The Wageningen Library Content Management System

Abstract

From the beginning of its library automation in 1979, Wageningen University and Research library has a tradition in developing its own local library system. This strategy is not common. Most libraries buy their library systems from commercial vendors. Such a strategy must be reviewed every few years. With the last review a few years back, we decided again for local development when we were going to replace our old system. This time we decided to build a Library Content Management System, integrating a large part of the functionality not covered by traditional Integrated Library Systems. This paper explains our reasons for this and gives an overview of the system's architecture.

[“Good morning. Welcome to you all. We have been busy organizing this conference and its web site using our Library Content Management System and I would like to tell you all about it, but first I would like to tell you a bit about the history of Library automation and the creation of Wageningen Digital library.
Then I would like to explain you why we decided to build a Library Content Management System. I will shortly describe the architecture of the system.
And by showing you some application examples I will illustrate you how this architecture makes it very easy to develop such applications.
I will finish this presentation with some thoughts on Pro's and Con's of our approach.”]

A short history

Library automation in Wageningen started some 100 years after the library was founded back in 1873. On the first of January 1980 the Agralin system became operational. This system, was developed in-house and was based on, at that time, a very sophisticated Textual Relational DBMS called Minisis. It grew into a local Integrated Library System and served us for almost 25 years. For the first 15 years, access was mainly through terminal screens and terminal emulators on PC's.
In 1994 the first graphical web browser, called Mosaic, created at the National Center for Supercomputing Applications in Illinois, became known to the world. This inspired the developers at NCSA to start a company called Netscape. It inspired developers at Wageningen University Library to create WebQuery, enabling a browser to search Minisis data bases and thereby creating one of the first web based library catalogs.

The early web site mainly consisted of one page with a few links of which the major link was the one to the web based library catalog. The growing amount of electronic sources available made is necessary to design a more elaborate structure for our website. We then decided on a basic set of design principles and created the the Agralin Desktop Library
By 2003 we dropped the Agralin acronym that had put us at the top of alphabetic lists for 23 years and traded it in for Wageningen Desktop Library, moving us way down the alphabetic lists.

In 2007, the web site had to be restyled in order to comply with the new Wageningen UR house style. It was also renamed again, this time to Wageningen UR Digital Library *[‘containing all the information mentioned on this sheet’]*. We built Wageningen UR Digital Library using our newly developed Library Content Management System.

Commented [v1]: De naamswijziging was op last van het MT dat dacht dat Desktop niet werd begrepen

Why would one build a Library Content Management System.

Library system vendors are still selling us ILS’s, Integrated Library Systems. An ILS is meant to support a number of traditional library processes (circulation, cataloging. acquisitions, serials administration, etc). A library does so much more nowadays, not supported by an ILS. It may manage an institutional repository, a meta search engine, an openURL resolver, a web site, manage other bibliographic data bases, support full text indexing of documents, etc. Having just an ILS is not enough nowadays.

We think an ILS is conceptually outdated. It will only integrate a limited number of functions, mainly within the context of the environment, defined by the ILS vendor, or even worse, within the context of the specific system, because some vendors sell more than one ILS.

We wanted a component based architecture, more specifically an architecture based.on web services. Library system vendors nowadays feel the same way about this, but only when they are selling you their new products, such as their resolvers, electronic resource management systems, etc. The ILS often remains the closed fortress it has always been.

These thoughts made us decide to buy components or use open source components when they were available and to build them when they were not. For this reason we decided to build a Library Content Management System and to create it using Open Source development tools.

System architecture

[‘The overview I am showing you may shock you and scare you off. I grabbed it from the web and just put it in to wake you up. It has got nothing to do with our architecture, which is much simpler.’]

The core of the system is a program we call WebQuery. I mentioned earlier that this was our tool to enable Web access to Minisis data bases. Over the years WebQuery developed into a more generic application to search, retrieve and update an XML data base.

For historical reasons we do not use a ‘native’ XML data base. We used to work with a Minisis data base and nowadays we store our XML records in an Oracle data base. Our usage of Oracle is very basic and we try to avoid using Oracle tools to remain data base independent. We would prefer to have WebQuery communicate with a XML data base using the XQuery protocol. XQuery is a language to access XML data bases, pretty much like SQL is for traditional relational data bases. However, neither Minisis nor Oracle support XQuery, so we wrote connectors for these data bases. The Oracle connector uses SQL text retrieval, which supports the use of XPath.

To remain data base independent, we do not use data base specific tools to check the internal integrity of the XML records. We trust on validation of XML through XML schema, which is an open standard and for which many open source validation tools are available.

Applications are always web based. Each WebQuery request is a URL. This URL has a structure, quite like the structure used by the SRU protocol. In the URL one can specify that WebQuery has to transform the XML using a specific XSLT style sheet. We could of course let the browser do that, but we prefer to do this server side. XSLT is a standard language to transform XML documents to another representation, most of the time into an HTML page representation.

To modify or insert an XML record in the data base we would like to use XForms. XForms will be the standard to define web based forms. XForms enables you to structure these forms much better and it will pass direct XML to the web server. However, XForms is not yet supported in standard browsers. That's why we now make use of HTML forms of a well defined structure, WebQuery will be able to transform the posted form into the insertion or update of the XML record.

Applications

[‘By now quite a lot of applications have been built using this architecture.
A lot of them are mentioned on this sheet. On the left you will find library components you will find in any library system. On the right you will see applications you may find in web portals but not very often in library systems, like a news service, a module handling all kind of forms, like surveys, suggestions to acquire books or conference registration and a module to handle user comments.’]

It all just URL's
Each request to the system is a URL. It starts with a fixed string to identify it as a URL that WebQuery has to handle. If it is followed, for example, by "/patron/xml/26310" it will show you the XML record containing all Meta data for patron 26310. When we replace ‘xml’ with ‘renew’, WebQuery will transform the returned XML with a style sheet linked to ‘renew’. This style sheet will transform the XML to an HTML page with a form to renew items on loan for this patron.

Another example. This is a request for all subscriptions that we have subscribed to via Swets subscription service, returning a set of XML records with meta data on all these subscriptions (abonnementen in Dutch). When we replace ‘xml’ with ‘subscriptions.csv’, WebQuery will invoke transformation using a simple style sheet to turn the result set into a comma separated file, your browser will probably open Excel or Open Office and you will find the data represented in a spreadsheet
In this last example you see a URL requesting all catalog records, classified on 19, 20 of 21 November. This time a XSLT transformation transforms this XML result set into another XML format according to the RSS 2.0 protocol and we have a RSS feed on the catalog.

One big pile of bibliographic records
The content management system contains all bibliographic descriptions we collect, in XML, in one table within the data base. Descriptions of books, journals, articles, book chapters, web resources, etc. are all in one collection. The library catalog is just a view on this collection, only containing a well defined set. This is actually not true at this moment. Descriptions of publications by Wageningen authors, our institutional repository, form a separate collection for historical and technical reasons, but we will merge them soon. This way a record present in one collection can be easily presented differently in another view, just by using a different style sheet.

One collection, lots of views
By now we have developed lots of views with their own interface and style.
You see some of them appear on the screen. Name a few …
The last 3 views are kind of interesting, since they are 3 professional journals in Dutch. The library manages a bibliography in which it covers a lot of professional journals in the Dutch

language. For some smaller publishers we made a few XSLT style sheets, so they can use a view on the articles from a certain journal as their journal archive.

Portal views
In addition to these ‘administrative views’ that came up in the previous sheet, we can also define subject oriented views and so create subject oriented information portals. Within the Digital Library we have constructed portals for six main subject areas. In these portals we do not only introduce the subject librarians for the selected area, but we also offer links to portal pages on various relevant topics within the area. On the sheet the portal page for Social Sciences is shown.When we select ‘Agricultural economics and Agricultural policy’, we will query the Library Content Management System for all relevant resources on this subject,. On thhe resulting page we will see an index for the grouping of the search results, followed by a meta search box representing our MetaLib link resulting in a search withn a relevant quick set, followed by the relevant resources. All generated from a query on the Library Content Management System.

Other features
A thing you will not see that often in Library Systems is the integration of news. This feature was also built with the Library CMS, as can be seen by the news items on the home page or the calendar items on the same page.
News items can be entered and edited by Library Staff, after which items may appear on several pages within the Digital Library. Using the right URL, referring to a query and a stylesheet, selected items can be presented as a RSS feed or as a monthly newsletter.

Integration
This architecture is not something we invented locally. This architecture has become the standard when it comes to building web services. Many systems are created this way nowadays and it is the main reason why everybody is talking about Service Oriented Architectures nowadays.
Because it is being applied so much, integration with other systems has become so much easier, like for example SFX our OpenURL resolver product from Ex Libris. When I find an article in Science in for example Scopus, Scopus is kind enough to provide our users with our SFX button. Clicking this button will send the Open URL for this publication to our SFX server, which will then show our SFX menu. Very often, this menu will provide a link to the full text of the article on the site of the publisher. The logic to build this link, based on the OpenUrl that is sent to SFX is found in the SFX Knowledge Base, that is maintained by Ex Libris.
This Knowledge Base contains knowledge about many, many sources, but not all. When the source is not a journal but an electronically available monograph, the knowledge base will probably not resolve the OpenUrl into a link to the full text. However, it was relatively simple to make SFX use a web service to look in the Library Content Management System for a link to the full text if the Knowledge base doesn’t have one and so extend the SFX knowledge base with the knowledge that can be found in our own catalog.
We also use web services to show, within the SFX menu, whether we have a paper version and where it is located.
Using another web service SFX can also check in the Library CMS if somebody is logged on to the system and provide personalized services, like saving a reference on your ‘my library’ favorites list or subscribing to a Table of Contents Alert.

Integration with external services

For the same reasons, integration with applications beyond our control has become easy to develop. Within the order administration module, library staff can easily check with the supplier, why an order has not yet delivered and refrain from sending claims. (Unfortunately very few suppliers support this kind of service) We also use web services provided by Amazon when entering new book orders. Once an ISBN for an order is provided we check Amazon for some metadata and merge this into the order form while also adding price information from Amazon and a picture of the cover on the order ingestion form.

Pro’s and Con’s
So what are the Pro’s and Con’s of such a home grown Library Content Management System.

I assume that the advantages of the LCMS are now clear.
Standardization on XML and web services provides the basics for a service oriented architecture, making integration with other components much easier. I think the given examples illustrate this well enough.
Using generic tools, like XSLT, which more and more developers learn about, makes the system easy to develop for. It enables us to do fast and flexible adaptation of the system.
Staying independent of proprietary solutions as much as possible avoids ‘vendor lock in’ and keeps us in control of our own environment.

Are there any disadvantages?

Well, you need to employ software developers. However, I see that other university and national libraries employ software developers as well. They now own a large variety of systems and need developers to make these systems work together one way or another.
We have to defend our decision for local development over and over again. I have done this very often by now and you get a little tired of it. However, it forces you to rethink what you are doing every time and this is a good thing.
Keeping IT staff is important. We try to put effort in documentation and quality control of the development process, but knowledge and experience of people is still very important. We find it also very important for developers to learn about libraries. It is valuable to have developers who know the environment they are working in, although ICT environments tend to look more and more alike. We are happy to say that our IT staff do not run away easily. I think that has got a lot to do with the fact that we are building this exciting environment which people feel a part of and do not separate from easily.

So, as our former soccer star Johan Cruijff, who is famous for explaining simple things in complicated ways, says: “Every disadvantage had its advantages.”

Thank you for your attention