CPS, to address the shortcomings of the “boxes” interface. CPSSkins features:

- a WYSIWYG interface for editing the global look of a site, also called a “theme”; an arbitrary number of themes can be created and used for a site,
- a visual editor with drag and drop facility for moving information boxes, or “templets”, around the page, and reconfiguring them,
- a user interface for managing “portlets”. This gives users that are not site managers the possibility to add visual content to pages while preserving the overall site design.

Using CPSSkins, a site can have its look drastically altered in a matter of minutes, either in totality or in selected sub-sites. This caters for the need of complex enterprise websites where different organizational units need different user experiences. This also makes it possible to build and distribute CPS Business Tem plates that allow dramatical changes to the user inter face to fit business needs (for instance a public web site's backoffice should look different than an online community).

B. CPS Portlets

CPS portlets are CPSDocument-based objects imple menting the concepts defined within the JSR 168. The portlets are then using the same concepts as docu ments and directories. (i.e : schemas / layouts / wid gets / vocabularies). Schema, layouts, widgets and vo cabularies can thus be shared in between documents , directories and portlets.

CPS portlets are managed by the WYSIWYG inter face provided by CPSSkins and CPS3 provides a set of default portlets such as navigation, RSS, etc...

CPS portlets provides an integrated RAM cache manager that enhances the page rendering time when the portal has to scale. The cache invalidation system relies on the event service system of CPS3. It can easi ly react on user event such as modification or docu ment creations, etc.

Note, the pages are never loaded entirely with this system avoiding a lot of useless load on the applica tion server. (the system is in clear opposition with the old main_template / macros system that is not suit able with the Zope application server. Zope is not PHP)

C. Subscriptions and notifications

Within CPS3, content is driven by several workflows and handled by different actors depending on their rights at a certain time and in a given context. The workflow schemas within large scaled organizations can also be extremely complex. It seems to be quite vi tal for portal actors to be informed when they have to do something (validating, publishing, transmitting, etc.) without having to lookup on the portal to find what they need to do.

CPS provides placeful subscriptions controlled by the users granted with manager rights at a given place. It can subscribe people according to their roles or let them subscribe themselves through dedicated actions and interfaces. The user is notified by mail when an event occurred for which the user has sub scribed. (or because he has the corresponding roles necessary to receive the notification that the manager set)

The default notification mode is real time. The user will receive a message directly after the event oc curred. It is possible to subscribe in a daily, weekly or monthly basis. Here, in this case the message notifica tions are archived and then scheduled by an external service such as cron under *NIX.

Recipients rules objects take care of computing the recipients based on roles, workflow or they are explic it ones (when the user is subscribing). It is possible to write computed recipient rules where the user is free to implement whatever logic he wants to define recipi ents for a given notification within a given context.

A central management tool is provided to the site administrator in ZMI to add events in contexts, change notification event messages.

The notification blackened can be extended for SMS, Jabber or whatever kind of notification by de sign.

VIII. CPSMAILACCESS AND CPSSHAREDCALENDAR

A. Five use in CPS

To provide a flexible way to migrate CPS to the new Zope version (Zope 3), new products are being devel

oped based on Five, which is a framework that allows you to create Zope 3 products that can be used within Zope 2.

The binding is not complete, so there are still some Zope 2 specific parts. But the gap between a Product using Five and a Zope 3 is much smaller than the gap between a pure Zope 2 product and a Zope 3 product. This is mostly due to the fact that Zope 3 products are architectured in a totally different way.

At this time, there are two CPS products that are based on Five. And CPS will have more and more of these Five based product, until the gap between CPS 3 and a product fully based on Zope 3 is minimal.

B. *CPSMailAccess*

CPSMailAccess is a full-featured webmail that cre ates a folder tree containing mail stubs on the Zope side. A mail stub is an object that just gathers minimal information about a mail, like its unique id and head ers, to fullfill 90% of client manipulations. Further more, it indexes all mails for fast Zope-side searches. The tool keeps a list of stateful connections to the mail server to minimize network access.

This architecture lets the user synchronize his mail box by comparing the Zope-side tree to the mail serv er mail tree. The interface to send mails is based on AJAX principles and lets the user work as if it was us ing a heavy client. For example, validations on the re cipients emails are done on server side through an asynchronous javascript connector, thus avoiding a full page reload. The ergonomic is also enhanced by letting the user drag and drop mails and folders.

C. *CPSSharedCalendar*

CPSSharedCalendar is an advanced, flexible calen daring component. It allows CPS users to access per sonal calendars using a web interface. Events on the calendar can be shared between multiple users, and a user can invite others, which makes the event appear on the other's calendar.

Features of the component include a web-interface for managing calendars, integration with iCalendar clients (Apple iCal, Mozilla Sunbird, KOrganizer) us ing iCalendar, invitation workflow, meeting support, meeting helper that looks for free time, and much more. See the features list and the use cases for more information.

IX. DEVELOPMENT PROCESS

CPS is developed using an agile, “test-driven” ap proach. When new coded is added, unit tests or func tional tests are added at the same time to ensure qual ity and non-regression [BEC, HOL]. When a bug is iso lated, a test is written for it so that in the future no re gression can happen. Periodical automated processes run the whole test set and report any problem. Load testing is also carried out on preproduction websistes.

Nuxeo joined in 2004 the EDOS European project [EDOS] to further enhance its tools used to manage the quality and address the complexity of the project which currently comprises 30 packages for its “base” version, 45 packages for its “full” release (all the pack ages that are currently supported by the core develop ment team) and a dozen additional packages (pack ages under development by the CPS core team, or de veloped by independent parties).

CPS is originally a project of Nuxeo but it is open source and has been extended by third-party develop ers. The very innovative CPSSkins project, for in stance, was started independently before being merged during the CPS 3.3 development cycle. Trans lations in languages other that French and English are also contributed by third parties.

Development “sprints” (multi-days sessions of fo cused and intense development, a concept loosely based on Extreme Programming and other agile methodologies) have been organized to foster collabo ration between Nuxeo and third-party developers.

Moreover, to ease collaboration with other parties, a community website [CPO], several mailing lists and a bugtracking system are online and publicly avail able.

X. APPLICATIONS AND ADOPTION

CPS has been used by Nuxeo, by major IT compa nies in France and Europe, and directly by developers that download the freely distributed framework, for projects such as:

- major Internet websites for French ministries (Ministry of Interior[1] and Ministry of Defense[2]), local administrations (city of Lyon[3], region of Brittany) and public-sector organizations (French Institute of Statutory Auditors[4]),
- major intranets for French and foreign ministries (Senegal Government, French ministries of Culture and Interior),
- collaborative applications for major companies (Groupe Suez, STMicroelectronics, Central Bank of West-African States),
- paper and electronic mail and files management application for the French ministries of Interior and Justice
- civil-state management for the city of Antananarivo and other cities in Madagascar,
- all the web sites of the Chalmers University of Technology[5] in Göteborg, Sweden.

In most of these projects, the fact that CPS is open source has been a major criterion for its selection by the customers, who value the higher adaptability of the software, the long-term strategic independence it guarantees, the benefits brought by mutualization be

[1] http://www.interieur.gouv.fr/
[2] http://www.defense.gouv.fr/
[3] http://www.lyon.fr/
[4] http://www.cncc.fr/
[5] http://www.chalmers.se/

tween similar projects carried out for different customers or by different users, and the lower price when the application has to scale to tens of instances and/or tens of thousands of users.

Thanks to the efficiency of the Open Source development and mutualisation models, and the technical excellence of the Zope framework, CPS has been compared favorably by customers and industry analysts against the market leading proprietary solutions.

XI. FUTURE WORK

A. *Evolution CPS 3.4 and beyond*

CPS 3 has still a number of planned improvements to its framwork, that will appear in CPS 3.4 or later Zope 2-based releases. Among them are:

- Use of CMFSetup for full XML input/ouput of the site configuration and of the content
- Speed improvements by optimizing the security mechanisms used by the proxies
- Use PAS for authentication with directory plugins for storage.
- ...
- Five
- Scalability: we want CPS to scale to hundreds of thousands of users and terabytes of document storage.
- ...

B. *Refactoring to Zope 3*

Zope is currently undergoing a transition from the now classical Zope 2 to the very innovative, fully component-based Zope 3 [RIC, WEI]. But although architecturally superior to Zope 2, Zope 3 currently lacks the content management facilities provided by the CMF.

Work has been started in mid-2004 at Nuxeo and other companies to gradually migrate the functionalities of the CMF and CPS3 to the Zope3 framework. This migration is to be understood as a complete rearchitecturing, and not a simple port, but upwards compatibility with the existing CPS 3 instances is to be taken into account during the development.

To ensure the wide adoption of the new underlining framework, and to benefit from all the brainpower of the Zope ECM and Zope 3 developers communities, Nuxeo has started a worldwide collaboration with other companies and individual developers by setting up a collaborative website [Z3lab] and a mailing list and organizing or participating in sprints dedicated to the Zope 3 ECM project.

It is expected that a fully Zope 3-based version of CPS (tentatively called CPS4) will be available in prototype form by the middle of 2006.

XII. RELATED WORKS

Plone [Plone] is another content management based on Zope and the CMF. Plone doesn't use the repository/proxies model of CPS which makes developing version-controlled application harder than with CPS. Plone, as of version 2.0, uses Archetypes, a framework for creating content types similar to CPSSchemas/CPSDocument, but Archetypes is mostly based on code generation or global schemas, whereas CPS has placeful (local) schemas that can easily be customized by the administrator in the ZMI (placeful schemas also allow per-document schemas, embodied in CPS as flexible documents).

Silva [Silva] is a content management based on Zope (without the CMF) and with a strong emphasis on collaborative XML documents production.

The CPS event system is based on ideas borrowed from Zope 3. The main difference is that in CPS events are sent to a placeful event service, when in Zope 3 they are sent to a global one, but the difference is not crucial as one can emulate the other.

CPSUserFolder is a big simplification of the user folder framework, because it delegates actual storage to other frameworks, and concentrates on user authentication. Zope 2's Pluggable Authentication Service (PAS) (or Zope 3's equivalent) provides this kind of abstraction, but CPSUserFolder benefits from the power of directories. However it only deals with storage, not authentication; in the future the two will be bridged.

We don't know of any system equivalent to CPSSkins.

XIII. REFERENCES

[BEC] K. Beck, “Test Driven Development: by Example”, Addison-Wesley Professional, 2002.

[BER] Philip A. Bernstein, "Repositories and Object-Oriented Databases," *ACM SIGMOD Record*, vol. 27, no. 1, march 1998 (http://www.sigmod.org/record/issues/9803/bernstein.ps)

[CMF] http://www.zope.org/Products/CMF/

[CPO] http://www.cps-project.org/

[CPSSkins] http://www.medic.chalmers.se/~jmo/CPS/

[DeltaV] http://www.webdav.org/deltav/

[EDOS] S. Abiteboul, X. Leroy, B. Vroldjak, C. Brice, R. Di Cosmo, S. Fermigier, T. Milo, A. Sagi, Y. Shtossel, S. Laurière, F. Lepied, R. Pop, F. Villard, E. Panto, J.-P. Smets, “EDOS: Environment for the Development and Distribution of Open Source Software,” to appear in *OSS2005*, Genova.

[HOL] S. Holek, “Unit Testing Zope for Fun and Profit”, EuroPython 2002, http://www.zope.org/Members/shh/UnitTestingZope.pdf

[JCR] The Java Content Repository, JCR-170, http://jcp.org/en/jsr/detail?id=170

[Plone] http://www.plone.org/

[RIC] S. Richter, “The Zope 3 Developer's Handbook”, New Riders, 2005.

[Silva] http://www.infrae.com/products/silva
[TCH] A. Tchertchian, “CPSWorkflow: how to set up workflows using CPSWorkflow,” june 2005, available on http://www.cps- project.org/
[WEI] P. von Weitershausen, “Web Component Development with Zope 3”, Springer, 2005.
[Z3lab] http://www.z3lab.org/
[ZB] A. Latteier, M. Pelletier, “The Zope Book”, New Riders, 2001.