Systematic Development of 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 model-based approach for the development of physical hypermedia applications, i.e. those mobile (Web) applications in which physical and digital objects are related and explored using the hypermedia paradigm. We describe an extension of the Object-Oriented Hypermedia Design Method (OOHDM) and present an improvement of the popular Model-View-Controller (MVC) metaphor to incorporate the concept of located object; we illustrate the idea with 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 evaluate our approach and finally we discuss some further work we are pursuing.

1 Introduction

Physical Hypermedia (PH) extends the idea of hypermedia navigation to the real world; in a PH application, the (mobile) user can explore physical objects and their relationships with other physical or digital objects. Real world objects are augmented with digital information which allows that, when the user is in the vicinity of an object, he can access this information in his mobile device. Physical objects are considered nodes in a hypermedia network and therefore the information corresponding to such an object may include links and thus 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.

PH extends the idea of Location Based Software [35] by not only providing information access based on the user's location but also providing a sort of information structuring mechanism (namely hypermedia graphs) which allows users to explore

[†] This work has been partially funded by Project PICT 2003, Nro 13623, SeCyT

information with the well-known navigation paradigm of the World Wide Web, and unifies the ideas in hypermedia with some of the new concepts in ubiquitous and pervasive computing [31].

A simple example is a mobile tourist guide such as [6]. The user travels through the physical space (e.g. a city) exploring monuments, tourist spots, etc. When he is in front of a monument which is PH aware (i.e. the user can be sensed to be in its vicinity and the application has information on that object) he can read data about the monument in his mobile device, for example in a Web page; by following links, he can also explore some meaningful relationships which this monument has with other objects (its history, information about the architect, etc). In this way, the user traverses 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 [20] to visit the corresponding tourist spot. In this case, the software system should react to his intention to navigate by providing him a map showing the best way to access the target place and eventually some active help to guide him.

Notice that this application behavior, while somewhat similar to existing families of location-based services [35], is completely based on the well-known ideas of hypermedia navigation that became popular with the advent of the Web. It is not surprising then that the PH paradigm has been considered to be a good vehicle to integrate the Web and the world [19].

Developing PH applications is hard as we have to solve the problems typical of hypermedia and Web software [8, 9, 10] and besides we have to face the challenges in mobile and context-aware applications [1]. Summarizing, these are some of the engineering problems one need to consider for developing this kind of software:

- Modeling and Design Aspects: While modeling and design issues have been widely discussed in the Web Engineering community [3, 4] and some issues related with context-aware and ubiquity have been also the focus of research projects [23, 24], the problem of combining the real and the digital world has not been tackled from a modeling point of view so far. We must be able to clearly express the physicality of some objects, the nature of "walking" links, the semantics of physical navigation, etc. Modeling of context-aware functionality is also critical.
- Implementation and Deployment: These concerns involve a myriad of problems; while the implementation of Web software is now considered a well-known domain and there are mature frameworks for creating performing applications, software which deals with mobility issues must deal with some specific issues such as sensing the user's position in different coordinate systems (e.g. latitude/longitude, local location models, etc), mapping this position to the application's semantics (e.g. in terms of street names and numbers), relating the user's position with appropriate application objects (e.g. a building in this address), etc. At the same time we must decide which components will run in the server side and which in the client device; we have to deal with problems related with unreliable network connections, etc.

- User Interface Aspects: The user interface of applications running in small devices (like phones or PDAs) is difficult to design; they must simplify the user's task, therefore providing an easy to use interaction style, but at the same time they can not involve much information to avoid cognitive overhead. Usability is even more critical than in Web applications [22, 44].
- Placement Issues. Making real-world objects application aware, implies either enriching them with infrared or other kind of sensing devices (such as beacons [25]), or expressing their locations in terms of geographical coordinates (e.g. with GPS-based sensing is used). In either case, we have to deal with new problems: can we be sure that infrared devices will be "seen" by the user's device and also correctly identified? how do we deal with objects which can change their positions (e.g. in a museum or exhibition)?.

Many of these problems have been discussed and solved in the broader fields of mobile and context-aware computing [21, 34, 38]; some of them have been already dealt with in the hypermedia community (as discussed later in the Related Work section).

We have been working on different aspects of the PH applications' life cycle. In [16] we presented a modeling and design approach that allows a high-level specification of the intended functionality of a PH application. In [15] 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 describe the whole development approach; particularly, we describe 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 Java infrastructure [41] and therefore can be easily used by Web application designers.

The main contributions of this paper are the following:

- We describe the process of developing PH applications from a Web Engineering point of view and present an original model-based approach which covers the whole development life cycle.
- We show how to incorporate the concept of location in the MVC metaphor for interactive applications.
- We present an easy to use and practical framework for deploying PH applications.
- 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 survey related work in this area; in Section 3 we summarize our approach and in Section 4 we present the design framework. Section 5 is devoted to describing the development framework by both describing the conceptual extension to the MVC and one specific implementa-

tion. In Section 6 we present a running example and in Section 7 we briefly evaluate our approach. Section 8 concludes the paper with the presentation of some further work we are pursuing.

2 Related Work

Physical Hypermedia has its roots in the well-known fields of Hypermedia and, Augmented Reality and Ubiquitous Computing. The idea of adding Web presence to real world objects have been early explored in [25]; the use of these ideas to improve collaboration in social settings by using some kind of augmented reality have been explored in [12]; a good example of providing hypermedia functionality to the mobile user has been shown in [6]. However the term PH has been coined in [17] and later elaborated in [19]. Topos [17] allows manipulating and maintaining spatial relationships between mixed materials (digital and physical) in a tri-dimensional environment. It is a physical hypermedia system providing users with a digital workspace which is familiar to their physical workspace in which digital representations of physical material can be linked to pure digital objects and grouped in mixed collections.

HyCon [19] meanwhile is an object-oriented framework whose goal is to extend the Hypermedia paradigm with the manipulation of real World objects. In particular, it has been used to create context-aware hypermedia systems; it supports classical mechanisms of Hypermedia such as navigation, information searching, notations and implementations of guided tours in the physical (real) world. In addition to the facility to extend digital with physical objects, HyCon supports automatic collections of contextual and social information and works with different types of mobile devices.

In [32] meanwhile, an object-oriented framework called HyperReal, based on the Dexter hypertext reference model is presented. HyperReal allows building augmented reality applications, using the basic abstractions mechanism of the Dexter model [18]. HyperReal uses concept from adaptive and spatial hypermedia to integrate, in the same setting, virtual documents, 3D environments and the physical world to build mixed reality applications.

In [20] the authors describe proXimity, an approach for improving users' accessibility to real world objects by giving them a hypermedia presence. The main goal of proXimity is to extend the metaphor of links to the real world and to show how the basic ideas behind adaptive hypermedia can be applied to physical hypermedia spaces. The idea of "walk" the link in proXimity inspired our WLinks (described in Section 4.2).

Closely related with this field we can mention the UWA project [43], which has aimed to the design of ubiquitous Web software. The UWA design model, described in [23, 24], comprises different sub-models covering the various aspects of a typical customizable Web application: the core domain model, the user's context (which itself comprises the technical, location, social and other sub-contexts) and the adaptation rules. All these components are modeled using a light-weighted extension of the UML [42].

Our research is different from the work in [19, 20, 32] because it has been initially oriented towards the modeling and design of physical hypermedia applications more

than the run-time support. We have emphasized the need to produce implementation-independent descriptions of PH applications; while [19, 20, 32] describe software substrates for implementation, we aim to produce design documents that can be later implemented in any run-time setting. Once the intended structure and behavior of a PH application has been specified using the extended OOHDM, it could be implemented for example using HyCon or HyperReal. From the implementation point of view, we believe, that for these technologies to become mainstream, some standardization at the level of implementation architectures is needed. To fulfill this objective we made a proof of concept by both extending a well-known standard paradigm for interactive applications, the MVC and materializing this extension in a popular framework for Web software, Struts. Different from UWA we decided to keep the design approach compact and to rely on the mature OOHDM by devising a light extension which supports physical objects and walking links.

3 Our Approach in a nutshell

In this section we summarize the most important aspects of our approach and give an overall view of the whole development process. In Figure 1 we show a graphical representation of Physical Hypermedia and its relationship with Web software.

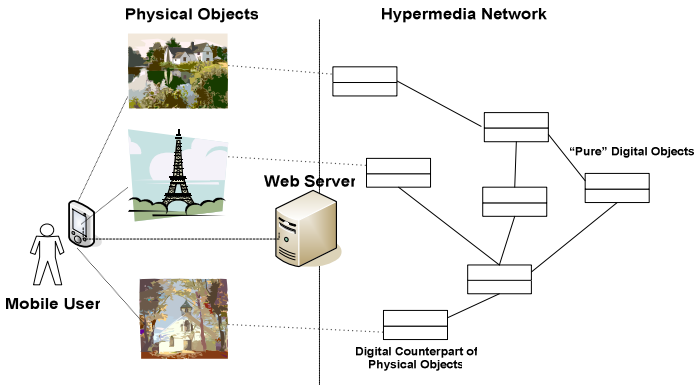


Figure 1: Physical Hypermedia and the Web

As previously discussed, researchers have emphasized the feasibility of the PH paradigm by building software infrastructures that support the ideas underlying PH [17, 19, 32]. Our objective meanwhile is to provide a modeling and design approach and a development framework by reusing and extending existing methods and software. We have carefully analyzed the main requirements for a development approach; in summary it should:

- 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,
- Be easy to learn for Web developers; this means that it should not depart too much from existing Web software infrastructures.

Finally, applications built using the development 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.

3.1 Design Issues

We define a PH application as a hypermedia application (i.e. the access to information objects is performed 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. Objects of interest can also move and eventually the user may not be mobile and explore objects which are put in his interaction range (e.g. a production line in a factory in which application objects like products move through the line and the user is standing in some place). A good example of this kind of application is presented in [17].

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 (see Figure 1).

We chose to extend the Object-Oriented Hypermedia Design Method (OOHDM) [37] by incorporating the concept of physical objects and "walking" navigation [20]. 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. In Figure 2 we show the basic OOHDM approach together with the hot-spots in which PH specific features are modeled.

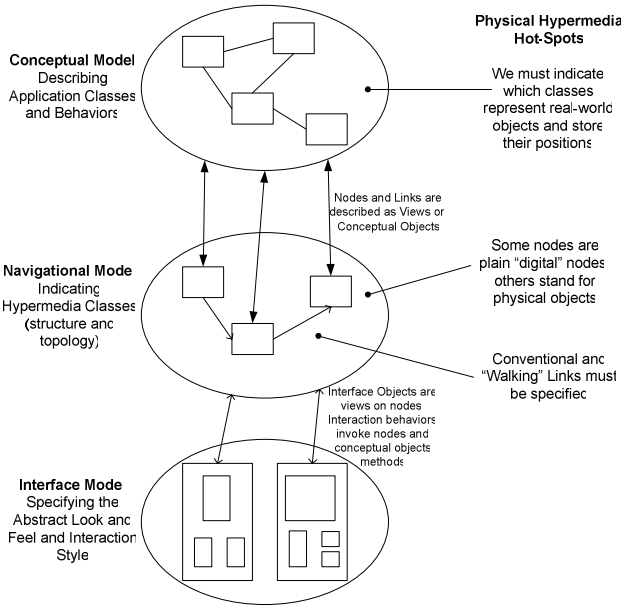


Figure 2: The OOHDM approach for Physical Hypermedia

3.2 The implementation approach

One of the key requirements in our research was to create a development framework which was straightforward to use for developers of Web software and also not to tight to a specific development platform (such as .Net or J2EE). We therefore decided to rely on the MVC metaphor, which is widely supported in different platforms and then to instantiate our solution in each specific environment. The Model-View-Controller [27] 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 [26].

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 [11], and finally it is a well-known metaphor, used in every modern middleware platform for Web application development. Our extension basically enhances the controller component of the MVC making it location-aware, and therefore being able to respond to requests containing physical locations.

In the following sections we describe our approach in more detail; for the sake of conciseness we concentrate on the server-side. Details on client side aspects 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. When some functionality has to be assumed in the client side, we will make it explicit in the paper.

4 The Design Approach

As previously described we have adapted the OOHDM development framework by including some features which are specific to PH software. We limit our discussion to these novel features and ignore the use of basic OOHDM primitives to improve the design of PH applications. Further information on these aspects can be read in [15].

4.1 Application Modeling

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 [42] as in the standard OOHDM approach.

In the second layer, and for these classes whose instance may have physical presence, we describe their physical counterparts. Physical classes are described as roles, in fact decorations [39, 14] of base application classes and contain, by default, the objects' location, geographical relationships among them, and the basic 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 3, we show these two layers.

The two physical classes (*PH_A*, and *PH_B*) wrap application classes *A* and *B*, thus obliviously adding them their physical properties. As shown in Figure 3, concrete physical classes inherit location properties which are further decoupled from the abstract class *PhysicalObject* and included in abstract class *Location* and their subclasses. Details about location modeling, location properties and behaviors can be read in [28].

Physical classes may also exhibit geographical relationships such as *near*, *beside*, etc; some of them will be expressed explicitly between specific objects, others will be calculated on the fly when needed.

Physical classes may also have attributes which describe the physical object and which don't make sense in its digital counterpart: in a painting we can describe information about its actual placement, surroundings, data about recent restorations which may be unnecessary in a "pure" digital access (other examples are shown in Section 6). This layer may also contain objects which do not make sense in the digital world, such as streets, corridors, and other "pure" physical objects.

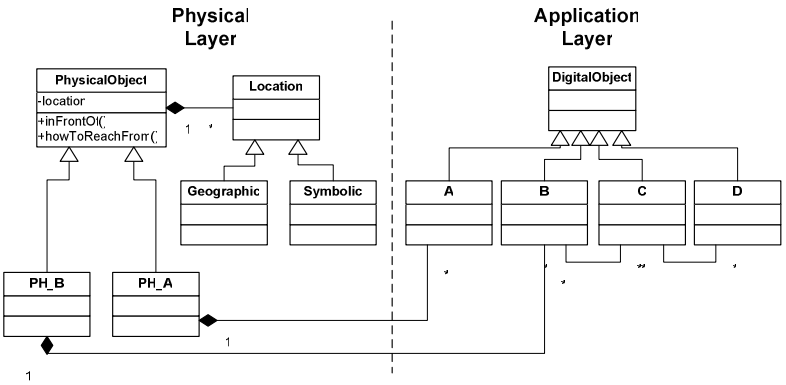


Figure 3: The products of Application Modeling

By separating physical from digital properties we add modularity to our design and therefore simplify application evolution. For example, a navigational model can be built by solely considering digital objects, thus yielding a "conventional" Web application. As technology evolves and we have new possibilities for object sensing, we can add new Physical classes without disrupting base classes with the new, physical, features.

Our approach also allows greater flexibility with respect to the process of adding physical presence to application objects. Roles (and their lower-level counterparts, object decorators) are handled at the object instead of the class level. The schema in Figure 3 implies that we can decorate individual objects with physical properties. This means that a particular application class (e.g. *Church*) can have some instances which can be explored physically, i.e. they have a location (which means, of course, that there are ways to sense the user in front of them); however, there may be churches which can not be explored physically, for example because they are in restoration, they are far from the place in which the application runs or simply they do not exist any more (though we may have digital information describing them).

Treating the physical counterparts of application objects as roles, that they can play, is a better solution than defining specializations of application classes; in this latter case we pollute the application class hierarchy with location issues and besides an object belonging to a "physical" sub-class can not loose this peculiarity for example if the object is not longer available physically because physical properties are defined at a class level. In the role-based solution, an object can play the physical role or not dynamically because to cease playing the role means just changing the value of an attribute; this is important if we want an object to be temporary not accessible in the real-world (e.g. an artwork or monument in restoration).

4.2 Navigation Design

The navigation model specifies which nodes the user will explore and the links connecting these nodes. Nodes are defined as views on application objects or on their physical counterparts and contain the information to be displayed to the user. As physical objects act as decorators of digital objects, a node which is built on a physical object has access to both physical and "pure" digital information (as a consequence of a property of Decorators, see [14]). This allows that all nodes are built similarly, i.e. there may be no essential difference among viewing digital or physical application objects (an interesting exception can occur with links outgoing from a physical object as described later).

Links meanwhile can be digital or physical. A digital link allows "conventional" navigation (i.e. as in the Web), which means that the semantic of digital links is similar to a conventional link in hypermedia, i.e. when the link is activated, the target node is opened. Physical or "walking" links (WLinks) express a relationship in which the target object is physical, and therefore exploring the object implies that the user must change his current position. When the user activates a WLink, he is indicating his intention to explore the target; the system response may be a map or plan to guide him to the target object. He then has to "walk" through the link and when he arrives to his destiny, he is sensed and the corresponding node is opened. Physical links might be derived from both conceptual and geographical relationships. For example, while we are exploring a monument we may have links to other objects of interest which are "near" (which means a geographical relationships) or to other spots which have a stronger semantic relationship, for example a monument which was built by the same architect.

WLinks are not a particular meta-class of OOHDM links; instead we decided to engineer them by refining the default behavioral response of links, which in OOHDM is defined as a separate class (using the Strategy pattern [14]). This solution, described with detail in [15], allows defining the "walking" nature of a link in an instance basis. In Figure 4, we show the relationship between two navigational classes *Artist* and *Church* and a link *workedIn* which allows navigating from the description of the artist to those churches in which we can see his artworks. The <w> tag indicates that instances of this link class may be WLinks; in this way we don't need to define a different class for targets (churches) which are not accessible physically (e.g. they have been destroyed) though we still have information on them. Of course it is possible to define a digital link which connects this two node classes; in this case the link will be traversed "just" digitally.

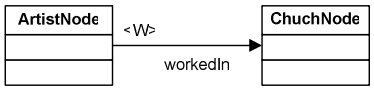


Figure 4: Walking Links vs. Conventional Links

Similar to the base OOHDM approach we can define navigational contexts, i.e. set of nodes with some common property, according to geographical properties to allow the user to explore them physically. For example the context *Monuments near the river Seine* contain all instance of navigation class Monument whose corresponding physical counterpart satisfies the predicate *near* with the object *Seine*. The context *Churches by Neighborhood* indicate all Churches which are located in the same neighborhood. In this case the property fulfilled by all churches is not necessarily represented with a geographical relationship (but perhaps with an attribute which contains the name of the neighborhood); traversing the nodes in the context implies traversing WLinks. Physical Guided tours as those shown in [6] are also described as navigational contexts. More examples on navigational contexts related with physical properties can be read in [15].

Once we have specified the application and navigational models, we can proceed with the abstract interface model which, for the sake of conciseness, we don't describe in this paper; details on our interface primitives can be read in [29, 37]. The application can then be implemented; though OOHDM does not prescribe any particular runtime setting we next show our strategy to implement PH applications on top of the MVC model.

5 A MVC framework for Physical Hypermedia

A PH application has two main differences with a conventional Web application: the first one, which implies the strongest requirement for a development framework relates with the mechanism for opening pages. While in Web software, nodes (i.e. pages) open as the result of the triggering of a link, in PH the node corresponding to a physical object opens when the user stands in front of that object; in other words, as the result of the user movement, a node might need to be opened. The second one is that the triggering of WLinks does not imply that new information is presented to the user, but that a path to the target object should be provided by the system. In both cases, there is a request to the application: the first one should be generated (implicitly, via a push or pull mechanism) when the user is sensed to be in front of a meaningful object, the second one has to be mapped to the process of triggering a link. The solution presented in this paper involves the extension of the MVC architectural pattern to support some kind of location-based behavior. A slight extension of client software (e.g. the browser) is also necessary to provide the corresponding information on sensed data.

As explained before, in the MVC metaphor, the responsibilities of the application are divided in three coarse grained components: the Model which contains the application objects, the View which deals with interface issues and the Controller which handles interaction. In a simplified interaction, the user perceives the information of the model; this information is shown in interface objects (which comprise the view). When he exercises the interface (e.g. clicking on a button, selecting an option, filling a form, etc), the interface event is handled by a controller object, which in turns decides which is the model object that will provide the application's response. Once the model returns control to the controller, it generates a new view and the cycle begins again.

As a consequence, in the Web applications domain, controller objects handle part of the application logic flow.

We have thoroughly analyzed the standard responsibilities and concerns of each one of the MVC's components; we decided to extend the scope of the Controller component to support location-aware requests, those requests which are originated in the user's movement, and to manage the triggering of physical links.

The main rationale behind this decision is that we found that both the View and the Model components can support PH functionality, without modifying their essence. Notice that following our approach, a software engineer builds a slightly more complex model (which contains physical classes); however the responsibilities of the model component are still the same as in the standard MVC.

As explained before, 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. We next describe some high-level issues in our extension.

5.1 A conceptual view of the Location-Aware MVC

The two main components of the controller, in a Web setting, are according to [26]: the *InputController* and the *ApplicationController*. Their main responsibilities are the following:

- The *InputController* extract relevant request information and cooperate with the *ApplicationController* component to specify which *Action* will be executed, invoking it in the appropriate context.
- The *ApplicationController* component coordinates the application logic flow, the error handling, maintains long-terms states and decides which view will be shown.

Our extension involves mainly modifying 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 with the standard MVC, we avoided changing this component but we introduced a new separated 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 eventually involve other issues, we devised a *DispatcherController* as a Façade [14] 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 5. In the case of requests which do not need any pre processing, the *DispatcherController* delegate control directly to the *ApplicationController.*

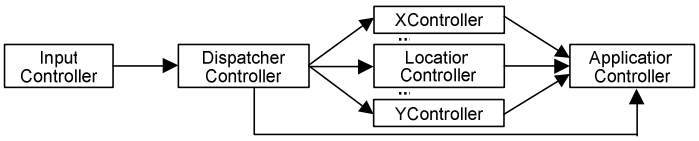


Figure 5: A new Controller Schema

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 6.

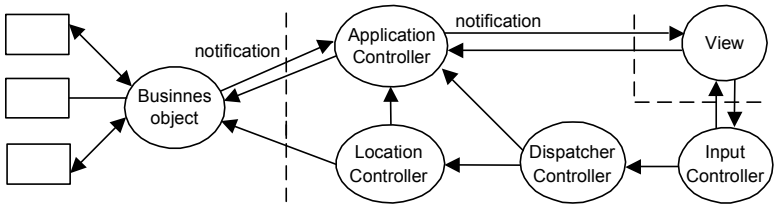


Figure 6: Extended MVC for Physical Hypermedia

The *LocationController* is activated to analyze a request which implies location. Therefore, it will have to deal with that location using the same location techniques (and models) which the actual application uses. Once the location has been analyzed the processing proceeds as an ordinary request. For this, the *LocationController* gives control to the *ApplicationController*, which itself works oblivious from the nature of the request.

To make this discussion more concrete we detail one specific materialization of this idea: and extension to the Struts [41] implementation of the MVC.

5.2 Adding Location-Awareness to Struts

There are basically three standard ways of extending the Struts framework:

- Creating a *PlugIn*; this option is used if we want to add some customized business logic at the beginning or end of an application.
- Create a *RequestProcessor*, if we want to execute some business logic during the processing of the request.
- We can also extend the *ActionServlet*, if we want to execute some business logic either at the beginning or end of an application or during the processing of the request. However, we should use the *ActionServlet* only for those cases in which neither *PlugIns*, nor *RequestProcessors* can fulfill the intended

functionality. In the case of our *DispatcherController*, none of these extensions can provide its functionality.

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*.

This functionality corresponds to the case in which the *DispatcherController* delegates control to either *LocationController* or to the *ApplicationController* in Figure 6.

Both, the configuration of the new *Servlet*, and the format of the location-compliant URL are configured in the Struts file web.xml as shown in Figure 7.

```
<web-app>
  …
  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-
    class>org.apache.struts.action.ActionServlet</servlet-class>
  </servlet>
   <servlet>
    <servlet-name>location</servlet-name>
    <servlet-
    class>org.apache.struts.location.LocationActionServlet
    </servlet-class>
  </servlet>
  …
  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>location</servlet-name>
    <url-pattern>/location/*</url-pattern>
  </servlet-mapping>
  …
</web-app>
```

Figure 7: Description of Struts configuration file

We next examine how to carry out the *LocationController*'s task. PH 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 locatiosystem to obtain a correct result. To achieve this goal we decoupled 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.

The *ConcreteLocationFinder* is configured in the file web.xml as an additional initialization parameter of the servlet *LocationActionServlet*. This is specified as shown in Figure 8.

```
<web-app>
  ...
  <servlet>
    ...
    <init-param>
      <param-name>finder</param-name>
      <param-value>ConcreteLocationFinder</param-value>
    </init-param>
    ...
  </servlet>
  …
</web-app>
```

Figure 8: Specifying a ConcreteLocationFinder

Concrete sub-classes of *LocationFinder* must implement at least two methods: one to identify the physical object 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. The abstract specifications of these methods (in *LocationFinder*) are as follows:

```
public abstract Object inFrontOfObject(HttpServletRequest obj);

public abstract Object howToReachFrom(Object from, HttpServle-
tRequest to);
```

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 as shown in Figure 9.

```
<web-app>
  ...
  <servlet>
    ...
    <init-param>
      <param-name>objectName</param-name>
      <param-value>Name of the physical object</param-value>
    </init-param>
    <init-param>
      <param-name>travelName</param-name>
      <param-value> Name of the travel object</param-value>
    </init-param>
    ...
  </servlet>
  ...
</web-app>
```

Figure 9: Specifying session's names for physical and the travel objects

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 as shown in Figure 10.

```
<struts-config>
  ...
  <controller  processor-
  class="org.apache.struts.location.LocationRequestProcessor" />
  ...
</struts-config>
```

Figure 10: Specification of LocationRequestProcessor

In Struts there must be a one to one correspondence between a control servlet and a *RequestProcessor*. Therefore, the existing Struts *ActionServlet* will still correspond with the *RequestProcessor* to assure compatibility, and the *LocationActionServlet* will correspond with the *LocationRequestProcessor*.

The *LocationRequestProcessor* collaborates (with the mediation of the *LocationActionServlet*) with the concrete *LocationFinder* to implement the pre-processing of the request. *LocationRequestProcessor* is a sub-class of *RequestProcessor*, and redefines the following method:

```
protected boolean processPreprocess(HttpServletRequest re-
quest,HttpServletResponse response)
```

In this method we process the request, determining if it involves a search for a path or for an object. Depending on the nature of the search, it will invoke the appropriate method in the corresponding *LocationFinder*.

To determine how the controller establishes whether the request involves an object or a path, we use *ActionMappings*, in which all information about the request is stored. Struts allows defining *ActionMappings* with additional information besides the default one, provided by the framework itself. We therefore defined a customized *ActionMapping,* namely, the *LocationActionMapping*. The *LocationActionMapping* will be associated with the *LocationActionServlet*, while the *ActionServlet* will be still related with the default *ActionMapping* provided by Struts.

The *LocationActionMapping*, besides the default properties, defines a new one, *locationEvent*, which allows to specify which of the finder methods will be invoked by the *LocationRequestProcessor.* We indicate this as shown in Figure 11.

This property is defined in the configuration file location-struts-config.xml, (the configuration file of *LocationActionServlet*). It is worth noticing that this property is mandatory, because without this information the request can not be processed.

All configuration files are related with DTDs, determining the internal structure of these files. When adding a new property in location-struts-config.xml, we need to define a new DTD (location-struts-config_1_1.dtd) including this new property, in the file structure.

```
<location-action-mappings>
  ...
  <location-action …
            locationEvent=" inFrontOf " … />
  <location-action  …
            locationEvent="howToReachFrom" … />
  ...
</location-action-mappings>
```

Figure 11: Specification of the locationEvent's property in the configuration file

To change the file structure it is necessary to consider that parsing rules of this file change; therefore we define new rules in the *LocationConfigRuleSet* allowing to parse the location-struts-config.xml file.

In summary the relationship between the MVC elements and those that arise from the extension of Struts can be represented as shown in Figure 12:

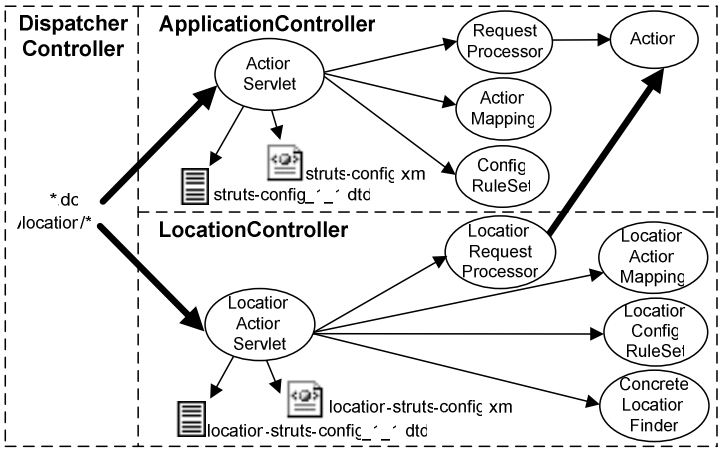


Figure 12: The complete Struts extension for handling location data

The relationships between the conceptual specification and the Struts implementation is determined as follow:

- The behavior of the *DispatcherController* is solved using the structures of the URLs.
- The functionality of the *ApplicationController* is solved using the Struts Framework.
- Responsibilities of the *LocationController* are obtained by defining components' specializations of the Struts Framework which in turn, collaborate with new components.

Figure 13 shows how control flows through the elements involved in solving a request.

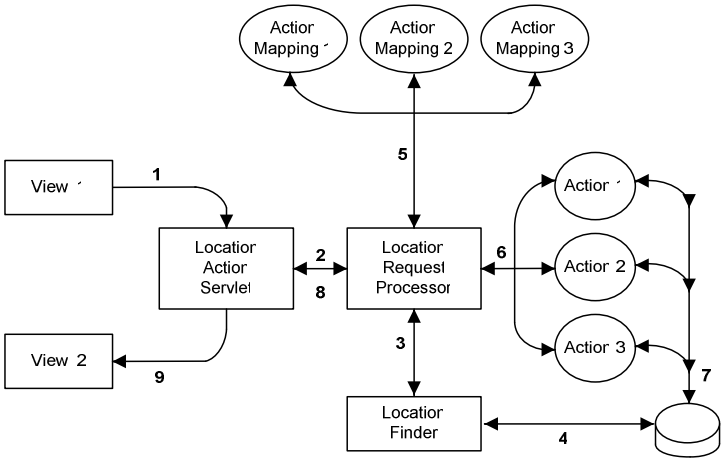


Figure 13: The control flow of a request that involves location

In Figure 13 the following event/transitions occur:

1 The request is handled by the *LocationActionServlet*.
2 *LocationActionServlet* delegates control to the *LocationRequestProcessor*.
3 *LocationFinder gets the control.*
4 *LocationFinder* localizes the physical aspect and returns control to the *Location-RequestProcessor.*
5 *LocationRequestProcessor* queries the corresponding *LocationActionMappings* to determine the corresponding *Action*.
6 *LocationRequestProcessor* gives control to the adequate *Action*.
7 The *Action* realizes the corresponding model operations and returns control to the *LocationRequestProcessor*.
8 *LocationRequestProcessor* gives control to *LocationActionServlet*.
9 Control is forwarded to a new view.

One of the advantages of our approach is that the framework is very easy to instantiate for a specific application (i.e. a particular model in the MVC). We only need to:

- Specify a Finder to get physical objects and to compute paths between two physical objects; specify the name that will be used to store them in session for physical objects and paths.
- Create the application *Actions* and configure them in the corresponding files depending whether they are related with location issues or not (location-struts-config.xml and struts-config.xml).
- Create the corresponding JSPs depending the information we need to show.

6 A Running Example

To illustrate our ideas, we next present a simple example in which we show the modeling and design approach and the instantiation of the framework in practice.

As a proof of concept we have instantiated the framework in a Natural Sciences Museum, which contains skeletons of prehistoric animals. Our prototype uses a particular sensing device (infrared sensors) and location model (symbolic). However, most design decisions can be easily understood while analyzing the example independently of these specific features.

6.1 Design

We first produced a conceptual model, including the location enrichment. In Figure 14, we show a simplified conceptual model 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 symbolic location model. The class *PhysicalAnimal* also includes some attributes corresponding to the physical object. Objects in the conceptual model have been instantiated and mapped into a Java implementation which finally represents the model of the MVC triad; the specifications of nodes in the navigational model were used to produce a set of JSP specifications (some of them are shown in Figure 20.a and 20.b).

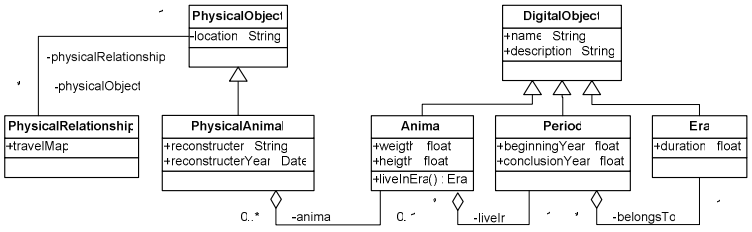


Figure 14: Application and Physical Models for the Museum

The class *PhysicalRelationship* model the path between different physical objects. For simplicity, in this example the path is expressed as a *travelMap* though it could be enhanced to be further more elaborated. Notice that *travelMaps* are objects with no digital counterpart in the model.

An oversimplified navigational schema is presented in Figure 15. Nodes for animals show the same attributes defined in the conceptual model; they are described as views on class *PhysicalAnimal* in Figure 14. All links pointing to animal nodes are tagged as WLinks; when populating the information base one could treat one instance of those classes as a digital link (if the animal's skeleton is not in the Museum)

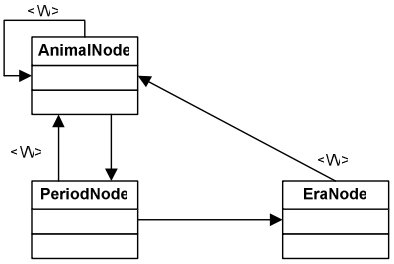


Figure 15: Navigational Schema of the Museum example

6.2 Instantiating the framework

For the sake of understanding we separate the configuration step from the explanation on the dynamic aspects.

6.2.1 Configuration Issues

The following steps are necessary to create a running application using our framework: first, we need to create a Struts project with the framework as the library; second we need to implement a specific finder (we will call it *ExampleLocationFinder*) with two outstanding methods: *inFrontOfObject* and *howToReachFrom*. The first one identifies the object the user is facing, the second searches a path between two objects.

In our prototype each physical object (skeletons of animals in the museum) has been assigned a code and the infrared sensor placed in the object emits a signal with that code. For example the Herrerasaurus emits "1", while the Diatrina emits "2". The implementation of *inFrontOfObject* returns the associated object by querying each object with the code which has been received in the request.

The web.xml file is configured as shown in Figure 16.

In the specification of Figure 16, "example.finder.ExampleLocationFinder" jeans that in the package example there is a package named finder containing the Class *ExampleLocationFinder*. *PhysicalObject* and *physicalTravel* are the names in which a physical object and path are stored in the session.

The configuration file struts-config.xml indicating actions related with digital information looks as shown in Figure 17.

```
<web-app>
  ...
  <init-param>
    <param-name>finder</param-name>
    <param-value>example.finder.ExampleLocationFinder</param-
    value>
  </init-param>
  <init-param>
    <param-name>objectName</param-name>
    <param-value>physicalObject </param-value>
  </init-param>
  <init-param>
    <param-name>travelName</param-name>
    <param-value>physicalTravel</param-value>
  </init-param>
  …
</web-app>
```

Figure 16: Specifying the initialization parameters of the application

```
<struts-config>
  ...
  <action-mappings>
    ...
    <action  path="/DigitalRelationship"
             type="example.actions.DigitalRelationshipAction"/>
    ...
  </action-mappings>
  ...
</struts-config>
```

Figure 17: The configuration of file struts-config.xml

The attribute path allows specifying how the action will be invoked in the corresponding JSP. The attribute type indicates the *Action* class which will handle the request, in this case *DigitalRelationshipAction* (in package actions).

The configuration file location-struts-config.xml, contains the configuration of physical information as shown in Figure 18.

Path and type attributes have the same semantics than in Figure 17. The attribute *locationEvent* indicates which method must be invoked in the Finder.

Package example contains another package namely locationActions in which classes *InFrontOfAction* and *HowToReachFromAction* are specified. The first one is used to handle requests when the user is in front of a physical object, the second one is used when walking a link. Classes *InFrontOfAction*, *HowToReachFromAction* y *DigitalRelationshipAction*, are subclasses of *Action*.

Finally we have to specify the JSP according to the information we want to show; in this case we need to define one for presenting digital information and another one to present the path to an object.

```
<struts-config>
  ...
  <location-action-mappings>
    ...
    <location-action  path="/InFrontOf "
              locationEvent=" inFrontOf "
              type="example.locationActions.InFrontOfAction"/>
    <location-action  path="/HowToReachFrom"
              locationEvent="howToReachFrom"
              type="exmple.locationActions.HowToReachFromAction"
              />
    ...
  </location-action-mappings>
  ...
</struts-config>
```

Figure 18: The configuration of the location Actions

6.2.2 Dynamic Aspects

To complete the description of our running case, we will show how things work in the framework as the user moves. We assume that the client is running the needed software to map the signal of a sensor into a browser request; this functionality can be performed by an applet in the JSP, an ActiveX component or a java application embedded in the browser.

Suppose that the user is in front of a Herrerasaurus (whose corresponding sensor emits the identifier "1"). The client software generates a request of the form:

```
…/location/InFrontOf?id=1
```

In Figure 19 we show how this request is handled. When the request arrives, the framework detects (from the configuration file) that the URL corresponds to a location and therefore it gives control to the *LocationActionServlet*. This delegates in the *LocationRequestProcessor* the pre-processing of the request. The *LocationRequestProcessor* invokes in the *ExampleLocationFinder* the method *inFrontOf(HttpServletRequest req).* This method searches the physical object which corresponds to the code "1" by collaborating with model objects. The *ExampleLocationFinder* detects the corresponding physical object (in our example the "Herrerasaurus") and stores it in the session under the name *physicalObject*.

Once that the request pre-processing is finished, control returns to the *LocationRequestProcessor* which queries the *LocationActionMapping* to detect which *Action* corresponds with the URL.

According to the configuration of location-struts-config.xml, this request must be handled by *InFrontOfAction.* Afterwards the corresponding JSP is generated (Figure 20.a), it retrieves the information from the *physicalObject*, which has been stored in the session.

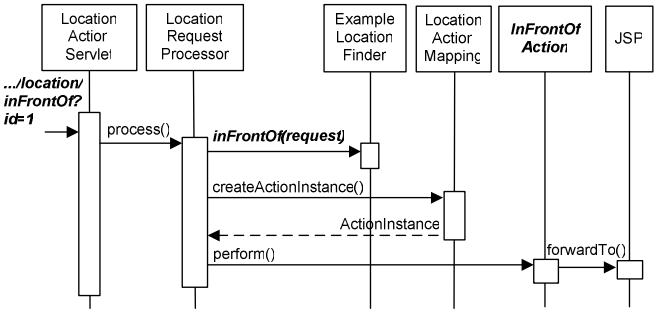
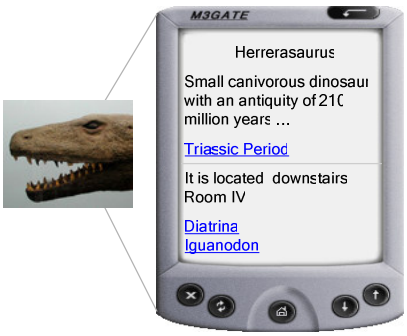


Figure 19: Viewing information on a physical object

The generated JSP (Figure 20.a) contains in its upper part digital information and links; in the bottom we have placed physical information and links. Though this separation is arbitrary and can vary for different applications, it is useful when one wants to preserve physical links (those which depend on the user's location) while the user navigates through physical information; this allows us to provide digital navigation (e.g. the user clicks on "Triassic Period") without changing the physical links exposed to the user. This means that while the user stands in front of the Herrerasaurus, he can navigate digitally and the bottom pane does not change, as shown in Figure 20.b.



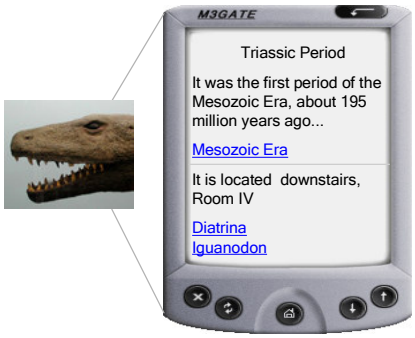


Figure 20.a: Exploring a physical object **Figure 20.b:** Pure Digital navigation

The views in Figure 20.a and 20.b are obtained by using several collaborating design patterns during framework instantiation, namely Composite View, View Helper and Builder. First, we applied the Composite View pattern [7], which is a specialization of the Composite pattern [14]. According to this pattern each view is itself composed of sub-views (either composite or atomic). In this case the page is composed of two sub-views: one for the digital and the other for physical information. We use the View Helper pattern [7] to express that each view delegates code processing to helper classes. Helpers in our case are custom-tags which have been used in our framework to present both physical and digital information. Finally, Builder [14] is a creational

pattern used to represent the JSP which is visualized by the user from the digital and physical information which are stored in the session. Physical information is stored under the name *physicalObject*, while digital information is stored after the execution of the *Action* under the name *digitalObject*. Figure 21 shows a schema of this collaboration of pattern instances.

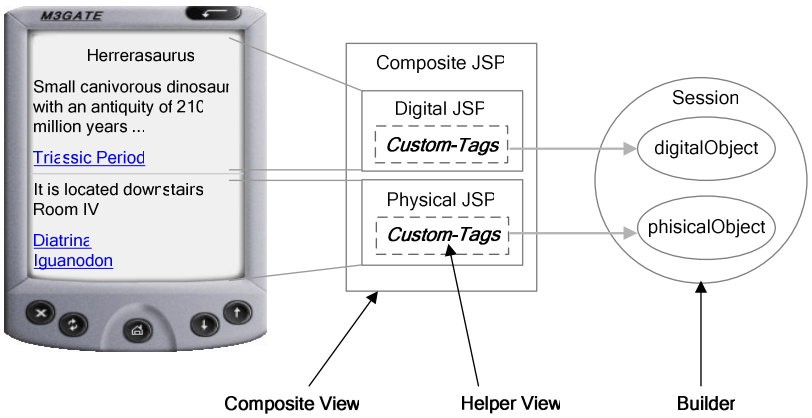


Figure 21: Using patterns during framework instantiation

As previously discussed 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 22.

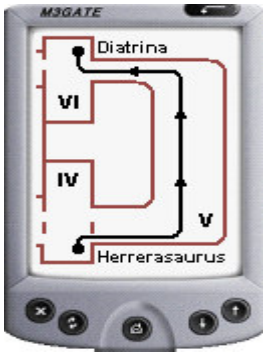


Figure 22: The physical path to reach an object

In this case, the request is generated by the user when he chooses a physical link. The URL which represents this request has the form:

…/location/HowToReachFrom?id=2

The control flow is similar to the one in Figure 19 except that the *LocationRequestProcessor* invokes (in the *ExampleLocationFinder)* the method *howToReachFrom(HttpServletRequest request)*. This method will find the path between the current physical object and the object which is referred in the link (in this case the one with identification 2, Diatrina). The *ExampleLocationFinder* detects the physical path which is stored in the session under the name *physicalTravel*. Similar to the former case, control returns to the *LocationRequestProcessor* which queries the *LocationActionMapping* to detect which *Action* corresponds with the actual URL. In this case the file location-struts-config.xml, indicates that it must be handled by *HowToReachFromAction.* The corresponding JSP is generated, showing the path to Diatrina (Figura 21). A sequence diagram with the previous control flow is presented in Figure 23.

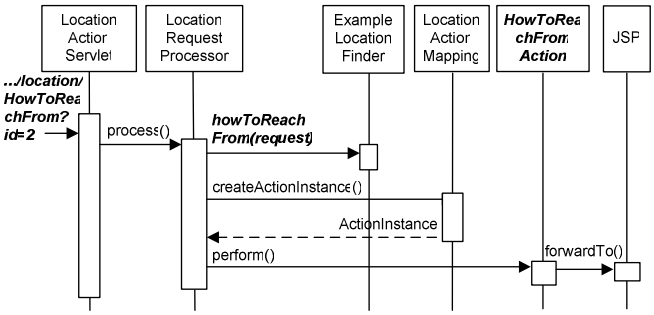


Figure 23: Viewing information of how to reach a physical object

Digital links are processed a bit differently. In Figure 24 we show the sequence diagram corresponding to the processing of the request generated when the link Triassic Period is triggered. Notice that this sequence does not imply any pre-processing as we are dealing with a pure digital request.

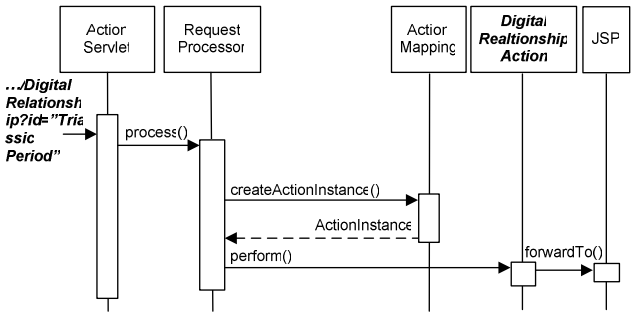


Figure 24: Viewing information as a result of digital navigation

Summarizing the previous example, 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 just a set of pre-defined classes of our Struts extension.

7 Discussion

While our approach is original and it does comprise the whole life cycle of a PH application, we still believe that there is place for improvement, both regarding the design method and the implementation setting.

The evaluation of a development approach involves many different aspects, some of them rather technical, others more related to the underlying process. We limit our evaluation to the specific foci of the paper: the design approach and the supporting framework. We do not address interface or usability issues because they are beyond the scope of our current research; however the interested reader can find a thorough analysis of some of these issues in specific projects in [17, 19].

7.1 Evaluation of the design approach

To evaluate a design approach we must analyze its goals, modeling primitives, products, how easy it can be learned, the supporting tools, etc. A good framework for comparison of methodologies for ubiquitous applications (which includes OOHDM) can be found in [23]. In the past we have used the bare OOHDM model to specify personalized applications. In [36] we showed that using the base OOHDM query language for defining nodes and links, and applying some advanced object-oriented principles, we don't need to extend the modeling vocabulary to cope with personalization and adaptation functionality. In this sense we consider that the base OOHDM is adequate for applications which involve some kind of customization.

Our current extension of the OOHDM approach for PH is a light-weighted extension of OOHDM and therefore it is easy to learn for OOHDM designers as it introduces only some new modeling primitives. As an example, roles (used to specify physical aspects) are well-known mechanisms for achieving dynamism in assigning behaviors to objects and their use is considered a good practice in object-oriented design [39]. Therefore, as the conceptual modeling armory is basically the same as in the native OOHDM approach, it does not need further analysis.

WLinks meanwhile imply an important departure from the concept of a link in hypermedia. Their behavior is different from the usual link's behaviors. However, their implementation in OOHDM is straightforward and does not require important modifications in the underlying meta-model; as a consequence, the learning process is not substantially modified. Once understood, the documents we produced while modeling the navigational aspect of a PH application are concise and not much complex than a usual OOHDM model.

We have not yet developed a tool for model-based derivation of running applications from the OOHDM design documents. Some other further extensions which will improve the method are discussed in Section 8.

7.2 Framework Evaluation

A critical consideration when developing the extended Struts framework was to keep the existing implementation, and use the basic Struts' extension mechanisms to add location-awareness. In this way, we preserved all benefits of the base framework and simplified the learning process. Besides, the migration of a "conventional" Web application to the new framework (for example to include physical objects and user's mobility) is straightforward; we just keep the base classes and obliviously add the machinery for location description and objects finding.

Table 1: Evaluating L-Struts

Evaluation Criteria	Results
Customs Tags	L-Struts allows the use of Struts tags libraries and provides a set of specific customized tags to support physical information presentation.
Validation	L-Struts keeps the possibility (available in Struts) of using Common Validators.
Testability	The StrutsTestCase can be used during the development of L-Struts applications.
Post and Redirect:	Double-posting is managed by L-Struts both for physical and digital requirements.
Internationalization	L- Struts (as Struts) allows local ResourceBundle.
Page Decoration	The use of tiles is possible in L-Struts, allowing the use of visualization Templates to separate digital from physical information and links.
Tools (particularly IDE)	Any IDE which supports Struts, could be used to create a L-Struts application.
Development	L-Struts provides the needed support for physical and digital requirements.
Pattern MVC	L-Struts extends MVC for supporting physical hypermedia applications.
Configuration	L-Struts adds complexity in the configuration steps, as physical features must be specified separately from digital ones. Though the configuration is modular, the process implies some additional difficulty.

As a summarized evaluation, we use the framework in [40] by adapting it to our location-aware Struts; the results are shown in Table 1, where our framework is called L-Struts. We are also exploring alternative implementations and analyzing with more detail client-side issues as described in Section 8.

8 Concluding Remarks and Further Work

The construction of physical hypermedia applications presents several challenges to the developer; some of these challenges had been early indicated in the broader field of ubiquitous computing [1]; others have been reported in the seminal work presented in [17].

In this paper, we have presented a design and implementation framework for developing physical hypermedia applications. We have detailed a light extension to the OOHDM design approach which adds physical objects and walking links to the basic set of modeling primitives.

We have also shown how to slightly extend the MVC metaphor to support location-aware controllers; we have then presented an implementation of our ideas on top of the well-known Struts framework; we presented a simple proof of concept for a physical hypermedia in a Natural Sciences Museum.

Our research in this area continues in several directions; we are now:

- Studying how to provide better modeling primitives without compromising simplicity; particularly we are interested in obtaining more expressive navigational models in which we can indicate which links are maintained visible while navigating physically or digitally. We are also studying the use of UML-like stereotypes to improve the notation of WLinks. The notation for context-aware navigation is rather limited in OOHDM; it must be improved for this specific field.
- Researching on how to specify orientation and navigation aids for WLinks; it is evident that when traversing a WLink, the user can deviate from his path eventually loosing his way to the intended object (or just changing his mind). Providing physical navigation services (the equivalent of their counterpart in the Web) we can help him in his detour. We have already defined a framework for pervasive services [33], which we are currently improving.
- Devising model-based development tools to simplify the creation of a running application, using L-Struts.
- We are working on an open source Web browser to adapt its basic set of features to the PH domain. For example, while the “back” operation has well understood semantics in digital documents, it is not clear what it means in the physical world. Similarly other well-known features of our browsers should be analyzed to make them PH-aware.
- Exploring other possible implementation settings; particularly we are studying how to adapt an architecture for context-aware software [13] with physical hypermedia services [5].

Model-based design of Physical Hypermedia and other similar kind of ubiquitous Web software is not a new area as described in Section 2; however many problems in this area still need further research. In this paper we have presented some of these problems together with our solutions; we think that the strategy of seamlessly improving existing standard development tools is a key to the progress of this domain, as it allows that existing developers can migrate their application easily to support the mobile user.

References

1. G. D. Abowd: Software Engineering Issues for Ubiquitous Computing. Proc. 21st Int'l Conf. Software Engineering, ACM Press, 1999, pp. 75-84.
2. Adaptive Hypermedia Home Page: http://wwwis.win.tue.nl/ah/.
3. L. Baresi, F. Garzotto, P. Paolini: From Web Sites to Web Applications: New Issues for Conceptual Modeling. ER Workshops 2000, 89-100.
4. S. Ceri, P. Fraternali, A. Bongio: Web Modeling Language (WebML): a Modeling Language for Designing Web Sites. Proceedings of the 9th International World Wide Web Conference, Elsevier, Amsterdam, Netherlands, (May, 2000).
5. C. Challiol, S. Gordillo, G. Rossi, R. Laurini: Designing Pervasive Services for Physical Hypermedia Applications. Proceedings of the IEEE International Conference on Pervasive Services. Lyon, June 2006.
6. K. Cheverst, N. Davies, K. Mitchell, A. Friday, C. Efstratiou: Developing a Context-Aware Electronic Tourist Guide: Some Issues and Experiences. In Proc. of CHI 2000.
7. J. Crupi, D. Malks, D. Alur: J2EE PATTERNS Best Practices and Design Strategies. Publisher: Prentice Hall / Sun Microsystems Press, June 2001.
8. Y. Deshpande, A. Ginige, S. Murugesan, S. Hansen: Consolidating Web engineering as a discipline. The Journal of Software Engineering Australia, pp. 31 - 34, 2002.
9. Y. Deshpande, S. Hansen: Web Engineering: Creating a Discipline among Disciplines. IEEE MultiMedia 8(2): 82-87 (2001).
10. Y. Deshpande, S. Murugesan, A. Ginige, S. Hansen, D. Schwbe, M. Gaedke, B. White: Web Engineering. Journal of Web Engineering, vol. 1, pp. 3 - 17, 2002.
11. M. Douglas, D. Schwabe, G. Rossi: A software arquitecture for structuring complex Web Applications. Journal of Web Engineering 1 (1): 37-60 (2002).
12. 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.
13. A. Fortier, G., Rossi, S. Gordillo: Decoupling Design Concerns in Location-Aware Services. In Proceedings of the IFIP Conference on Mobile Information Systems (MOBIS),2005, Springer, 2005.
14. E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns. Elements of reusable object-oriented software, Addison Wesley 1995.
15. S. Gordillo, G. Rossi, D. Schwabe: Separation of Structural Concerns in Physical Hypermedia Models. CAiSE 2005: 446-459.
16. S. Gordillo, G. Rossi, F. Lyardet: Modeling Physical Hypermedia Applications. SAINT Workshops 2005: 410-413.

17. 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.
18. K. Grønbæk, R. H. Trigg: Design Issues for a Dexter-based hypermedia. Proc. ACM ECHT Conference (1992) pp. 191-200.
19. 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.
20. 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/.
21. T. Hofer, M. Pichler, G. Leonhartsberger, W. Schwinger, J. Altmann: Context-Awareness on Mobile Devices - The Hydrogen Approach. Proceedings of the International Hawaiian Conference on System Science (HICSS-36), Minitrack on Moblie Distributed Information Systems, Waikoloa, Big Island, Hawaii, January 2003.
22. M. Jones, G. Marsden: Mobile interaction design. John Wiley & Sons. February 2006.
23. G. Kappel, B. Prll, W. Retschitzegger, W. Schwinger, T. Hofer: Modeling Ubiquitous Web Applications -- A Comparison of Approaches. Proc. Of the Int. Conf. on Information Integration and Web-based Applications and Services (iiWAS), Austria, Sept. 2001.
24. G. Kappel, W. Retschitzegger, W. Schwinger: Modeling Customizable Web Applications - A Requirements' Perspective. International Conference on Digital Libraries: Research and Practice (ICDL), Koyoto, Japan, November 2000.
25. T. Kindberg, J. Barton, J. Morgan, G. Becker, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, H. Morris, J. Schettino, B. Serra: People Places Things: Web Presence for the Real World. in Proceedings of WMCSA, pp. 19-35.
26. A. Knight, N. Dai: Objects and the Web**.** IEEE Software, January/February 2002, 51-59.
27. 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.
28. U. Leonhardt: Supporting Location-Awareness in Open Distributed Systems. PhD thesis, Department of Computing, Imperial College of Science, Technology and Medicine, University of London, May 1998.
29. S. S. Moura, D. Schwabe: Interface Development for Hypermedia Applications in the Semantic Web. WebMedia/LA-WEB 2004: 106-113.
30. OMG Model-Driven-Architecture. In http://www.omg.org/mda/.
31. S. Pradhan, C. Brignone, J. Cui, A. McReynolds, M. Smith: Websigns: Hyperlinking Physical Locations to the Web. Computer, Vol: 34:8, 42--48.
32. 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.
33. G. Rossi, S. Gordillo, C. Challiol, A. Fortier: Context-Aware Services for Physical Hypermedia Applications. In Proceedings of the 2006 Workshop on Context-Aware Mobile Systems (CAMS'06), Springer Verlag, LNCS, forthcoming.
34. D. Salber, A. Dey, G. Abowd: The Context Toolkit: Aiding the Development of Context-Enabled Applications. Proceedings of ACM CHI 1999, pp 434-441.
35. J. H. Schiller, A. Voisard: Location-Based Services. Morgan Kaufmann 2004.
36. D. Schwabe, R. Guimarães, G. Rossi: Cohesive Design of Personalized Web Applications. IEEE Internet Computing 6(2): 34-43 (2002).
37. 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.

38. J. P. Sousa, D. Garlan: Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments. (3rd IEEE/IFIP Conference on Software Architecture) WICSA 2002: 29-43.
39. F. Steimann: On the Representation of Roles in Object-Oriented and Conceptual modeling. Data and Knowledge Engineering 35 (2000) 83-106.
40. Struts Evaluation in https://equinox.dev.java.net/framework-comparison/WebFrameworks.pdf.
41. The Struts Home Page: http://struts.apache.org/.
42. The UML Home Page: www.omg.org/uml/.
43. UWA Project. www.uwaproject.org.
44. S. Weiss: Handheld usability. John Wiley & Sons, July 2002.