XML-Based Dynamic Content Generation and Conversion for the Multimedia Home Platform

Uwe Zdun
Institute for Computer Science
University of Essen, Germany
zdun@acm.org

ABSTRACT: Many systems gather content from multiple input sources and provide it to multiple output channels. Usually content has to be (partly) generated, and content has to be converted to different formats. In this paper, we will discuss the domain of digital content broadcasting with the Multimedia Home Platform (MHP) as a case study domain. However, there are other domains that require content generation and conversion as well, such as web engineering and content management. As a solution, we will present a generic XML-based architecture for dynamic content generation and conversion. It provides content converters for multiple input and output formats. Content format templates, fragments, and content format builders are alternatives for dynamic content generation. Page templates are used to impose common styles and portal layouts for interdependent content. A Service Abstraction Layer supports service-based integration of different new media platforms.

I. Introduction

Many interactive applications have to generate formatted content on request, and/or convert given content to others formats and styles. The content is usually not or only partly available in pre-built files of the target format. Generated content often has to be formatted in different markup languages, such as HTML, WML, and XML. In many cases, other formats, such as graphical user interfaces and textual representations, have to be supported as well. Moreover, the content usually has to be provided to different channels with different protocols, such as HTTP, WAP, UMTS, MMS, etc.

In our experience these issues are recurring in different contexts in which content has to be flexibly gathered and provided for different platforms, as for instance interactive digital broadcasting, web publishing, and content management. In this paper, we will discuss an architecture for dynamic content generation and conversion based on XML technologies.

As a case study we present an industry project with the goal to define a generic product line architecture for the content provider BetaBusinessTV on top of the Multimedia Home Platform (MHP) [5]. The MHP is a standard, client-side environment, based on the Java programming language, for interactive digital broadcast applications.

All the problems of content generation and conversion, as named above, apply for a MHP product line: the content presented on a MHP terminal usually has to be provided to other channels as well, such as the web or mobile devices. These (may) use different formats and styles than a MHP application. The MHP itself supports (and will support) different content formats as well, including compiled Java application classes, XHTML pages, ECMA scripts, etc. That means content created in multiple input formats has to be converted to the target formats. Usually, content is partly pre-built, and partly it has to be generated on request, say, on basis of data received via a return channel. The pre-built content may have to be dynamically converted on server-side.

In this paper, we will first explain the MHP domain and present a reference architecture on the server side in which our content generation and conversion architecture will be integrated. Then we will identify a set of recurring problems in the domain of generating and converting content dynamically. As a solution, we present a dynamic content generation and conversion architecture based on XML technologies. Finally, we will analyze the impact of the architecture, and discuss related work.

II. Case Study: Multimedia Home Platform Product Line

As a case study domain, we will discuss content presentation, generation, and conversion issues in a generic product line architecture for the MHP. In this section, at first, we will briefly explain the MHP domain. Secondly, we will discuss the issue of dynamically generating and converting content in this domain. Thirdly, we will outline some problems and open issues tackled by our work.

A. Multimedia Home Platform (MHP)

In 1998, the Digital Video Broadcasting Project (DVB) embarked on an ambitious project to produce a European platform for the converging multimedia services of the future. The DVB-MHP specification [5] aims at the deployment of an open standard API that will facilitate seamless services across broadcast, telecommunication, and computer platforms. The MHP standard defines a client-side software layer that runs on a MHP terminal. Possible MHP

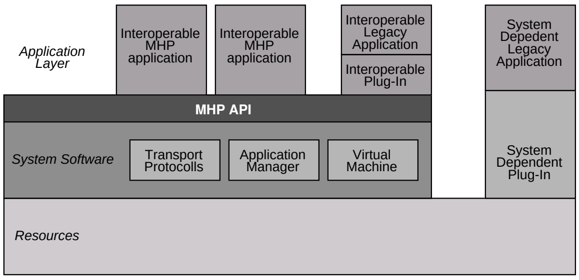

Fig. 1. MHP Basic Client-Side Architecture

terminals are digital set-top boxes (STB), integrated digital TV sets (IDTV), and multimedia PCs. The MHP standard provides specifications and APIs for MHP terminal implementations, including:

- *Platform Architecture:* The client-side architecture depicted in Figure 1 is defined in detail. It consists of three basic layers: a resource layer comprising the hardware and software resources provided by the MHP terminal (such as a settop-box), a system software layer for reliable control of the platform and application management, and an application layer running interoperable MHP applications such as Java TV applications (XLets) and HTML applications.
- *DVB-Java Platform on Top of a Java Virtual Machine:* The MHP is based on an embedded Java virtual machine implementation running DVB-J applications. It includes most standard Java APIs, and a set of custom APIs for user interfaces, security, TV APIs, etc.
- *Transport Protocols:* A set of communication channels is specified, including broadcast channel protocols (such as MPEG-2 video and data transmission via a DSM-CC object carousel), and interaction channel protocols (such as TCP, HTTP, UDP, and DSM-CC User-to-User).
- *Content Formats:* There is a determined set of required content formats to be supported by a MHP client. These are static bitmap formats, like JPEG, PNG, and GIF. Other static formats are MPEG-2 frames, video drips, MPEG-1 audio clips, and UTF-8 format. The MHP client has to support at least a resident font, but portable downloadable font resources are also supported.
- *Application Model and Lifetime Signaling:* A DVB-J application has four different states: loaded, paused, active, and destroyed. The application manager has to manage and signal lifetime events, and handle resource allocation.
- *Security:* There are several security means specified, including conditional access to control the management of a set of authentication keys used to descramble or decrypt downstream video or data streams, secure communication over the return channel with SSL, and the bytecode verification of the Java VM.
- *Graphics and GUI*: The HAVi graphics reference model and a few extra specifications for the MHP are supported. The model includes a background screen (backgrounds are still images), a graphics screen, and a video screen. On top of the graphics model the HAVi Level 2 UI is used as User Interface API for the MHP. The HAVi level 2 UI is based upon the Java AWT 1.1 lightweight widget framework. But only a subset of the AWT is required since other parts of the AWT widget set are not considered as "TV-friendly."

We believe that software development for the MHP will differ from current embedded or PC platforms, since the MHP cross-cuts these domains and the television broadcast world. Moreover, MHP applications need to be integrated with applications and content from several different sources. E.g., the content has also to be provided to other platforms, like the web, mobile devices, or teletext. Applications have to be integrated with broadcaster applications, existing legacy applications, and other similar platforms, such as web applications.

B. Dynamic Content Creation in an Integrated Broadcast Architecture

Our project was concerned with the definition of a generic MHP product line architecture for the content provider BetaBusinessTV. The company mainly aims at business TV customers in the financial and insurance business sectors. The customers usually want to use the business TV platform for rapidly informing their employees with financial news, stock prices, interest rates, etc. Other channels, such as mobile phones and the web, have to be integrated into the framework.

The MHP specification mainly defines a client-side standard. However, many essential ingredients of a product line architecture have to be defined on server-side. In the course of the project we have defined a server-side reference architecture which takes the full content production chain and user interaction into account.

We propose to understand the broadcasting process as the reference architecture depicted in Figure 2. The content provider authors or supplies the content. It is important that there is an integrated software support, so that technical non-experts – at the knowledge level of today's web publishers – are able to easily publish content in a suitable appearance. For instance, in many MHEG-5 environments content is still authored solely with text editors. This is not very appealing for the average content author. The content provider controls the content that is to be broadcasted. These issues are handled by content authoring tools. But the content provider also has to be able to control when to broadcast the content. Therefore, the broadcaster's content management system has to build runtime logs to ensure correctness. To a certain degree the content provider also has to have control on how to broadcast, e.g. to optimize the order of broadcasting content chunks, say, to minimize startup times of interactive applications.

Content providers may be small, medium, or large size companies, that have for various reasons an interest in publishing content via a digital broadcast. An important aspect is that many content providers may share a broadcaster's network and that they may operate remotely from the broadcaster. In the broadcaster's architecture a central media management handles all audio, video, and data content. The data is sent to the multiplexer. Content and media management also handles the synchronization with the return channel. This information may be provided to the content provider. There can also be a direct return channel connection to the content provider.

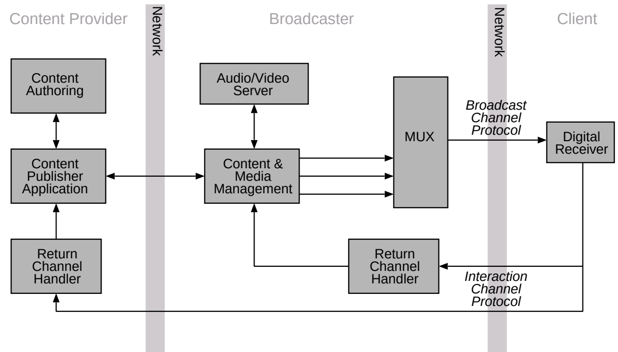

Fig. 2. Intergrated Broadcast Architecture

C. Dynamic Content Generation and Conversion: Problems and Open Issues

In this paper we will focus on a specific part of the integrated content broadcasting architecture sketched above: the generation and conversion of content on the content provider's side. Here, the content may be given in different input formats, and the output formats accepted by the broadcast network (and thus the MHP terminals) have to be generated from these inputs. Usually, other channels such as the web have to be supported as well. In this context there is a set of recurring problems. Some of these problem are unique to the interactive digital TV domain, some problems are general problems of dynamic content generation and conversion. Typical general problems are:

- *Incoming Content Format Conversion:* Incoming content usually comes in a variety of different formats that may differ from the formats and styles in which the content has to be represented on the MHP terminal. Given that there are n input formats and m target formats, $n*m$ content converters have to be written.
- *Multiple Target Formats, Styles, and Channels:* In the MHP context, there are multiple target formats, including compiled Java application classes, XHTML pages, ECMA scripts, and others. Moreover, often content has to be presented to other channels than the broadcast channel as well. Typical other channels are the web and mobile devices such as mobile phones and PDAs. Those usually have quite different representational characteristics. E.g. on the web usually smaller fonts can be used and more information can be presented on a page than on a TV set; on a cellular phone less information can be presented within one page. Typical TV consumer behavior is different to PC user behavior.
- *Abstraction from Content Format Specifics:* When content has to be generated on-the-fly it is undesirable to code the content creation logic for each content format on its own. Ideally there would be some abstraction from specifics of different content formats so that the same business logic can be used to build content in different formats.
- *Non-Programmers as Content Creators:* In the digital TV domain usually there is a lack of qualified programmers, but not of content creators at the knowledge level of today's web publishers. Therefore, it is undesirable that for providing new content always Java classes have to be written, but there should be simpler (e.g. graphical) tools.

The interactive digital television also has a few unique characteristics that distinguish the domain from similar PC-based platforms, such as the web. Those have to be considered for a content generation and conversion architecture as well:

- *Resource Limitations:* Settop-boxes typically have more limited hardware resources than PC platforms, and a broader variety of different hardware platforms has to be supported. In the TV market, older appliances have to be supported for a longer period of time, and the market saturation with high-end boxes is usually low. Thus, the resource model of MHP compliant set-top boxes or personal workstations limits the design freedom.
- *Broadcast Channel Flexibility:* The broadcast channel provides a very high bandwidth, however, it is quite inflexible in its handling. First, the data has to be sent to the broadcaster, and a DSM-CC object carousels has to be built up before the data can be sent out. This implies much higher reaction times than for instance on the web. Moreover, it is rather impractical to broadcast personalized content for each user. For such tasks, the return channel can be used. The content also has to be prepared in well-sized chunks and a suitable order so that the application and its data can be downloaded in one turn of the object carousel. Otherwise long loading times are the result.
- *Return Channel Bandwidth:* Compared to the broadcast capacity, the communication bandwidth of the auxiliary return channel is rather low.

As such, even though we concentrate on the server-side architecture, our work also addresses the design spaces given by the unique constraints of the MHP standard and MHP compliant set-top boxes in the context of dynamic content generation and conversion.

III. XML-BASED ARCHITECTURE FOR DYNAMIC CONTENT GENERATION AND CONVERSION

In this section, at first, we will present the idea to use XML as a generic data glue in a content conversion architecture. Then we will discuss an architecture for publishing and gathering content together with the broadcast architecture presented, and we discuss XML-based content conversion in this context.

A. XML as a Generic and Flexible Data Glue

In a many interactive applications different forms of content have to be provided. Usually content to be presented on the web can be given in different formats, different legacy APIs, and over different channels. The code for conversion to and from different formats should be reusable, and the number of conversion should be minimal. Often different programming languages and programs should be able to access the same information base. Ideally, we would be able to access content that is provided in several different formats on different channels in a generic, flexible, and reusable way.

Scripting languages are often called "glueing languages" because they serve as a behavioral glue to compose components. In this section we present the idea to use XML as a generic data format that serves as a *data glue*. Of course, we can also choose other generic (standardized) content representation formats that fulfill the requirements of the application, such as XSLT, DOM, RDF, etc. The basic idea is to provide one intermediate representation for all data – in the same way as scripts glue components – to glue the data of applications requiring different formats.

In this context, we have to define a task-specific language on top of XML that defines the presentational objects in a content provider's domain. Usually, we define hierarchical content structure with primitive and compound content elements. E.g. in the financial news example, the news of a new CEO for a company can be composed as a composite stock news with title and picture of the new CEO in the header, a formated text fragment that is loaded from the content cache, and links to the data sheet of the stock item in the footer.

```
<stockNews stockID="851399">
  <header type="verbose">
    <title> A. Nobody employed as new CEO</title>
    <picture docID="jpgs/anobody.jpg">
      A. Nobody
    </picture>
  </header>
  <formatedText docID="texts/anobody1.xml"/>
  <footer>
    <seeAlso docID="stocks/851399/dataSheet.xml">
      Current stock price and data sheet
    </seeAlso
  </footer>
</productPage>
```

The Composite structure provides an information architecture that follows the structure of the XML files. In Figure 3 an excerpt of the information architecture for stock news pages is shown. If the parser finds a new XML node, the responsible class is instantiated, and the new object handles the arguments and inner nodes. Each compound node class has a set of constraints that determine which inner nodes can be accepted. For instance, a "watch list" can have a "see also" link to a "stock news" page but cannot have the "stock news" page itself as a list child. If the constraint is violated, a central content gatherer (see next section) returns an error message that can be displayed by the content editor's tool. The xoXML and xoRDF parsers, discussed in [12], provide an automated architecture for building up such object trees with constraints from XML files.

Usually, we would integrate the generic, domain-specific content format in XML with an information architecture following the XML hierarchy. That is, a Composite [6] structure is built to model the information architecture required to support XML elements as classes. Here, we define compound classes for stock news, header, and footer, and leaf classes for title, formated text, picture, and see also links.

Using XML as a generic content format has the advantage that XML allows for integrating content from heterogeneous sources. It reduces the necessary number of converters from $N * M$ converters to N input format converters plus M target format converters. Conversions can be applied automatically. XML is a basics to implement an efficient content conversion and generation architecture in the context of multiple formats and channels, since fragments can be created and cached. Content editors can edit and understand the XML files, and thus, they can define page constructions with a simple format and without programming. Graphical tools can easily be written. By adding new classes content formats can be customized to new domains. This is very important for a business TV content provider who has to customize applications rapidly for different customers.

However, there are also a few liabilities in using XML for content integration. The format and its information architecture have to be defined centrally, thus, as applications evolve, it may be hard to evolve the XML format non-centrally (in a distributed and collaborative working environment). Therefore, initial formats have to be well designed for the particular domain, and extension processes have to be defined. Conversion can mean to loose information if the expression power of other supported formats and the XML format vary. For unknown documents it may be hard to guess automatically which parts of the XML format conform to which part of the unknown document.

B. Integration with the Broadcast Architecture

In the previous section we have proposed to generically define the content with XML and to integrate it with the information architecture by using Composite classes. As a next step, we have to integrate this business logic architecture with the broadcast architecture from Section II-B. That is, we have to discuss how content is stored and managed, how necessary conversion can be triggered by the broadcast architecture, and how other channels can be integrated. For these tasks we require central instances that handle central content publishing and gathering tasks, content caching,

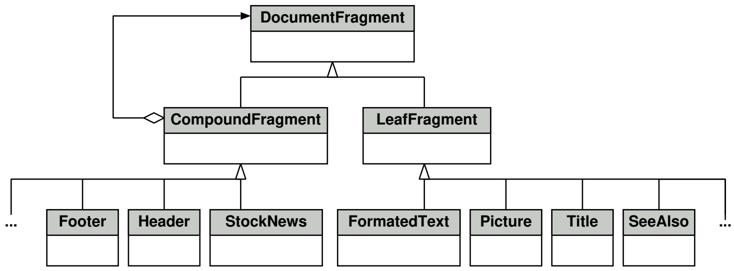

Fig. 3. Composite Structure for the Stock Information Example

conversion to the generic XML format, and conversion from the generic XML format to target formats.

In Figure 4 we see a schematic overview of content management and conversion on basis of XML. A central content publisher can be accessed by different content management and creation tools. The gatherer accepts content in the generic XML format, or content in any other formats for which a converter to the generic XML format is available. Then conversion is triggered, and the content is stored in the content cache. A central content publisher allows for retrieving content. It may be triggered by different clients, including the broadcast architecture that builds up the DSM-CC object carousel, and thus, delivers content to MHP clients. Other platforms are also integrated such as web or mobile device clients. Again, content conversion may have to be triggered.

Often content creation and conversion may be triggered without a request as well. For instance, the cache can pre-calculate certain elements according to certain caching rules. Content change detection and propagation have to find and invalidate cached elements that are not valid anymore.

Besides conversion, we also dynamically create (parts of the) content. Here, we will discuss three alternatives: fragments, content format templates, and constructive approaches. To impose common styles, portal layouts, and other customizations of interdependent content, page templates can be used. The parts of an integrated architecture for content creation and conversion are depicted in Figure 4.

C. Dynamic Content Conversion

The architecture based on publisher and gatherer does not rely on certain conversion technologies or concepts. There are different alternatives for XML-based conversion. In general, at first, we have to convert input formats to XML (some input formats such as images may not be converted but stored in the input format). The gatherer allows us to access the documents in the generic data glue format. In three steps other formats may be generated on request from the XML representation stored in the content cache: XML input processing, conversions, and XML output processing. The result may be stored in the content cache again (e.g. after change propagation and update), or the result is directly delivered in the target format.

XML input processing means to parse and validate the XML text. After parsing, the relevant information has to be recognized and/or searched in the XML document. Once the information is located it can be extracted and connected to the business logic. After performing conversions using the business logic, or with the internal (e.g. DOM tree based) representation, or by applying XSLT style sheets, we can either recreate XML text or other target formats during XML output processing.

Given that we use XML as a generic representation format, we have to convert the input formats to XML, and XML documents to the target formats. Of course, some target formats may also be input formats, and XML may also be input or target formats. For some conversions input and target format may also be identical, say, if a XML document has to be convert from one DTD to another one. E.g. in the financial sector there are different content management tools producing different XML output, and there are multiple proprietary formats. The XML information architecture provides a common ontology, and on the content converters it is defined how to map each individual input format ontology to the common one. Of course, there may be lossy conversions and other related problems that have to be fixed. In the content provider business, a possible solution is to provide semi-automatic converters that "guess" an ontology mapping, and learn from content editor corrections.

In the XML context, in general, there are three different models to handle XML processing:

- *Event-Based Processing:* SAX [10] is a simple API for event-based parsing of XML text. Expat [2] is a XML parser that provides another event-based model. In general event-based parsers produce a flow of events from a given XML text (like start, end, data of a XML node). Usually, the APIs are relatively simple, and the memory usage is low. The basic idea is to catch the relevant events as they are processed. Therefore event-based processing is especially well-suited when the target information has to be accessed only once.
- *Tree-Based Processing:* DOM [15] and xoRDF [12] are models that create a tree-based representation for XML (and RDF respectively) in memory. Therefore, the document can easily be searched and queried using a tree traversal API

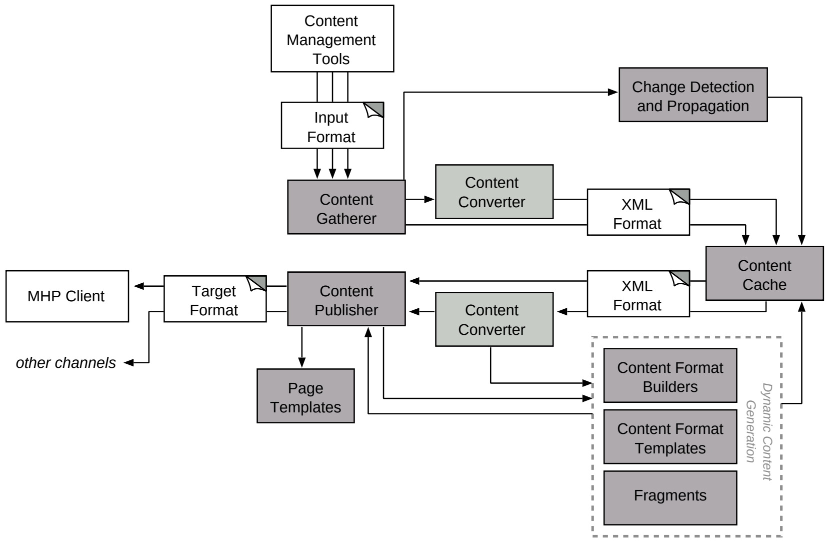

Fig. 4. Integrated Content Publishing and Gathering with XML

for random access in memory. DOM parsing usually suffers from high memory usage (about 2-3 and more times of the document size depending on the DOM implementation).

- *Rule-Based Processing:* In XSLT [3] rules are given that are to be applied when specified patterns are found in the source document (which is XML text). These patterns are specified using the Xpath language. Xpath is used to locate and extract information from the source document. While event- and tree-based processing require the developer to write a program for information extraction, XSLT mostly requires writing style sheets which are themselves XML documents. Thus non-programmers are (to a certain degree) capable of writing these rules. However, the non-imperative model of XSLT is often not well-accepted by developers used to imperative programming models.

In Table I the three processing models are compared regarding their capabilities during XML input processing. As discussed in the previous section, XML output processing is required as well in a generic content conversion architecture. The capabilities during XML output processing are compared in Table II.

In the MHP context, all three models may be relevant, and therefore, often they have to be combined. Event-based manipulation is especially well-suited for low-level tasks, such as integration with dynamic content generation (see next section). The MHP provides interactive applications, and in almost any interactive application setting dynamic content generation is a requirement. Thus, event-based parsing is important for the developers of the product line architecture. For direct manipulation of the content tree-based and rule-based content manipulation are well-suited. Tree-based parsing is often used for visualization of the content structure, and rules are used for high-level customizations that are independent of program code. Often developers used to imperative languages prefer the tree-based model, whereas the rule-based model can be used for direct manipulation by non-programmers.

Many XML processing architectures use a scripting engine as well. To a certain degree a XSLT engine can be seen as a scripting engine, however, for many behavioral modifications an imperative scripting language is usually better suited and more accepted by developers. A scripting engine can be used with all processing models to ease API access, glue different components, and enable rapid behavioral customizations.

IV. Dynamic Content Generation

In the MHP context, often the same content has to be processed for one of the MHP formats and for formats of other platforms as well. Sometimes the same format has to be supported in different variants, say one XHTML page processed for display on a TV set and one HTML page for display in a web browser. The content target format has to be generated from the generic XML representation on request. Sometimes parts of this generation have to be performed on-the-fly because the content contains dynamically changing elements, other parts can be cached. Ideally we would be able to abstract from specifics of the content format in the business logic code so that an application programmer and content creator has not to be aware of these specifics.

To resolve these problems we basically have two different choices: we can abstract from content formats by using only an abstract interface of content elements, and *construct* a generic representation using these elements. As an alternative we can use a *template-based* approach; that is, we can integrate meta-information in the XML content representation that are replaced with dynamic content later on, or we can provide different fragments that are composed to complex pages. In the MHP context, a constructive, pro-

	Event-Based	*Tree-Based*	*Rule-Based*
Parsing Model	Generating events (node start, end, and data)	Building up a node tree (using event-based processing)	Using tree and event-based processing
Extracting content	Event catching	Tree Traversal API	Getting content with XPath statements
Searching content	Event catching	API for content extraction	Getting content with XPath statements

TABLE I
XML INPUT PROCESSING: COMPARISON OF THE PROCESSING MODELS

	Event-Based	*Tree-Based*	*Rule-Based*
Creating the Processing Representation	Generating events as method calls (hand-built)	Factory methods of the tree	XSL statements
(Re-)creating XML text	Customizing event handler methods (hand-built)	Custom tree traversal class (hand-built)	XSL output method statement

TABLE II
XML OUTPUT PROCESSING: COMPARISON OF THE PROCESSING MODELS

grammatic approach is usually used by developers of the product line and for building user tools. Templates can be used by content editors for specify dynamic content without programming. Fragments have to be combined with both approaches to build pages efficiently from smaller building blocks, and for integration with content caching. The issues discussed in this chapter are discussed in more detail in [16].

A. Constructive Approach: Content Format Builder

To construct a variety of different content formats with one generic interface, we can provide an abstract content format builder class that determines the common denominator of the used interfaces. Then we build special classes that implement this interface for each supported content format, as well as special methods (e.g. callbacks) for required specialties. For instance, we can derive one content format builder for HTML on the web, one for XHTML on the TV set, one for a Java-based Havi Level 2 User Interface according to the MHP specification, etc. Thus, in this example, Java code is generated that represents the same user interface as provided by the HTML forms in the two HTML representations.

The content format builder classes' instances enable the application to incrementally build up pages according to the content format. For each page a result can be retrieved as well. Usually we would provide different abstractions, such as menus, forms, lists, etc., that can be supported by each specific content format. Of course, some elements provided by one content format have to be ignored by others. This way limited content formats, say, for a WAP platform or other mobile devices, can be supported as well. As a drawback, to a certain degree we limit the user interfaces to the common denominator of the involved formats, or we have to program more advanced functionality for the less advanced formats by hand.

Usually for each content format element we have methods for starting and ending the element, so that elements may be placed in between. The content format builder is either a Composite [6] object that is built up incrementally, or a hierarchally structured list. Content format builders generically build up content formats; thus, they are a generic constructive approach. In contrast, content format templates are a generic template-based approach for the same problem.

B. Template-Based Approach: Content Format Templates

As an alternative, we can provide meta-information regarding the user interface construction in the XML files. These XML files are handled as templates to be converted by a template engine into the respective content formats. The content is enriched with user interface meta-information. Thus a little language has to be defined for specifying the substitutions to be performed by the template engine. In some variants this is a whole scripting language. Popular examples of template languages in the web context are PHP, ASP, and JSP. In the MHP context, often a domain-specific template language has to be defined which integrates all supported new media platforms.

The substitution elements are usually independently computed so that it is hard to reuse recurring elements without changing the language, and concerns cutting across different elements are hard to express as well. In most cases a content format template is faster than a content format builder performing the same task.

C. Template-Based Approach: Fragments

Generating web pages from dynamic content is costly in terms of memory and performance as content has to be fetched from databases and has to be dynamically built. Both, content format builders and templates therefore may cause performance problems. Moreover, in typical MHP applications interactive content is composed with static elements in the same pages. That means, a whole content page is not a suitable building block of interactive applications with high hit rates. For interactive applications that are presented to a broader audience using multiple channels, as it is usual in the business TV context, high hit rates can be expected. For acceptance of the service among customers, quick response times are very important. Moreover, inconsistencies of pages have to be avoided.

Therefore, the information architecture should concentrate on fragments of pages which can be assembled in different ways, either as raw content, by a content converter, by a content format builder, or by other fragments. The hierarchical nature of fragments indicates that we should use the classes represented in the generic XML format as fragments of the information architecture. Each fragment has to have a unique document ID, and can be stored in the cache. A component for content change detection and propagation has to invalidate cache elements that are inconsistent. The gatherer has to trigger content change detection and propagation.

D. Page Templates

Often content has to be represented in a common style; e.g. logos, surrounding frames, style sheets, etc. should be provided. Some required content changes are not solely expressible by Cascading Style Sheets (CSS), but require behavioral specifications of the changes to be performed. Often a business TV customer has multiple different portals, and each portal should appear in a different layout. Nonetheless, the same information architecture should be reused for all portals. Content editors should be able to make changes to the portal layout without programming.

A *page template class* [16] has a repository of methods for imposing customizations on pages during creation of the target format. Page template methods are triggered by the publisher, and their results may be stored in the content cache. Page template classes compose the aspects that are cutting across a portal's pages in one computational entity. Styles and layouts are only one typical customization of interdependent content. Others are, for instance, channel-specific customizations, specifics for different content formats, and broadcast sequence order.

Each individual portal is dynamically customized with a simple XML file that is also stored in the content cache. This way the overall portal appearance can be dynamically changed by content editors without programming. The system has to automatically re-parse the XML files upon changes and propagate inconsistencies. In the design depicted in Figure 5 we can see that portal layouts, styles, and other customizations are handled by a list of page templates. Those decorate the top-level document fragment that represents the whole page. The portal page template aggregates all relevant parts. Upon a request the proper parts for the page are selected and assembled. Styles are more simple, and can be implemented by a single object.

This generative architecture on server-side creates static Java classes, web page forms and/or applets, MMS pages, etc. In the MHP context, for many requests it is unknown until runtime which format or template has to be supported. Dynamic changeability is also important because usually servers cannot be stopped for changes. End-user customization can be enhanced by defining the page template characteristics in separate XML files that are dynamically reloaded upon changes.

V. Supporting Multiple Channels

Often different requests coming from different clients, communicating over different channels, have to be supported. It should not be required to change the business logic every time a new channel has to be supported or a new service is added to the application. In many business systems, this problem is solved by a Service Abstraction Layer architecture [14]. A service abstraction layer is an extra layer to the business tier containing the logic to receive and delegate requests. Service Abstraction Layer abstracts over different service providers. The Service Abstraction Layer, in turn, implements different channel adapters to support calls via different protocols.

When pages have to be generated dynamically, the Service Abstraction Layer may serve for more purposes: it may be a Message Redirector [7] for indirecting symbolic calls sent by the server to actual implementations of appropriate document selection, conversion, creation, and delivering in the target format. In Figure 6 the integration of Service Abstraction Layer with the broadcast architecture is depicted. Content is either generated by services connected to backends, or provided by content creation/management systems. In the Service Abstraction Layer content conversion and generation is integrated. The Service Abstraction Layer also "knows" the different channels and corresponding formats, so that the correct content format for the respective platform is created on request.

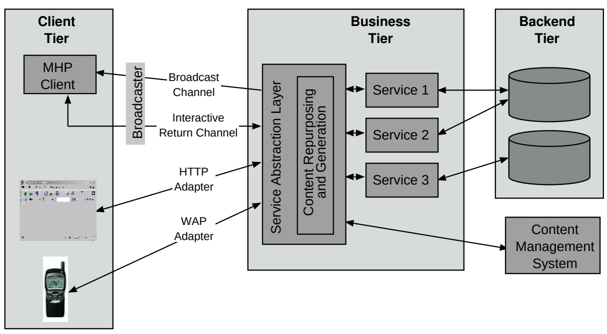

Fig. 6. Service Abstraction Layer Architecture for MHP, Web, and Mobile Phone Platforms

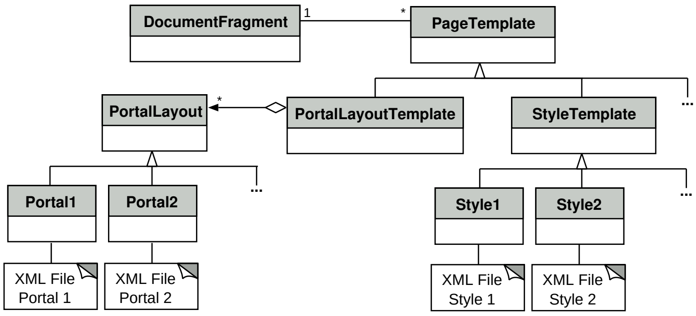

Fig. 5. Page Templates for Portal Layout and Common Styles

VI. Contribution of the Content Generation and Conversion Architecture

The architecture presented mainly focuses on abstracting from given content formats, conversions, content generation, and new media channels in a generic fashion. It integrates dynamic content creation components and other behavioral elements. However, the other problems and open issues identified in Section II-C are also tackled to a certain degree.

Incoming content format conversion is handled by content converters. The different formats are integrated by XML as a data glue so that there are $n + m$ converters to be written for n input and m output formats, instead of $n * m$ content converters.

Content format builders, fragments, and templates are different approaches for *abstracting from content format specifics*, if the content has to be generated dynamically. However, if *multiple target formats* are integrated, we more or less limit the design space in the formats to the common denominator. Content converters handle multiple target formats as well, if only a conversion is necessary. Of course, content format builders can be used in the converter to actually implement the format creation.

Using converters, builders, and templates we can associate static *styles*, such as Cascading Style Sheets in HTML, with the documents as well. The conversion itself can also impose certain stylistic conventions. If behavior has to be programmed with the conversion process, we can add page templates, as proposed in [16].

A Service Abstraction Layer allows us to integrate multiple services over *different channels*. This way typical other channels in the digital TV domain, such as the web or mobile devices, can be supported as well.

The generic representation such as XML text is usually readable and changeable easily, so that, for instance, *non-programmers as content creators* can manipulate it without programming experience. Graphical tools for manipulating the content have to be written additionally; however, this is a relatively simple and straightforward task. Existing content creation and management tools can be integrated as input formats, however, for a tighter integration more programming efforts are required.

The broadcasting context of the MHP case study imposes a set of unique issues that can only partly be addressed by our server-side architecture. For instance, in the architecture presented, the data glueing and conversion aspects are located on the server side so that the *resource limitations* of settop-boxes can be addressed. Thus, the XML parser and interpreter components do not have to be transfered to the client, and the costly conversions do not have to take place during application startup on client-side. However, this design decision means that we have to cope with the limited *flexibility of the broadcast channel*, or have to sent customization information through the return channel despite limited *return channel bandwidth*. E.g. if we create Java classes from the XML text, we have to code all configurations that have to happen on the settop-box into the generated Java code or send these informations via the return channel.

VII. Related Work

SysMedia's Magenta is a content authoring and management tool for interactive digital television that also supports content conversion. As a content management system it centrally unifies the activities of the content provider from content development over content management over content broadcast up to content consumption. That is, it can read all input types and convert them to output types. It stores the content and content application templates in central repositories. It handles the delivery of the content to the broadcaster. Here, the content management tool serves as a lightweight workflow engine unifying all tasks in the content production and publication chain.

Application generators compose applications by drag & drop. Several ”components” define building blocks for applications that can be assembled like movies in a cutter application. Naturally, such applications are quite simple and linear. Thus for very simple television applications, application generators, such as 4DL's 4DAuthor, may be an alternative to the architecture proposed.

An integrated development environment let us write Java programs in an editor with syntax high-lighting and other functionalities. A graphic GUI builder enables us to build parts of the application by drag & drop, that are simple (like GUI building) in their semantics, but complex to build in Java. Studio+ is an example product in the area of digital television.

In the web context there are quite a few projects that deal with similar issues as discussed in this paper. For generating and conversion to HTML these frameworks can be used for implementing the architecture presented.

Template-based approaches, such as PHP [1], ASP, JSP, or ColdFusion, let developers write HTML text with special markup. The special markup is substituted by the server, thus, a new page is generated, which composes content dynamically into the template. These approaches can be used to implement content format templates. Often web-based applications require more complex interactions than simply expressible with templates. Sometimes, the same actions of the user should lead to different results in different situations. Most approaches do not offer high-level programmability in the template or conceptual integration across the template fragments. For recurring design elements, the same fragments of a template have be implemented redundantly. Application parts and design are not clearly separated. Template fragments cannot be cleanly reused. Complex templates may quickly become hard to understand and maintain. Template languages are more simple than constructive approaches, and thus, easier to understand for non-programmers. They offer also a higher performance than constructive approaches.

Custom web servers, such as AOL Server [4], WebShell [13], Zope [9], the Ars Digita community system [8], or ActiWeb [11] provide more high-level environments on top of ordinary web-servers. Often they provide integration with high-level languages, such as scripting languages, for rapid customizability. Most often a set of components is provided which implement the most common tasks in web-application development. In WebShell [13] a Message Redirector for URL mapping is provided. WebShell constructs pages from Tcl code and it thus provides a constructive approach for building up pages dynamically. Actiweb [11] provides a Service Abstraction Layer architecture with a Message Redirector for URL mapping. It also provides content format builders for HTML code.

VIII. Conclusion

In this paper we have presented an integrated architecture for dynamic generation and conversion of content for digital television platforms. The architecture is well-integrated with the broader reference architecture for interactive broadcast product lines, as presented in Section II-B. Similar architectures may be applied for other channels (such as the web or mobile devices) as well. These other channels are integrated using a Service Abstraction Layer. The generic XML format enables a business television provider to define task-specific languages in the domain of the customer's business. These languages are well-integrated with the information architecture. It is not much effort to build graphical tools for end-users on top of this information architecture, say, by using reflection functionalities on the business logic classes. We can build architectures that provide a separation of concerns between content, styles, formats, and channels. Therefore, different technological choices can be exchanged against each other. Different converter, builder, and template technologies can be integrated so that the most suitable technology can be applied for each task.

Acknowledgements

The work described in this paper has partially be founded by the TPMHP EU project and an industry cooperation with the company BetaBusinessTV. The author would like to thank Kersten Dahl, Michael Goedicke, Gustaf Neumann, Julius Thiele, and Oliver Vogel for their helpful inputs on various aspects of the work presented.

References

[1] S. S. Bakken and E. Schmid. PHP manual. http://www.php.net/manual/en/, 1997-2001.
[2] J. Clark. Expat - XML parser toolkit. http://www.jclark.com/xml/expat.html, 1998.
[3] J. Clark. XSL transformations (XSLT). http://www.w3.org/TR/xslt, 1999.
[4] J. Davidson. Tcl in AOL digital city the architecture of a multi-threaded high-performance web site. In *Keynote at Tcl2k: The 7th USENIX Tcl/Tk Conference*, Austin, Texas, USA, February 2000.
[5] ETSI. MHP specification 1.0.1. ETSI standard TS101-812, October 2001.
[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
[7] M. Goedicke, G. Neumann, and U. Zdun. Message redirector. In *Proceeding of EuroPlop 2001*, Irsee, Germany, July 2001.
[8] P. Greenspun and E. Andersson. Using the ArsDigita community system. *ArsDigita Systems Journal*, Feb 1999.
[9] A. Latteier. The insider's guide to Zope: An open source, object-based web application platform. *Web Review*, 3(5), March 1999.
[10] D. Megginson. SAX 2.0: The simple API for XML. http://www.megginson.com/SAX/index.html, 1999.
[11] G. Neumann and U. Zdun. Distributed web application development with active web objects. In *Proceedings of The 2nd International Conference on Internet Computing (IC'2001)*, Las Vegas, Nevada, USA, June 2001.
[12] G. Neumann and U. Zdun. Pattern based design and implementation of a XML and RDF parser and interpreter: A case study. In *Proceedings of 16th European Conference on Object-Oriented Programming (ECOOP 2002)*, Malaga, Spain, June 2002.
[13] Andrej Vckovski. Tcl Web. In *Proceedings of 2nd European Tcl User Meeting*, Hamburg, Germany, June 2001.
[14] O. Vogel. Service abstraction layer. In *Proceeding of EuroPlop 2001*, Irsee, Germany, July 2001.
[15] W3C. Document object model. http://www.w3.org/DOM/, 2000.
[16] U. Zdun. Dynamically generating web application fragments from page templates. In *Proceedings of Symposium of Applied Computing (SAC 2002)*, Madrid, Spain, March 2002.