A Framework for Developing Pervasive Cross-Media Applications based on Physical Hypermedia and Active Components

Beat Signer
Institute for Information Systems, ETH Zurich
CH-8092 Zurich, Switzerland
Email: signer@inf.ethz.ch

Moira C. Norrie
Institute for Information Systems, ETH Zurich
CH-8092 Zurich, Switzerland
Email: norrie@inf.ethz.ch

***Abstract*—We present a framework that supports the development of pervasive cross-media applications through a clean separation of interaction design and application programming. The approach is based on a novel concept of active components that provides a lightweight mechanism for linking physical and digital entities to services. We show how the development of cross-media applications can be simplified by adopting an authoring rather than a programming approach and how we achieved this by integrating the active component concept into an extensible hypermedia server.**

I. Introduction

With the rapid growth of interest in pervasive cross-media applications, we address the problem of how to support the developers of such applications by minimising the programming effort. Our goal was to develop a platform and tools that, similar to the World Wide Web, turn the development of cross-media interfaces into an *authoring activity* rather than a *programming activity*. By this, we mean that users may develop a wide variety of pervasive cross-media applications by authoring links between all sorts of physical artefacts and digital content and services.

The concept of *active components* is used to link to pieces of program code that may either be applications in their own right or bridges to existing applications. By providing an extensive library of generic active components, developers can simply define links to a wide range of services without having to write any code themselves. Only in the case where the library has to be extended with new active components is some programming effort required. The concept of active components therefore provides a clear separation of interaction design and application logic which both simplifies the development task and provides maximum flexibility.

In this paper, we describe how the concept of active components was integrated into a hypermedia framework that supports the linking of physical and digital entities to provide a framework for the development of pervasive cross-media applications. The authoring process required to develop an application is presented in detail, including mention of specific authoring tools that have been developed to simplify the process even further. As a showcase of the approach, we describe how the framework was used by an artist to develop an interactive media installation.

We start with a discussion of related work and then present the underlying extensible hypermedia framework. The concept of active components and the mechanisms used to implement it are described in the following section. We then describe the process of developing an application through the process of authoring link definitions and present a specific application. Finally we give some concluding remarks and outline future work.

II. Related Work

Over the last decade there has been a trend to provide new types of tangible user interfaces for accessing information managed in digital information spaces in a natural and effective way. For that reason, information management systems have been extended to not only include digital information but also associations to physical artefacts based on evolving technologies for object identification such as RFID tagging. While few projects deal with the information-centric aspects of mixed-media integration of digital and physical artefacts, various frameworks and toolkits have been developed for the rapid prototyping of physical and tangible user interfaces in terms of the hardware integration.

Phidgets [1] is a commercial toolkit for the integration of different input and output hardware components on the basis of USB-based hardware boards. All hardware components are connected to a single computer and, as soon as a distributed set of sensors and actuators are used, detailed low-level programming knowledge is required for the communication and synchronisation between the distributed system components. The *Shared Phidgets* toolkit [2] supports developers in dealing with these hardware components by providing a shared data space maintaining a runtime model of all distributed Phidgets that allows components to be rearranged and combined in various ways. It is no longer up to the programmer to implement all of the functionality for the integration of a distributed component, as they can rely on the distributed Model View Control (dMVC) architecture offered by the Shared Phidgets toolkit.

The *VoodooIO* [3] platform provides the flexibility to rearrange hardware sensors and actuators based on a novel material called the *network substrate*. Arbitrary surfaces covered with the network substrate can easily be transformed into special control areas by just sticking different buttons and hardware components into the substrate material. New components that have been added to the network substrate are automatically detected by the system and can be configured using a graphical user interface. Note that VoodooIO does not provide a specific interface solution but rather enables users to design their customised tangible control structures.

The flexible integration of new hardware components is also supported by the *d.tools* [4] project. In addition, d.tools integrates the design, testing and analysis of new hardware devices. While the Phidgets project provides a clean API for programmers who want to develop new physical user interfaces, the d.tools software focusses on the prototyping of new tangible interfaces by designers. In addition to existing other physical prototyping tools, d.tools offers powerful functionality for the analysis of user test sessions. By selecting a specific state from the state chart or even performing a "physical query" directly by interacting with the device, the designer can access relevant parts of movie clips that have been recorded during the test sessions.

In contrast, the hypermedia community has always been concerned with the provision of flexible infrastructures for managing pieces of information through associative linking. While earlier hypermedia systems mainly focussed on linking various forms of digital information, more recent systems organise information in mixed-media environments where physical information coexists with digital data. For example, in the *WorkSPACE* project [5], real-world objects are digitally augmented to enhance the working experience in architectural settings. Physical objects are linked to digital entities by marking them with RFID tags and the digital representation of various real-world objects, including paper documents such as architectural drawings, is organised in three dimensional space based on the *Topos* spatial hypermedia infrastructure [6].

III. iServer Platform

In this section, we describe the iServer platform for physical hypermedia and some of the digital and physical resource plug-ins that it offers. The iServer architecture was designed as an extensible Java-based cross-media information management platform which can support any type of digital or physical media. The core component provides concepts for linking, layer and user management, while a resource plug-in mechanism based on the *resource* and *selector* concepts enables the integration of new resource types by providing a resource-specific Java implementation of a resource and its corresponding selector.

In Figure 1, there is a simplified version of the iServer's underlying *resource-selector-link* (RSL) model [7] showing only the main concepts. The `Selectors` representing elements within a resource in combination with the `Resources` which represent entire resources are the central components for the resource plug-in mechanism. For a particular media type, the iServer platform can be extended by introducing a component that defines selectors and resources for that media type. For example, Figure 1 shows a resource plug-in for physical objects (iObject) where a resource is a set of objects and a selector is represented by a physical object identifier (e.g. RFID tag). The iPaper plug-in enables the integration of paper documents within the iServer cross-media information space based on selectors defined by shapes within a page.

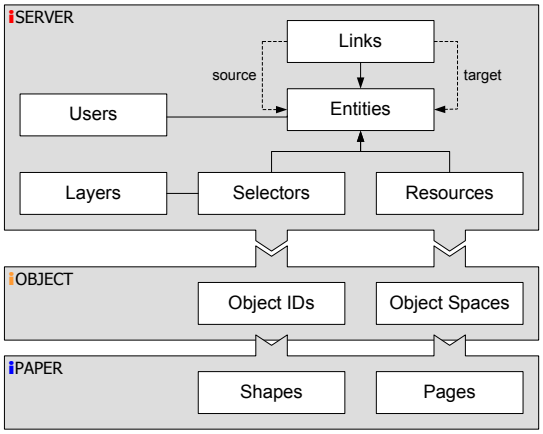

Fig. 1. iServer resource plug-ins

A major advantage of the iServer plug-in mechanism is the tight integration over all types of media based on a common core link model rather than isolated applications for specific kinds of media such as web pages or movie clips. As soon as a plug-in for a new resource type has been implemented, entities of that medium can be cross-linked with instances of any existing media type. In Table I, a list of existing iServer resource plug-ins is given with the corresponding resource and selector types for each type of media. Note that the list of plug-ins is far from being complete since arbitrary digital or physical resources could be added.

Medium	**Resource**	**Selector**
physical object	RFID space	RFID tag
web page	XHTML document	XPointer
paper	document page	shape
movie	mpeg file, avi file etc.	time span
image	gif file, jpeg file etc.	shape

TABLE I
iSERVER RESOURCES AND SELECTORS

The iServer plug-in for paper allows physical documents to be linked to digital content. iServer plug-ins are also available to integrate XHTML content, movies and RFID-tagged physical objects. This means that while XHTML documents and movies have basic support in the core iServer implementation that would allow a web page to be linked to a movie, these plug-ins allow elements within a web page or movie to be linked and not just entire resources. This implies that selectors

are defined for addressing parts of the corresponding resources. The integration of a completely new resource type requires some effort to implement the corresponding resource plug-in and all its necessary functionality. Therefore, to make it even easier for developers by minimising the programming effort, we have extended iServer with a concept of active content that allows different services to be integrated without having to implement a specific resource plug-in.

IV. Active Content

While regular links just return a single piece of information such as an HTML page or a movie, active content is represented by active components which are bound to a piece of Java code that can be executed either on the server or the client side. For example, an active component could open a given web page, contact a database or communicate directly with a running application. Figure 2 provides an overview of the active component architecture.

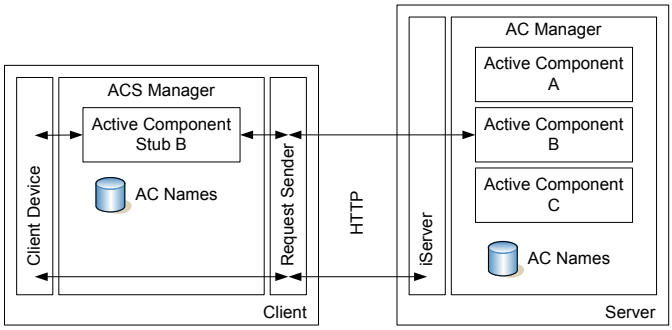

Fig. 2. Client- and server-side active components

After some input has been acquired by a client device, the client-side component has to check for any running active components. This means that the client distinguishes two working modes: a *default mode* where no active component is running on the client side and an *active mode* where an active component has been instantiated and is currently running. Let us assume that no active component is currently running on the client side. An incoming request by the client device is sent to the server using the `Request Sender` class. On the server side, the incoming request is handled by iServer. The resolved link target can either be "static content" or an active component. In the latter case, the Active Component Manager (`AC Manager`) will load the component on the server. Each active component stored in the database has an identifier and some additional parameters which are used to initialise the Java object at instantiation time. Before an active component can be used within the iServer framework, it has to be registered with the Active Component Name Directory (`AC Names`). For each active component identifier, a client (stub) and server (logic) binding has to be provided defining the Java classes to be loaded when an active component is activated. An example of an Active Component Name Directory with the bindings for a single active component is shown in Figure 3.

```
<?xml version="1.0" encoding="UTF-8"?>
<activeComponents>
<activeComponent>
   <identifier>COMMAND_LINE</identifier>
   <logic>org.iserver.logic.CommandLine</logic>
   <stub>org.iserver.stub.CommandLineStub</stub>
</activeComponent>
...
</activeComponents>
```

Fig. 3. Active Component Name Directory

The server will check if an active component for the given identifier has already been loaded since every active component is instantiated only once by the Active Component Manager. If a new active component logic has to be loaded, its `init` method, containing any program logic, is invoked by the Active Component Manager. After the `init` method has executed, an XML description of the active component is sent to the client side. Note that for each active component an arbitrary set of string parameters can be defined and handled in a generic way. On the client side, the Active Component Stub Manager (`ACS Manager`) does a lookup in the Active Component Name Directory, loads the corresponding stub component and invokes its `init` method for initialisation. Note that we are currently developing a distributed version of the Active Component Name Directory where information will be synchronised between different computers and be accessible by means of an active component lookup service.

The client switches to the active mode and dispatches the original request parameters, i.e. any information provided by the input device, to the active component's `processEvent` method. Since the client is now in active mode, all subsequent client device events will be handed over directly to the active component's `processEvent` method. Note that as soon as the active component's `setDone` method gets invoked as a result of the component's program logic, the active component stub will be unloaded and the client will switch back to default mode. However, before an active component is definitely unloaded, there is an upcall to its `finish` method. This enables the active component developer to execute specific program code to release any acquired resources (e.g. database or network connections). Each active component has an optional timeout parameter and will be terminated automatically if it has been idle for longer than a given timeout value.

A client-side active component stub can communicate directly with the server-side active component by sending special active component requests. All information encoded within an active component request is handed over to the server component's `handleActionRequest` method and the result is sent back to the client-side active component in XML format. Figure 4 summarises the functionality available for active component stub and logic entities.

The concept of client- and server-side active components that can communicate by sending special active component messages to each other has proven to be very useful if the client-side component has to retrieve additional information

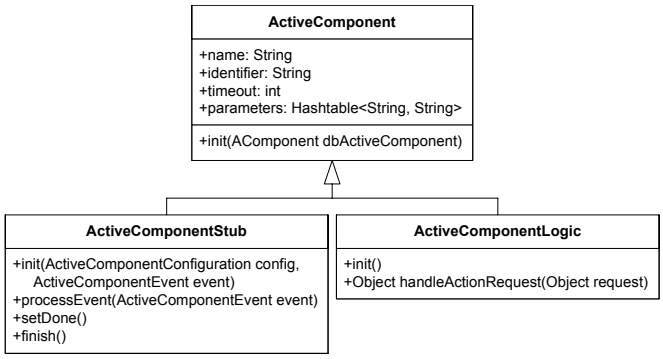

Fig. 4. Active component stub and logic

managed by the iServer database. Furthermore, the active component concept enormously simplifies the implementation of complex interaction components since the active component developer does not have to know all the details of the iServer architecture. They can focus on implementing the simple active component interface with only a few methods. Once a specific active component has been implemented, it can be reused in any application by just using it as a link target in the iServer framework. We have to distinguish two types of active components. There are *resource-specific active components* which require additional information from the input device triggering the link. An example of such an active component is the `CaptureNote` active component which captures information from an iPaper input device. These resource-specific active components can be reused in different applications but they always have to be used in combination with the appropriate input device. On the other hand, there are *generic active components* which do not depend on any additional information from the device triggering the link. An example of such a generic active component is the `Browser` active component which opens the system's default web browser with the URL specified as an active component parameter.

Various active components have been implemented for different interaction tasks as part of numerous iServer applications. The growing set of reusable active components simplifies the development of future applications since the application designer can focus on the realisation of new forms of interaction and does not have to spend most of their time dealing with the implementation of existing functionality. To give the reader an idea of how easy it is to implement a new active component we provide an example of the implementation of a `CommandLine` active component. The goal of this generic active component is to execute the string provided in its `command` parameter as if it would have been typed in the operating system's command line. The `CommandLine` active component could then be used to start arbitrary third party applications or execute any other system commands.

Figure 5 shows the implementation of the `CommandLine` active component's logic class. Since, for this active component, no operations have to be performed on the server side, we can simply inherit all functionality from the `EmptyLogic` active component without having to implement any new methods. Note that we could even not provide a new logic class and just register the `EmptyLogic` class in the Active Component Name Directory.

```
package org.ximtec.iserver.activecomponent.logic;
 public class CommandLine extends EmptyLogic {
 }
```

Fig. 5. CommandLine logic

On the client side, we have to perform a number of operations exactly once and then the active component should be terminated. As shown in Figure 6, this can be achieved by extending the `SingleEventStub`. This class simply terminates the active component stub the first time its `processEvent` method gets executed. As mentioned earlier, before an active component is unloaded, its `finish` method gets invoked and that is exactly where we can implement the required command line functionality. After retrieving the value of the active component's `command` parameter, we simply execute the command represented by the string value in a separate process.

```
package org.ximtec.iserver.activecomponent.stub;
import java.io.IOException;
...
public class CommandLineStub extends SingleEventStub {
 public static final String PARAM_COMMAND = "command";

public void finish() {
   try {
    Runtime.getRuntime().exec(getParameter(PARAM_COMMAND));
   }
   catch (IOException e) {
    LOGGER.log(Level.SEVERE, Constant.EMPTY_STRING, e);
   }
 } // finish
}
```

Fig. 6. CommandLine stub

The `CommandLine` active component example shows how easy it is to implement a new active component which can then be used in any future application. Most of the active components we have so far implemented only contain a few lines of code which, of course, may include access to external library functionality. While the active component concept has proven to be very effective in many tangible user interfaces that we have built to date, we will outline some potential extensions when discussing future work.

The idea of modular active components that encapsulate specific application logic is somehow related to Service Oriented Architectures (SOA). However, in the case of our active components, the program code is often executed on the client side to ensure tightly integrated real-time interaction with different input devices.

V. Authoring

The authoring of a pervasive cross-media interface based on the iServer platform and our concept of active components basically involves the definition of links between the

sources triggering an activity and the corresponding services represented by active components. Only in the case that some services have not yet been integrated or a new type of functionality is required, will a new active component have to be implemented by a programmer. As mentioned earlier, our goal was to have a clean separation of interaction design and application programming. Our framework provides different ways in which a tangible user interface's functionality may be authored in terms of linking the corresponding components together.

```
<?xml version="1.0" encoding="UTF-8"?>
<iserver>
...
<rfidTag id="rfidTagSky" creator="axel"
   layer="default" resource="lc">
  <name>RFID tag for Sky scenario</name>
  <id>010000004282B355</id>
</rfidTag>
<activeComponent id="skyScenario" creator="axel">
  <name>The Sky scenario</name>
  <properties>
   <parameter>
    <key>org.ximtec.iserver.ac:request</key>
    <value>anchor=get_mood&amp;mood=Sky</value>
   </parameter>
  </properties>
  <identifier>org.iserver.OMSWE_REQUEST</identifier>
</activeComponent>
<link id="skyLink" creator="axel"
   sources="rfidTagSky" targets="skyScenario">
  <name>Change to the Sky scenario</name>
</link>
</iserver>
```

Fig. 7. XML-based authoring

A first possibility is to author an application by writing an XML file with all the necessary information. Figure 7 shows parts of such an XML file containing a selector, an active component and an association between these two components represented by a link. In this case, the selector is an `rfidTag` which means that the link will be triggered each time the antenna reads the RFID tag with the identifier `010000004282B355`. The link's target has been defined as an active component with the identifier `OMSWE_REQUEST`. This active component sends the value of its `request` parameter to the database system (OMSWE) used in the Lost Cosmonaut installation described later. In the case of the information encoded in this request, the ambient mood of the installation will be changed to `Sky` which means that there will be a special ambient sound, lighting and pictures. So basically the XML snippet shown in Figure 7 binds an RFID tag to a service for changing the setting of the ambient environment. Having defined all the necessary functionality in the XML file, it is imported into the iServer database and the system can immediately be tested.

We have had multiple projects together with artists and designers and, after a short introduction, they all managed to

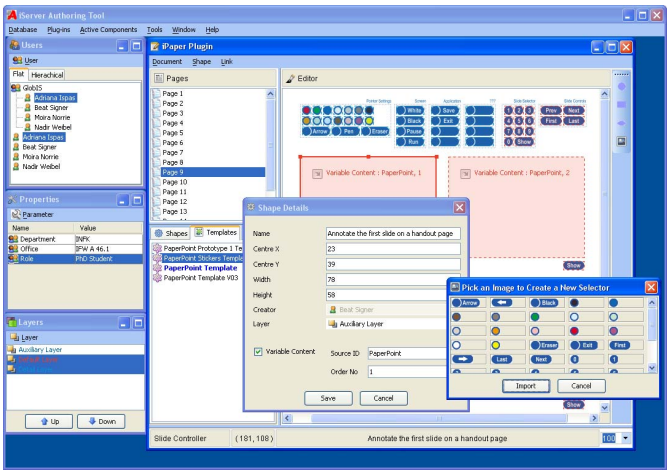

Fig. 8. iServer authoring tool

create the necessary XML files used to generate the required iServer databases. However, the manual authoring of XML documents is clearly tedious and also error prone. We therefore developed a visual authoring tool that hides all of the XML "complexity" and further simplifies the authoring process. An early version of this iServer cross-media authoring tool is shown in Figure 8. A visual authoring plug-in was developed for different resource types in a similar way as resource plug-ins were developed for the iServer data management. Note that, in addition to the visual authoring tool, the XML-based authoring approach still makes sense for developers who would like to enter new iServer data from third-party applications.

VI. Application

The Lost Cosmonaut project [8] is an example of a pervasive interactive cross-media installation that was realised based on iServer and the active component concept. The main goal of the project was to design and realise an artistic setting for interactive narratives and story writing and the project was carried out as six month collaboration between an artist and our research group. In the Lost Cosmonaut installation, a user sits in a darkened room in front of a semi-circular desk. The wall in front of the user contains a large round screen for projecting digital information. On the desk there is a digital pen[1] and three documents forming part of an interactive narrative about a cosmonaut lost in space: a star map, a book of broken images and a collection of love letters. While a visitor is interacting with the documents, the content presented on the round screen as well as the ambient sound and the lighting are changing. There is some pre-authored content but visitors are encouraged to add texts and drawings to the artefacts thereby continuing the interactive narrative.

In the Lost Cosmonaut installation, the ambient mood automatically changes based on the document that a visitor is working on. The documents are tagged with RFID identifiers

[1] www.anoto.com

and an antenna is placed underneath the table. The RFID antenna detects when a new document is placed on the table, triggers the corresponding link which was defined using the iObject plug-in for iServer and accesses the associated active component which activates the appropriate mood. The overall Lost Cosmonaut architecture includes an iServer Client and a Client Controller for controlling various things such as ambient sound and light as well as iServer and a specific Lost Cosmonaut application database on the server side. The Client Controller also communicates with the RFID antenna installed underneath the table and sends any recognised tag identifier to the iObject resource plug-in.

During the development of the Lost Cosmonaut installation, the requirements changed as the artist developed his ideas and it was essential to have an extensible and flexible system architecture. The chosen architecture not only supported rapid application prototyping in terms of content and services, but also enabled an easy integration of new input and output channels. Furthermore, there was a clear separation between the implementation of new application functionality and services in terms of active components and the authoring and design of the interactive information environment. This enabled the artist to author the required links and active components without having to write any program code by editing an XML document containing the configurations of the corresponding active components.

An innovative aspect of the Lost Cosmonaut installation is the fact that information written into the physical space with the digital pen may be integrated in three different ways into the interactive narrative. To illustrate the three different ways of content handling, we discuss the interaction with the star map, one of the three documents, and outline how content from all three documents becomes interweaved to a single story space.

If a user writes a new dedication on the star map, the information is physically "stored" as new content on the document and forms part of a subsequent user's experience. By writing a dedication on the star map, a new active area is generated for the handwritten information. The new active area is randomly linked with image or film material. If later, the same or another user touches the dedication, an image or film is shown. Finally, the dedication is captured and stored in a database as an XML document containing a set of points together with a timestamp for each point. The digitally captured information is dynamically linked from active areas in the love letters and gets activated when the corresponding part of a sentence is selected in a love letter. The temporal information together with the stroke information is used to replay the handwriting as an animated drawing.

The application presented in this section is only one of many applications that we have implemented based on iServer and its active component mechanism. Given the constantly changing and evolving hardware available for services such as object tracking, it has been extremely beneficial to have a development platform that can be easily extended and adapted to cater for new technologies and services.

VII. Conclusion and Future Work

We have presented our framework for the prototyping and development of pervasive cross-media applications based on a physical hypermedia server and active components. The active component approach enables the rapid development of pieces of program code binding third party services or providing specific application logic. Furthermore, the event-based information processing within an active component, whereby an active component is instantiated on link activation and then gets control over any data provided by an input device until the active component is terminated, has proven be an effective solution for the tangible user interfaces we have implemented so far. The open architecture enables the integration and control of arbitrary services and, at the same time, the growing set of active components and resource plug-ins supports the rapid development of new applications.

We are currently developing different plug-ins for the visual iServer authoring tool which will support developers, designers, artists and other users in the authoring of different interaction scenarios. It is also planned that the active component configurations can be directly specified in the iServer authoring tool rather than having to use the iServer Java API or by providing the corresponding XML document.

Acknowledgment

We would like to thank Philipp Bolliger and Samuel Willimann for their work on the active component framework. We would further like to thank Nadir Weibel and Adriana Ispas for working on different parts of the iPaper framework and Axel Vogelsang for the Lost Cosmonaut collaboration.

References

[1] S. Greenberg and C. Fitchett, "Phidgets: Easy Development of Physical Interfaces Through Physical Widgets," in *Proc. of UIST 2001, 14th Annual ACM Symposium on User Interface Software and Technology*, Orlando, USA, November 2001.

[2] N. Marquardt and S. Greenberg, "Distributed Physical Interfaces With Shared Phidgets," in *Proc. of TEI 2007, 1st Intl. Conference on Tangible and Embedded Interaction*, Baton Rouge, USA, February 2007.

[3] N. Villar and H. Gellersen, "A Malleable Control Structure for Softwired User Interfaces," in *Proc. of TEI 2007, 1st Intl. Conference on Tangible and Embedded Interaction*, Baton Rouge, USA, February 2007.

[4] B. Hartmann, S. R. Klemmer, M. Bernstein, L. Abdulla, B. Burr, A. Robinson-Mosher, and J. Gee, "Reflective Physical Prototyping Through Integrated Design, Test, and Analysis," in *Proc. of UIST 2006, 19th Annual ACM Symposium on User Interface Software and Technology*, Montreux, Switzerland, October 2006.

[5] K. Grønbæk, J. Kristensen, P. Ørbæk, and M. A. Eriksen, "Physical Hypermedia: Organizing Collections of Mixed Physical and Digital Material," in *Proc. of Hypertext 2003, 14th ACM Conference on Hypertext and Hypermedia*, Nottingham, UK, August 2003.

[6] K. Grønbæk, P. P. Vestergaard, and P. Ørbæk, "Towards Geo-Spatial Hypermedia: Concepts and Prototype Implementation," in *Proc. of Hypertext 2002, 13th ACM Conference on Hypertext and Hypermedia*, College Park, USA, June 2002.

[7] B. Signer and M. C. Norrie, "As We May Link: A General Metamodel for Hypermedia Systems," in *Proc. of ER 2007, 26th Intl. Conference on Conceptual Modeling*, Auckland, New Zealand, November 2007.

[8] A. Vogelsang and B. Signer, "The Lost Cosmonaut: An Interactive Narrative Environment on Basis of Digitally Enhanced Paper," in *Proc. of VS 2005, 3rd Intl. Conference on Virtual Storytelling*, Strasbourg, France, December 2005.