Managing multimedia Content for the Internet

Pascal Dreer
swissinfo/Swiss Radio International

A news and information portal in nine languages ... streaming audio/video over the Internet ... content management systems ... data broadcasting ... data services for mobile phones ... a geographical information system ...

The changes have come thick and fast at swissinfo/Swiss Radio International over the past few years, with the introduction of a host of new services and applications. A traditional shortwave broadcaster has now turned into a multimedia venture, as described in this article.

Introduction / challenges

Swissinfo/Swiss Radio International (SRI) is an enterprise of the Swiss Broadcasting Corporation (SBC) – SRG SSR *idée suisse* [1] – and is mandated to deliver news and information to Swiss expatriates as well as anybody else in the world who is interested in Switzerland. While continuing to broadcast radio programmes in the shortwave bands and by satellite, the Internet has become the main focus of our activities. According to our strategy, the analogue shortwave broadcasting will gradually be reduced until the end of 2004. However, swissinfo/SRI will continue to broadcast radio programmes by satellite (e.g. WorldSpace) and is watching the developments in digital shortwave systems (e.g. Digital Radio Mondiale – DRM).

The news and information portal of swissinfo/SRI – swissinfo.org *(see Fig. 1)* – has been available on the Internet since 1998. One challenge, besides building up the editorial know-how to produce multimedia con-

Abbreviations

A/V	Audio / Video (Visual)	**NAS**	Network-Attached Storage
AMS	Asset Management System	**NFS**	Network File System
CMS	Content Management System	**PDA**	Personal Digital Assistant
CSS	Cascading Style Sheets	**SBC**	Swiss Broadcasting Corporation
CSV	Comma-Separated Values	**SRI**	Swiss Radio International
DB	DataBase	**SSJS**	Server-Side JavaScript
FTP	File Transfer Protocol	**SSR**	*Société Suisse de Radiodiffusion et Télévision*
GUI	Graphical User Interface		
HTML	HyperText Markup Language	**TSR**	*Télévision Suisse Romande*
JSP	Java Server Pages	**WAP**	Wireless Application Protocol
JSTL	Java Standard Tag Library	**WML**	Wireless Markup Language
LDAP	Lightweight Directory Access Protocol	**XML**	Extensible Markup Language

Figure 1
The home pages of www.swissinfo.org *(left)* and www.tsr.ch *(right)*

tent (text, images, audio and video), was to come up with a system that could handle multimedia information to produce attractive-looking web pages.

We developed a system called XOBIX (pronounced "zobix") to achieve those goals. As a measure of our success, XOBIX now produces web sites not only for swissinfo, but also for many of the SBC's domestic radio and TV stations.

Our approach

Back in 1998 there were hardly any Content Management Systems (CMSs) available that met the needs of a media enterprise with fast-changing multimedia content. Most systems only offered management of static web pages. This was not what we were looking for. We wanted to be as flexible as possible and were already thinking about distributing data (content) to mobile devices such as wireless application protocol (WAP) phones or personal digital assistants (PDAs). Another important issue was an "easy-to-use" GUI, since our journalists would feed the system by themselves. Other goals were: (i) the ability to handle multilingual content (even script languages such as Arabic or Japanese), (ii) the automated import of data feeds and (iii) the ability to use only a simple web browser in order to work with the CMS. So we came up with our own specialized content management system, designed for our media needs.

The publishing workflow *(see Fig. 2)* normally starts with the input of content. This can be done either manually (by our journalists) or can be automated (through news agencies). The second step includes putting content together until a multimedia story (or any type of information) is created and scheduled to be online (referred to as content management). Usually the journalist's work stops here.

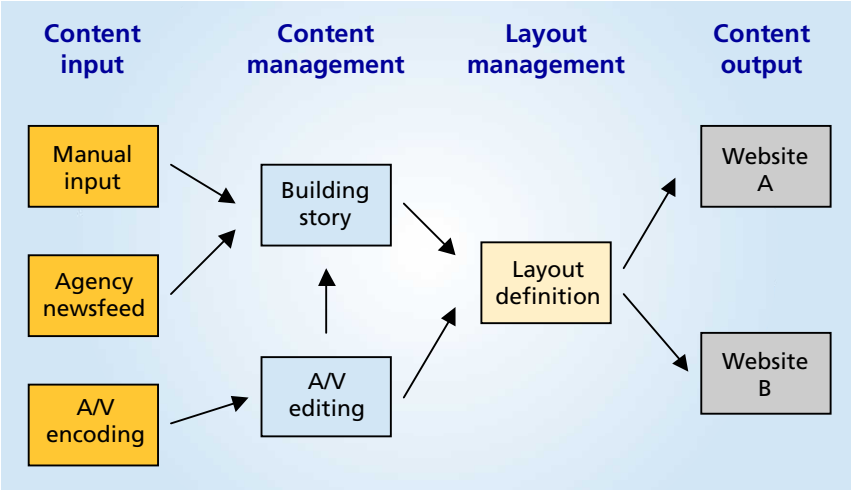

Figure 2
Publishing workflow

The layout definition includes the creation and modification of the website's layout (not the content) and is normally done by web-publishers and web-designers. Once the basic layout has been defined, only minor changes need to be made. However, it is also the web-publishers'

job to set up new pages for special occasions (in-depth features or dossiers) and to include additional services when required. There is one web-publisher for each language version at swissinfo/SRI.

During the last step, the content is usually rendered into HyperText Markup Language (HTML) according to the defined layout (website design). Of course, this step depends on the actual distribution channel and device. In the case of a WAP mobile phone, the content (text only) is rendered into WML instead of HTML. Content can also be exported into XML and other formats.

Separating the content and layout

From the beginning it was quite clear that we had to separate the content and layout in order to pursue our "any device – any distribution channel" philosophy *(see Fig. 3)*. It was also clear that all the content should be stored in a database system and that any output would be generated dynamically (on the fly). However, large pieces of data, such as images, audio and video clips, would be put on dedicated storage servers.

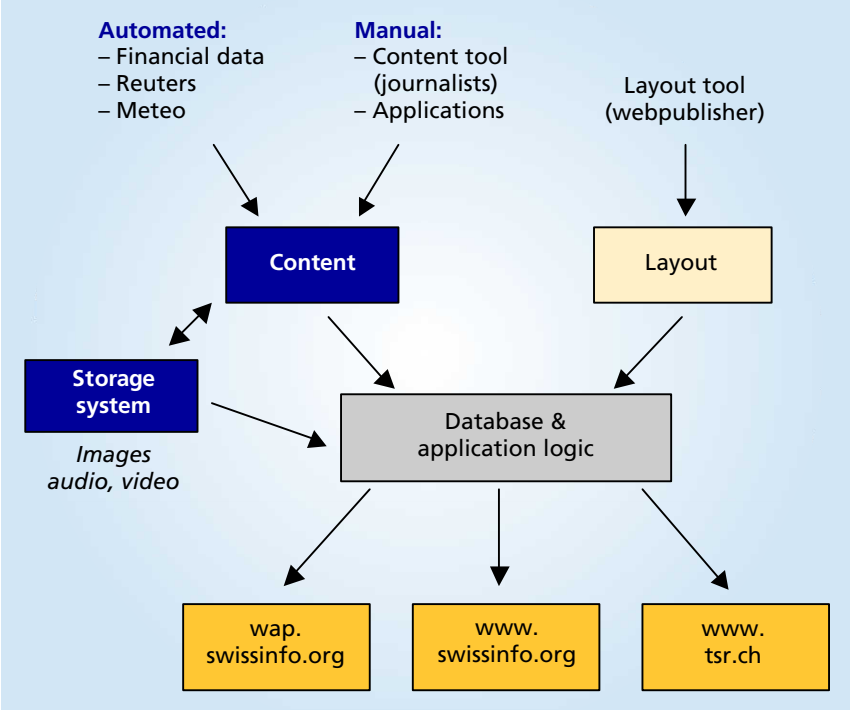

Figure 3
Separated content / layout — multiple sites

Another important requirement was the simple sharing of multimedia elements among different websites. Once data was stored in the database, it could either be used on the Internet or on any other device that allows the information to be displayed correctly. From a conceptual point of view, the database is just a huge container where data is stored in a structured form and is only separated logically. Every piece of data always has an attached source-identification tag. Depending on the definable access rights, applications can use data from a certain source or not.

Dealing with the content

There are two ways to put content into the system. One way is through mostly-automated applications. Data feeds come in various formats such as CSV or XML, or as a serial dataflow. The import module normalizes the data according to our internal data structure. This way we acquire data from news agencies, as well as reports on weather, financial markets, traffic and so on. Usually the data come via FTP (over the Internet) or on serial lines. Automatically-recorded audio and video from radio and TV programmes are imported in a similar way.

Figure 4
Building a story with the Content Tool

In contrast, the content is entered manually. The browser-based Content Tool (*see Fig. 4)* allows jour-

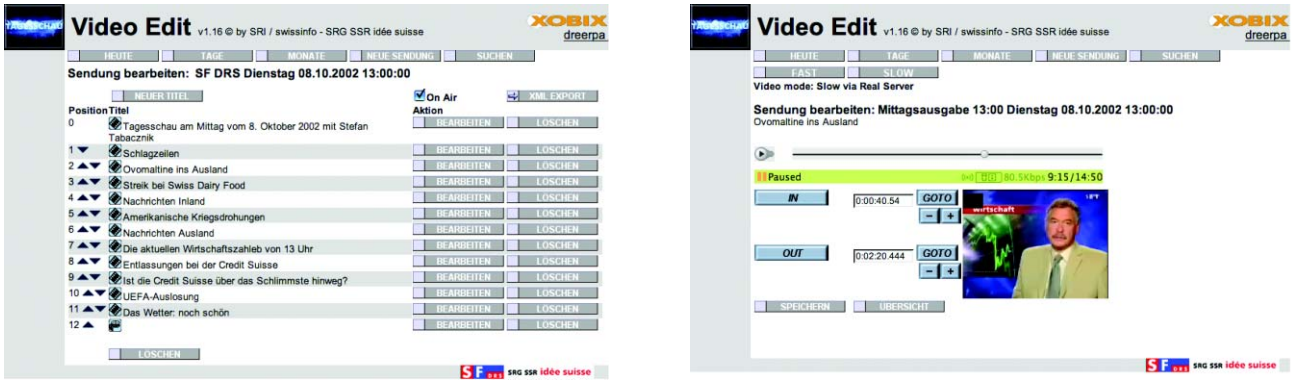

Figure 5
Video editing through a web browser

nalists to upload multimedia elements (images, audio, video) or to browse through automatically-uploaded content feeds. The next task would be to compose a story by typing text and adding "any number" of multimedia elements to that story. Besides images, audio and video, a story can also feature related links, quotes, key facts, etc. Once the story is complete, it can be scheduled to go online in one or many sections and can appear anywhere in the story list according to the priority that is attached to it. A preview function shows a story as it would appear online.

Every tool can be accessed with a personal user account. The access rights to any content (create, read, modify and delete) are individually granted. For example, a user may be allowed to upload images but not to write the stories.

The Video Edit application (which can also be used for audio) is part of the Content Tool and enables segmentation of the video clips *(see Fig. 5)*. Newscasts are normally encoded and stored as complete video clips (currently in RealVideo format only). A few seconds after the encoding is finished, clips will automatically be available in the Video Edit application, where a user can cut the broadcast logically into news segments and label them accordingly.

Building the layout

As already mentioned, the journalist does not have to bother with any HTML coding or with the layout of web pages (or of any other output device). This is done automatically by the rendering application, which applies the layout that was defined with the Layout Tool. Unlike the Content Tool, users of the Layout Tool need to understand HTML, CSS and JSTL in order to get the most out of it.

Typically, a website consists of numerous pages which, in our case, are labelled by unique numbers. *Fig. 6* shows the English version of swissinfo's front page with the number 100. A page is made up of one or several modules. A page with modules can also form a template that can be part of a page again.

The modules are implemented in Java. Users of the Layout Tool can choose from a

Figure 6
Swissinfo's English front page within the Layout Tool

basic set of modules and can customize them through various parameters. For example, a weather module offers a choice of different weather locations, and styles that the information can be shown in. Whenever a new type of data is added to the system, a corresponding module has to be programmed.

However, a web-publisher can also directly access data from the database by using Java Tags (JSTL) and HTML. JSTL [2][3] offers variables, simple conditional logic and loops to output data. We defined about twenty-five JSTL objects which the web-publishers can use to build their own reusable templates. *Table 1* shows the properties of the audio object. To get more information about an audio clip (e.g., what codec, bit-rate), one would have to query the "audio.media", which again returns a "medium" object (the least common denominator of audio, video, or image).

Table 1
Properties of the audio JSTL object

What?	Name	Returns
The audio's unique identifier	audio.id	Text
The audio's object type	audio.objectType	Text: "audio"
The audio's language	audio.language	Object: *language*
The audio's media	audio.media	Collection: *medium*
The audio's title, used for captions	audio.title	Text
The audio's full text, used for a description	audio.fullText	Text
The audio's subjects, which help to put it in context or classify it	audio.subjects	Collection: *subject*
The geographical points where the audio was recorded and to which it refers.	audio.geoPoints	Collection: *geoPoint*
The organization or company that originated the audio	audio.companySource	Object: *source*
The department within "companySource" that originated the audio	audio.departmentSource	Object: *source*
The name of the person who created the audio – the producer, the musician or whoever	audio.author	Object: *person*
A still image that can be used to accompany the audio	audio.image	Object: *image*
The icon used on a web page to show that something is an audio item	audio.icon	Object: *image*

The code fragment on the next page (a part of the story display template) fetches all the audio clips attached to a particular story (if there are any) and displays them with the proper audio icon and clip title as a link *(see Fig. 7)*. The interaction with the database is hidden in the audio-object, of which only the web-publishers need to know the properties.

Panel 1 — HTML/JSTL code fragment to display audio elements

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%--

   This template requires that the parameter "audio" has been set to the audio
   object.
   So before the import, you need to set the following:
   <c:set var="audio" value="${...}" scope="request"/>

   The template takes the following parameters:

   <c:param name="type" value="storyDe-     Show the audio formatted for a
   tail"/>                                  story detail page

   <c:param name="type" value="topStory"/>  Show the audio formatted for a top
                                            story page

   <c:param name="title" value="..."/>      Show another title for this audio

   <c:param name="fullText" value="..."/>   Show a full text paragraph for this
                                            audio


--%>

<%-- Get the icon --%>
<c:forEach var="medium" items="${audio.icon.media}">
  <c:set var="audioIconMedium" value="${medium}"/>
</c:forEach>

<%-- Get the title, defaulting to the audio's title --%>
<c:set var="title" value="${audio.title}"/>
<c:if test="${! empty param.title}">
  <c:set var="title" value="${param.title}"/>
</c:if>

<%-- Get the audio's medium --%>
<c:forEach var="medium" items="${audio.media}">
  <c:set var="audioMedium" value="${medium}"/>
</c:forEach>

<c:choose>
<c:when test="${(param.type == 'storyDetail') || (param.type == 'topStory')}">
  <tr>
    <td valign="top"><a href="javascript:openAudio(<c:out value="${audio.id}"/
    >);">
      <img src="<c:out value="${audioIconMedium.file.url}"/>"
        alt="<c:out value="${audio.icon.title}"/>" border="0"
        width="<c:out value="${audioIconMedium.width}"/>"
        height="<c:out value="${audioIconMedium.height}"/>"></a></td>
    <td valign="top" width="100%">
      <div class="s54">
      <a href="javascript:openAudio(<c:out value="${audio.id}"/>);"><c:out
        value="${audio.title}" escapeXml="false"/></a>
      </div>
    </td>
  </tr>
</c:when>
</c:choose>
```

Fig. 7 shows (underneath the title "Related Items") two links to audio clips which were produced by the HTML / JSTL code above.

System architecture

For simplicity, the name XOBIX is commonly used for all CMS-related tools developed at swissinfo/SRI. The first system configuration was built around an Informix database and Netscape's Enterprise web-server with Server-Side Java-Script (SSJS) technology. Unfortunately, SSJS never really became popular and eventually vanished.

Some parts of XOBIX, notably the Content Tool, are still running with SSJS. Most other parts were migrated to a common combination of Apache web-server and the Java-based Jakarta/Tomcat application-server [4]. Due to some performance and stability problems with the Informix database, we recently decided to undertake a migration to Oracle. Most applications run on Sun Enterprise servers using Sun's Solaris operating system.

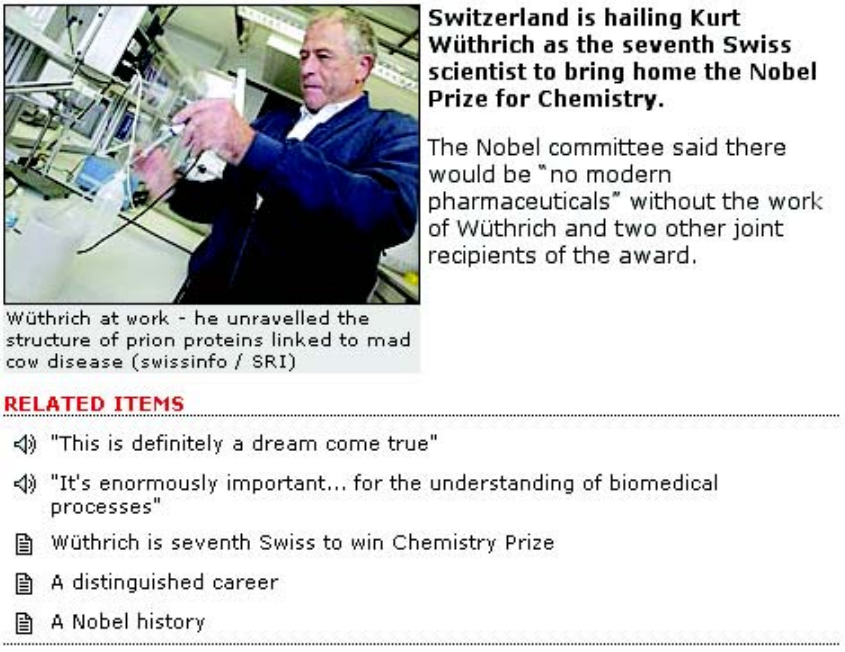

Figure 7
Actual output produced by the HTML/JSTL code

Since 1998, the server infrastructure has grown from three systems to dozens of servers in order to cope with the steadily-increasing performance and availability. However, a lot of attention was also paid to integrating the existing applications (e.g., our Dalet Newsroom system). *Fig. 8* illustrates a simplified view of the various components that make up the entire XOBIX system.

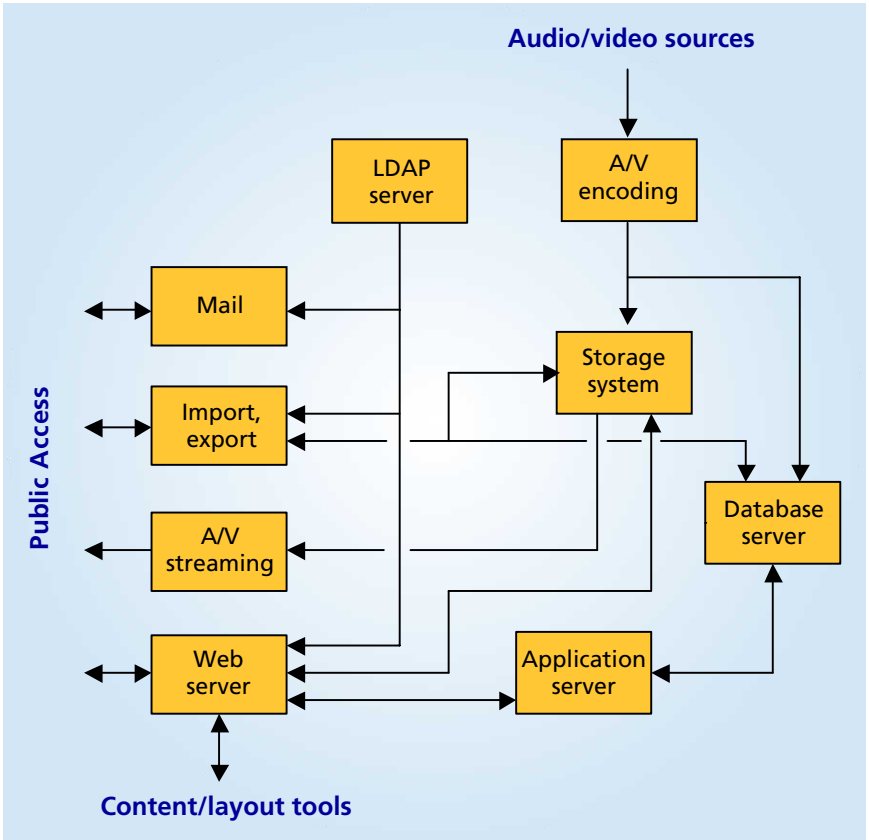

Figure 8
Simplified XOBIX architecture

Besides the database, web and application servers, the storage system also plays a crucial role as a centralized repository for all media content (images, audio and video). An EMC NAS system was chosen that shares its data via NFS with all the other servers.

Another convenient feature for XOBIX users (whether journalists or web-publishers) is the Directory Server (LDAP). The LDAP system automatically synchronizes (one-way only) all Microsoft Windows accounts to XOBIX. Therefore, users have to remember only one username / password for both systems.

Some statistics regarding the XOBIX system are given in *Table 2*.

Table 2
System figures (as of October 2002)

Number of stored story and text elements (DB)	> 438'000
Number of stored video clips (DB)	> 83'000
Number of stored audio clips (DB)	> 32'000
Amount of audio- and video clips (storage)	> 600 GB
Daily inflow of new audio clips (storage)	> 150 MB
Daily inflow of new video clips (storage)	> 450 MB
Daily number of new incoming multimedia elements	> 1000

Conclusions and outlook

XOBIX is a product resulting from swissinfo/SRI's need to develop a comprehensive publishing system for its own online activities. The goal was to come up with a system that suits the needs of a multimedia-oriented broadcaster that deals with text, images, audio and video. During the past three years, the system has found its way to several radio and TV studios within the SBC group.

These days, websites are also supposed to be up and running 7 x 24h without any interruption. Therefore, we put some emphasis on the availability issue by increasing the redundancy and performance. Load balancing systems (layer 4-7 switches) and cache engines (reverse proxy servers) are currently installed to deal with that problem.

We are continually extending the system by adding new functionality. Now in the pipeline is a personalization module, so users can easily build their own, personal, news web page. With a browser-based image-editing tool, users can cut their images directly in XOBIX, and so we can save on Adobe Photoshop installations. Another important issue is the better workflow support. A comprehensive mechanism for tracking the elements, until they finally show up on a user's web browser, has not yet been implemented.

Pascal Dreer graduated in 1995 with an MSc in Industrial and Systems Engineering from Ohio University in Athens, USA. He also holds a BSc in Computer Science from the College of Engineering HTL in Berne, Switzerland. Since 1996, he has been with the SRG SSR *idée suisse*, currently as Head of Information Technology at swissinfo/Swiss Radio International.

At swissinfo/Swiss Radio International, he developed and implemented its first web-based CMS (named XOBIX) in 1998. Besides being used in other SRG SSR web projects, XOBIX's architecture was built around the multilingual news and information portal swissinfo.org, which was launched in 1999.

Contact: pascal.dreer@swissinfo.ch

Bibliography

[1] swissinfo/SRI Fact Sheet
http://www.srg-ssr.ch/en/radio/sri/en_srifact.html

[2] Shawn Bayern: **JSTL in Action**
Manning Pubns Co, 2002.
http://www.jstlbook.com

[3] JSTL reference website
http://java.sun.com/products/jsp/jstl/

[4] Jakarta/Tomcat
http://jakarta.apache.org