Designing and Implementing Physical Hypermedia Applications

Cecilia Challiol[1], Gustavo Rossi [1,3,*], Silvia Gordillo[1,2], and Valeria De Cristófolo[1]

[1] LIFIA, Facultad de Informática, UNLP, La Plata, Argentina
{ceciliac, gustavo, gordillo, valeriac}@sol.info.unlp.edu.ar
http://www-lifia.info.unlp.edu.ar
[2] Also CICPBA
[3] Also CONICET

Abstract. In this paper we present a design approach and a software framework for building physical hypermedia applications, i.e. those mobile (Web) applications in which physical and digital objects are related and explored using the hypermedia paradigm. We show how we extended the popular MVC metaphor by incorporating the concept of located object, and we describe a framework implementation using Jakarta Struts. We first review the state of the art of this kind of software systems, stressing the need of a systematic design and implementation approach; we briefly present a light extension to the OOHDM design approach, incorporating physical objects and "walkable" links. We next present a Web application framework for deploying physical hypermedia software and show an example of use. We compare our approach with others in this field and finally we discuss some further work we are pursuing.

1 Introduction

A physical hypermedia (PH) application is a kind of ubiquitous software in which the mobile user can explore real world objects using the hypermedia paradigm. In these software systems, physical objects are augmented with digital information in such a way that when the user is in the vicinity of an object, he can access the additional information. Besides, physical objects can be seen as nodes in a hypermedia network; the user can follow a link to navigate to other related objects, either virtually, e.g. when the links are implemented using a Web browser, or physically by moving to the target object.

A simple example is a mobile tourist guide. When the user is in front of a monument he can read information about the monument in his mobile device (e.g. in a Web page); he can also explore the digital hyperspace by navigating to other related documents. Some links, however, may point him to other tourist spots in the same city. Instead of navigating in the usual digital way, he has to "walk" the link [9]. The software system may react to his intention to navigate by providing him a map showing the best way to access the target place. Notice that this application behavior, while somewhat similar to existing families of location-based services, is completely based

* This work has been partially funded by Project PICT 2003, Nro 13623, SeCyT.

M. Gavrilova et al. (Eds.): ICCSA 2006, LNCS 3983, pp. 148 – 157, 2006.

on the well-known ideas of hypermedia navigation that became popular with the Web. It is not surprising then that the physical hypermedia paradigm has been considered to be a good vehicle to integrate the Web and the world [8] and as a tool to improve collaboration in a social setting [3].

We have been working on different aspects of the PH applications' life cycle. In [6] we presented a modeling and design approach that allows a high-level specification of the intended functionality of a PH application. In [5] we analyzed a more complex engineering aspect of this kind of software: how to clearly decouple the most critical concerns that designers face when building PH software. We defined the concept of concern-driven navigation to support the user while exploring different application themes and to clearly separate digital from physical navigation.

In this paper we present a software framework that allows seamless implementation of PH applications. This framework, which implements an extension of the popular MVC metaphor, has been built on top of the well-known Jakarta Struts [16] Java infrastructure and therefore can be easily used by Web application designers.

The main contributions of this paper are the following:

- We present a reusable software substrate that supports the main abstractions in the PH paradigm,
- We show how to integrate this framework with a modular design approach thus covering the full PH software life cycle.
- By describing the implementation of a simple application we introduce a set of good design practices that help the implementer to cope with the difficulties that arise while building this kind of ubiquitous web software

The rest of the paper is organized as follows: In Section 2 we describe the requirements of an implementation framework for PH and present some background concepts needed in the rest of the paper. In Section 3 we present our extension to the MVC metaphor and describe our framework implementation. In Section 4 we present an example of use of the framework. In Section 5 we compare our work with other similar approaches and in Section 6 we present some concluding remarks and further work on this area.

2 Requirements and Background

Researchers have emphasized the feasibility of the PH paradigm by building software infrastructures that support the ideas underlying PH [7, 8, 13]. However, we think that for PH applications to become mainstream we need to provide tools to build them as a particular class of mobile Web software, i.e. we need to specialize existing Web development architectures and frameworks in such a way that they support the main concepts behind PH. We also need to provide a conceptual framework to reason on this kind of software and a comprehensive bridge among modeling concepts and the implementation tools. Particularly, a development framework must:

- simplify the development of this kind of software, providing reusable classes and semi-complete application structures together with "hot-spots" in which developers can add the specific aspects of their own applications,

- allow a clear separation between application objects and the lower level aspects needed to indicate their physical position and to check whether a user is in front of an object,
- support different navigation strategies, such as digital (as in the Web) or physical, allowing the designer to easily implement both of them,
- provide ways to maintain basic contextual information, e.g. when the user navigates digitally, keep the physical links corresponding to the current location visible.
- Additionally, supporting different mobile devices is a must; finally, applications built using the framework should also support conventional access, e.g. from a desktop browser. In the following sub-sections we briefly describe some background concepts that we will use throughout the paper, namely the philosophy underlying our design approach, and the basic concepts behind the MVC metaphor.

2.1 Design Issues for Physical Hypermedia

To make this discussion concrete, we define a PH application as a hypermedia application (i.e. the access to information objects is done by navigation) in which all or some of the objects of interest are real-world objects which are visited by the user "physically". The most usual scenario for these applications involves a mobile user and some location sensing mechanism and underlying software that can determine, for example, when the user is within interaction range of one of these objects. For the sake of conciseness, we also assume that digital information (data about physical objects and links) is obtained from a Web server and navigated using a browser. A simplify schema showing these ideas is presented in Figure 1.

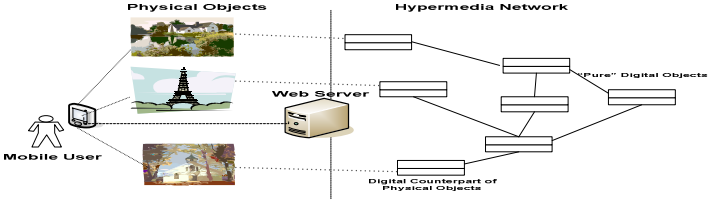

Fig. 1. Physical Hypermedia and the Web

We chose to extend the Object-Oriented Hypermedia Design Method (OOHDM) [14] by incorporating the concept of physical objects and "walking" navigation [9]. In a PH application, we aim at expressing, in an implementation-independent way, which are the objects of interest and their properties (including their location), how they are linked, which links should be implemented as conventional and which should be "walked" by the user. Following our approach, a PH application is developed in a four stages process: application modeling, navigation design, user interface design and implementation.

During application modeling we produce a two-layered model; the first layer contains the application objects, their properties, relationships with other objects and

behaviors (described in UML [17]); in the second one, we describe the physical (e.g. location) counterparts of application classes. Physical Objects are described as roles, in fact decorations [15,4] of digital objects and contain the object's location, geographical relationships, and the behaviors needed to manipulate positions, such as determining if the user is in front of the object or calculating how to reach a physical object. In Figure 2, we show these two layers.

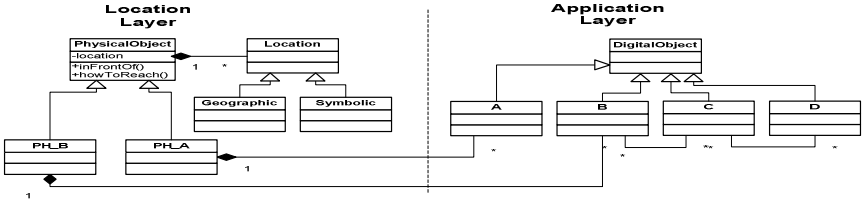

Fig. 2. The products of Application Modeling

The two physical classes (*PH_A*, and *PH_B*) wrap application classes A and B, thus obliviously adding them their physical properties. The navigation model specifies which nodes the user will explore and the links connecting these nodes. Nodes are defined as views on application objects and contain the information to be displayed. Links can be digital or physical. A digital link allows "conventional" navigation (i.e. as in the Web), while physical or "walking" links express a relationship in which the target object is physical, and exploring the object implies that the user must change his current position. Notice that physical links might be derived from both conceptual and geographical relationships. More details on the design approach can be read in [5,6].

2.2 The MVC Metaphor for Web Applications Development

The Model-View-Controller [11] is perhaps the most established paradigm for developing interactive applications. Originally developed for desktop software in the context of the Smalltalk environment, it has evolved and it is widely used in Web applications development. It proposes to partition the concerns of an interactive application in three components.

- the Model, which contains the basic application's data and behaviors.
- the View(s) which comprises the user interface objects.
- the Controller (s) which is in charge of managing user interaction, and coordinating the View and the Model.

The MVC has been implemented in different platforms and there are dozens of tools supporting software development with the MVC, for example [10]. In our case, we decide by the popular Jakarta Struts.

We chose to extend the MVC model for several reasons: first, MVC provides a reasonable model for separation of concerns in (mobile) Web applications; besides, we have used MVC-based architectures to support the implementation stage for OOHDM models [2], and finally it is a well-known metaphor, used in every modern middleware platform for Web application development. In the following sections we describe our approach, concentrating in the server-side. Details on client side

adaptations such as providing communication mechanisms between sensing hardware and software and Web browsers, though important in our research, are outside the scope of this paper.

3 A MVC Framework for Physical Hypermedia

From a thorough analysis of each one of the MVC's components, we decided to extend the Controller component to support location-aware requests. We decided to do this because we found that both the View and the Model components can support Physical Hypermedia functionality, without modifying their essence.

As explained in Section 2 the Controller acts as a coordinator among the Model and the View. In our extension, the controller will need to identify if a request implies managing a location, and it will be in charge to process those requests that do involve location information.

3.1 A Conceptual View of the Location-Aware MVC

The two main components of the controller are, according to [10]: the *InputController* and the *ApplicationController*. Our extension involves the *InputController* since it deals with resolving a parameter of the request, in particular the recognition of the physical object in the user's vicinity. For the sake of modularity and compatibility we avoided changing this component but we introduced a new one, the *LocationController*. In this way we didn't clutter the standard controller with new functionality and, besides, both of them can evolve separately.

The *LocationController* will deal with those implicit or explicit parameters which correspond to requests that involve location information. Considering that the kind of pre-processing needed by a broader range of applications might involve other issues, we devised a *DispatcherController* as a Façade [4] to determine which specific controller receives control; i.e. depending on the nature of the application the *DispatcherController* establishes which Controller will be the actual *InputController*'s collaborator as shown in Figure 3. In the case of requests which do not need any pre processing, the *DispatcherController* delegate control directly to the *ApplicationController.*

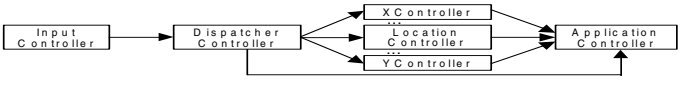

Fig. 3. Adding a new Controller

The *DispatcherController* analyzes the request. If it is a pure digital request it delegates control to the *ApplicationController*. If the request involves location information it delegates to the *LocationController*, which will analyze the location issues. A complete diagram of the extended MVC architecture is shown in Figure 4. To make the discussion more concrete we next detail our Struts implementation.

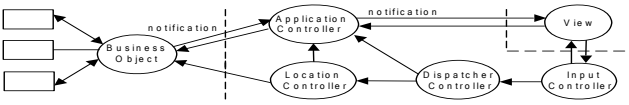

Fig. 4. Extended MVC for Physical Hypermedia

3.2 Adding Location-Awareness to Struts

By distinguishing the format of a URL contained in a request, Struts allows to define more than one control Servlet. In our implementation, a URL with the traditional format (*.do), will be dealt by the *ActionServlet.* Meanwhile, a URL with a location-compliant form (in our case /location/*), will be analyzed by the *LocationActionServlet.* This is an easy and straightforward way to implement the task of the *DispatcherController*. Both, the configuration of the new Servlet, and the format of the location-compliant URL are configured in the Struts file web.xml. We next examine how to carry out the *LocationController*'s task.

Physical Hypermedia applications may use different location models (e.g. symbolic, geometric, etc); this means that the location contained in the request has to be interpreted in the corresponding location system to obtain a correct result. To achieve this goal we decouple the corresponding functionality and create a hierarchy of *LocationFinder* classes.

LocationFinder is an abstract class which describes the common functionality of all location finders. Concrete subclasses (e.g. *SymbolicLocationFinder*) allow the developer to specialize this functionality. Sub-classes must implement at least two methods: one to identify the physical object which the user is facing (*inFrontOf*), and the other that returns the path between two physical objects (*howToReachFrom*). Each concrete sub-class implements this functionality using the concrete location model and interacting with physical (application) objects defined in the Model component.

As the objects returned by these methods must be used both by the Actions and by the JSPs (the View) we make them persistent by storing them in the Struts's session, under the name specified by the developer in the configuration file.

Following the standard way to extend Struts with specific business logics, we decided to create a specialized *RequestProcessor*, the *LocationRequestProcessor*, which is configured in the file location-struts-config.xml. The *LocationRequestProcessor* collaborates (with the mediation of the *LocationActionServlet*) with the concrete *LocationFinder* to implement the pre-processing of the request.

To complete the specification we also defined: *LocationEvent* (an obligatory property in the configuration file location-struts-config.xml, which allows to specify which of the *LocationFinder´s* method must invoke the *LocationRequestProcessor)*, *LocationActionMapping* (allows the retrieval of the new LocationEvent property), *Location-struts-config_1_1.dtd* (allows considering the incorporation of the new property)*, LocationConfigRuleSet* (incorporate the new property to the structure of the file location-struts-config.xml).

In summary the relation between the MVC elements and those that arise from the extension of Struts can be represented as shown in Figure 5:

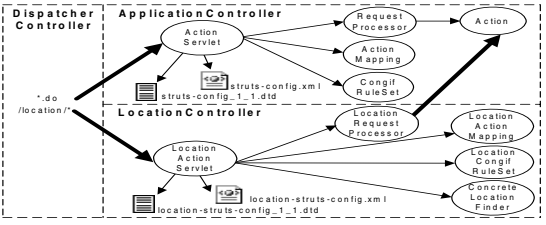

Fig. 5. The complete Struts extension for handling location data

4 Using the Framework

As a proof of concept we have instantiated the framework in a Natural Sciences Museum. Though our prototype uses a particular sensing mechanism (infrared sensors) and location model (symbolic), most design decisions can be easily understood while analyzing the example. We first produced a conceptual model, including the location enrichment. In Figure 6 we show a simplified diagram including some attributes and relationships for animals and the period in which they lived. The location attribute has been simply defined as an identifier, because we used a simple location model. Physical Animal also includes some attributes corresponding to the physical object. Objects in the conceptual model have been instantiated and mapped into a Java implementation; the specification of nodes in the navigational model were used to produce a set of JSP specifications (some of them are shown in Figure 8.a and 8.b).

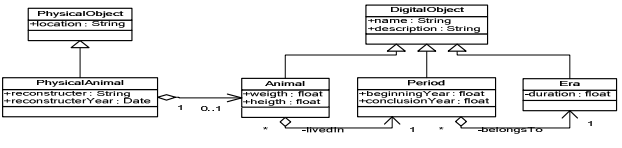

Fig. 6. Application and Physical Models for the Museum

Suppose that the user is in front of a Herrerasaurus (whose corresponding sensor emits the identifier "1"). The sequence diagram in Figure 7 shows how the process of generating the corresponding page proceeds.

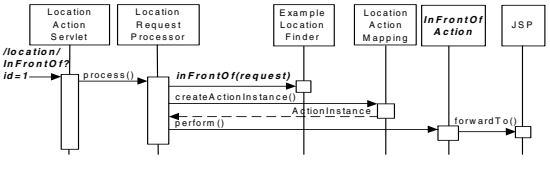

Fig. 7. Viewing information on a physical object

The *ExampleLocationFinder*, has the responsibility of finding the application objects which corresponds to the physical object with id "1". As a response the user receives the page shown in Figure 8.a, in which digital links are in the top pane and physical information and links in the bottom pane.

We used the Builder design pattern [4] to separate the construction of the two elements of the JSP (digital and physical information); this allows us to provide digital navigation (e.g. the user clicks on "Triasic Period") without changing the physical links exposed to the user. This means that while the user stands in front of the Herrerasaurus, the bottom pane does not change, as shown in Figure 8.b.

Physical links pose another implementation challenge. For example when the user selects "Diatrina" (another physical object in the museum), he should be instructed on the best way to reach the object, e.g. showing him a map as shown in Figure 9.

Fig. 8. a: Exploring a physical object, b: Pure Digital navigation

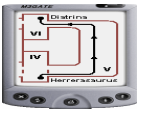

Fig. 9. The physical path to reach an object

In our extended MVC framework, the execution sequence is similar to the one in Figure 7. The only changes are the invocation of the request, the method that is executed in the *ExampleLocationFinder*, and the Action that performs the needed behavior.

We were able to develop the whole application by thinking in terms of a typical Web application, modeling it using the light extension of OOHDM and using predefined classes of our Struts extension.

5 Related Work

In [8] a comprehensive framework (HyCon) for deploying applications in which the hypermedia paradigm is extended to the physical world is presented. In [13] meanwhile, an object-oriented framework called HyperReal, based on the Dexter hypertext reference model is presented. We have followed a different strategy; instead of building a full-fledged, proprietary framework, we decided to extend a popular and widely used framework like MVC and its Struts implementation. In our first release, we decided to sacrifice facilities to keep our extension small and easy to use. The other important difference with respect to existing approaches is that our development framework is accompanied by a modeling and design approach. While the literature

has focused mainly on implementation and usability issues (See [8]), we think that modeling and design aspects are critical to assure quality and quality of use.

6 Concluding Remarks and Further Work

In this paper we have presented a design and implementation framework for developing physical hypermedia applications, i.e. those applications in which physical and digital objects are related using the hypermedia paradigm.

We have shown how to slightly extend the MVC metaphor to support location-aware controllers; we have then presented a Jakarta Struts implementation of our ideas, together with a simple proof of concept for a physical hypermedia in a Natural Sciences Museum. In this implementation, we have shown how to keep physical links available while navigating digitally.

We are currently working on several research directions. One of them relates with providing better modeling and design tools to express navigational structures. We are also experiencing further navigation issues. Physical navigation introduces new possibilities such as deviating from the suggested path (e.g. as shown in Figure 9) to explore other objects. We are researching on which software support must be provided by an application framework to support the developer job in keeping track of the user trajectory, suggest possible stops, etc. We are still compromised to closely follow the MVC metaphor, even in this kind of extensions. Many of the issues discussed here have been previously explored, for example in the hypertext community [1], though standard developing tools do not exist yet. We are finally porting our implementation to the .Net platform and studying usability issues.

References

1. Adaptive Hypermedia Home Page: http://wwwis.win.tue.nl/ah/
2. M. Douglas, D. Schwabe, G. Rossi: A software arquitecture for structuring complex Web Applications. Journal of Web Engineering 1 (1): 37-60 (2002)
3. F. Espinoza, P. Persson, A. Sandin, H. Nystrom, E. Cacciatore, M. Bylund: GeoNotes: Social and Navigational Aspects of Location-Based Information Systems. Proceedings of Third International Conference on Ubiquitous Computing (Ubicomp 2001), Springer Verlag, 2-17
4. E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns. Elements of reusable object-oriented software, Addison Wesley 1995
5. S. Gordillo, G. Rossi, D. Schwabe: Separation of Structural Concerns in Physical Hypermedia Models. CAiSE 2005: 446-459
6. S. Gordillo, G. Rossi, F. Lyardet: Modeling Physical Hypermedia Applications. SAINT Workshops 2005: 410-413
7. K. Gronbaek, J. Kristensen, M. Eriksen: Physical Hypermedia: Organizing Collections of Mixed Physical and Digital Material. Proceedings of the 14th. ACM International Conference of Hypertext and Hypermedia (Hypertext 2003), ACM Press, 10-19
8. F. Hansen, N. Bouvin, B. Christensen, K. Gronbaek, T. Pedersen, J. Gagach: Integrating the Web and the World: Contextual Trails on the Move. Proceedings of the 15th. ACM International Conference of Hypertext and Hypermedia (Hypertext 2004), ACM Press. 2004

9. S. Harper, C. Goble, S. Pettitt: proximity: Walking the Link. In Journal of Digital Information, Volume 5, Issue 1, Article No 236, 2004-04-07. Available at: http//jodi.ecs.soton.ac.uk/Articles/v05/i01/Harper/
10. A. Knight, N. Dai: Objects and the Web**.** IEEE Software, January/February 2002, 51-59,
11. 11.G. Krasner, S. Pope: A Cookbook for Using Model-View-Controller User Interface Paradigm in Smalltalk-80. Journal of Object Oriented Programming, August/September, 1988, 26-49.
12. OMG Model-Driven-Architecture. In http://www.omg.org/mda/
13. 13.L. Romero, N. Correia: HyperReal: A Hypermedia model for Mixed Reality. Proceedings of the 14th ACM International Conference of Hypertext and Hypermedia (Hypertext 2003), ACM Press, 2-9
14. D. Schwabe, G. Rossi: An object-oriented approach to web-based application design. Theory and Practice of Object Systems (TAPOS), Special Issue on the Internet, v. 4#4, October, 1998, 207-225.
15. F. Steimann: On the Representation of Roles in Object-Oriented and Conceptual modeling. Data and Knowledge Engineering 35 (2000) 83-106
16. The Struts Home Page: http://struts.apache.org/
17. The UML Home Page: www.omg.org/**uml/**