Separation of Concerns in Mobile Hypermedia: Architectural and Modeling Issues

Cecilia Challiol
LIFIA. Facultad de Informática. UNLP. La Plata, Argentina.
CONICET, Argentina.
ceciliac@lifia.info.unlp.edu.ar

Gustavo Rossi
LIFIA. Facultad de Informática. UNLP. La Plata, Argentina.
CONICET, Argentina.
gustavo@lifia.info.unlp.edu.ar

Silvia E. Gordillo
LIFIA. Facultad de Informática. UNLP. La Plata, Argentina.
CICPBA, Argentina.
gordillo@lifia.info.unlp.edu.ar

Andres Fortier
LIFIA. Facultad de Informática. UNLP. La Plata, Argentina.
DSIC. Universidad Politécnica de Valencia, Valencia, España.
CONICET, Argentina.
andres@lifia.info.unlp.edu.ar

ABSTRACT

Building Mobile Hypermedia and Web Applications is hard because of the myriad of concerns we need to face, such as those related to the specific application domain and those typical of mobile software. During the last years, we have been researching on modelling techniques for mobile hypermedia, and building infrastructure support for this and other kind of mobile and context aware software. In this chapter, we review the modelling features, design mechanisms and architectural support that we have developed to simplify the development process, and to obtain more flexible models and applications.

1. INTRODUCTION

Mobile Hypermedia and Mobile Web applications are difficult to build and maintain as they pose strong requirements to developers. As usual in complex Web software, this kind of applications must cope with sophisticated content domains, provide good navigation facilities and offer a usable interface (moreover if you consider the small size of mobile devices).

Additionally, mobility adds other issues to be considered such as content (navigation or interface) and adaptation to the actual user location in order to make the application aware of the user's context (his location). As a consequence, we are obliged to stress our usual modeling and support tools, as we face new modeling and development problems which are certainly challenging.

Unfortunately, run-time platforms are far from being standard (as in "old" Web software), and there are not yet widely used frameworks supporting the myriad of technological elements needed for this kind of software. Sensing devices (both for outdoor or indoor positioning) are also evolving fast, and their interfaces with higher level software components are also a matter of research. Consequently, mobile Hypermedia and Web software are built on top of ad-hoc architectures which encompass a set of abstractions and their corresponding communication and cooperation mechanisms.

Even simple applications, like mobile hypermedia tour guides, present interesting modeling, design and implementation problems. This way they cover different concerns both, functional (like integrating location-based services with navigation), and non-functional (like improving usability while assuring privacy and security).

In the last four years, we have been researching different aspects of mobile hypermedia development. We have proposed a modeling approach for physical hypermedia (Gordillo, Rossi, & Schwabe, 2005), and developed a modular architecture for building context-aware software (Fortier, Cañibano, Grigera, Rossi, & Gordillo, 2006). We have also analyzed different design aspects of mobile hypermedia, particularly those related to context-aware assistance to the traveler (Challiol, Rossi, Gordillo, & De Cristófolo, 2006; Challiol, Fortier, Gordillo, & Rossi, 2007; Rossi, Gordillo, Challiol, & Fortier, 2006) and the impact of different browsing semantics in physical hypermedia applications (Challiol, Muñoz, Rossi, Gordillo, Fortier, & Laurini, 2007).

In this chapter, we intend to synthesize our experience focusing on the different facets of the mobile and context-aware hypermedia engineering enterprise; particularly, we aim at analyze the impact of a wise separation of concerns in different quality properties of mobile hypermedia software.

The structure of the chapter is the following: In Section 2 we motivate the reader to study an example in which we illustrate some of the problems discussed above. In this ay we show that, even using state-of-the-art modeling approaches, there are problems which are inherent to mobile hypermedia, and which require to improve separation of concerns both in relationship with specific application concerns, and also with more "paradigmatic" concerns, such as navigation, user interface, "physicality" of application objects, etc. While some concerns remain orthogonal, others tend to crosscut and require different kinds of advanced techniques (aspects, roles, intelligent use of patterns, etc.). Next, in Section 3, we review existing research work both on modeling and architecting mobile hypermedia applications.

In Section 4 we characterize the different kinds of design and architectural concerns in these applications, and show how these concerns impact on the design complexity and specifically on the ease or difficulty to support the evolution of the application. Next, in Section 5 we analyze each of the problems, and present our solutions for them. We present our choices in such a way that they can be reused in different implementation settings. Particularly, we show how our finer grained micro-

architectural constructs, focusing on different concerns such as sensing, context modeling, adaptation, etc. These can be combined to yield solid, modular and easy to evolve systems. In Section 6 we focus on modeling issues and in Section 7 we present our modeling approach. In Section 8, we finally focus on browser support for mobile hypermedia and show how to support concern-oriented browsing. In this context, we revise our research on browsing semantics to outline our vision for a mobile hypermedia browser.

Finally, we summarize the most relevant aspects of the chapter, and present some further research we are current working on.

2. MOTIVATION

In the rest of this chapter we will assume that a mobile hypermedia application provides the final user with contents, services and links which depend on his actual location. In this sense, a mobile hypermedia application is similar to a conventional hypermedia (or Web) application except that the user might explore the information of the place "physically" by visiting specific places in that environment using not only conventional navigation, but also "physical" navigation (i.e. moving around that place for example walking) . Some authors have also called this kind of applications, physical hypermedia (Grønbæk, Kristensen, Orbæk, & Eriksen, 2003).

As an example suppose that we are building an application to be used by visitors to a city (we will use our city, La Plata) to support them while they are moving up around it. As in every kind of hypermedia software, different types of visitors (e.g. a tourist or a historian) will be interested in different details of the city or its buildings. In the following scenario we review some of the most important problems we face while building this kind of application.

When the user faces the Cathedral, his mobile device shows him hypermedia information about the Cathedral, links that he can navigate digitally, some operations he can perform, and some related places where he can go after visiting the Cathedral. Figure 1 shows a simplified version of the contents as perceived by the tourist and the historian. Notice that each user will perceive slightly different contents and links. Moreover, they can select a "physical" view (centered in geographical more than informational aspects) to see planned paths (as shown in Figure 2) and suggested paths (as shown in Figure 3). Both options have links which can be traversed by walking to the target object. However, the user will encounter two different types of paths: those that we call *static* links, which are specified by the application designer, and will always be shown when displaying the web page and the application will also show additional links that are suggested by the system, which we call *dynamic* links. These links are calculated on the fly according to what the user is actually seeing on the hypermedia view. In our example, the planned path is the same for the two users, but dynamic links are calculated according to the (more personalized) links shown in Figure 1.

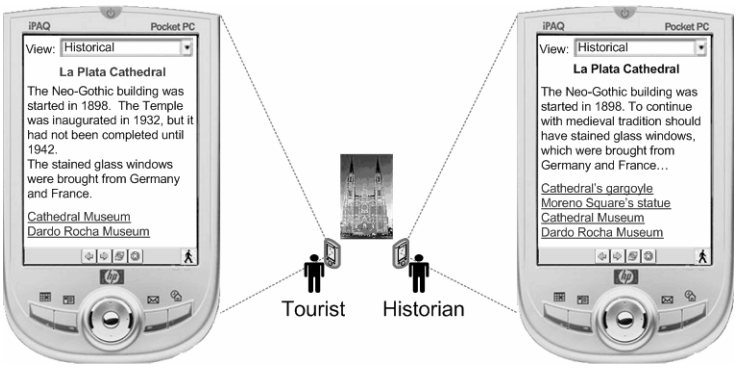

Figure 1: Both users are standing in front of the Cathedral.

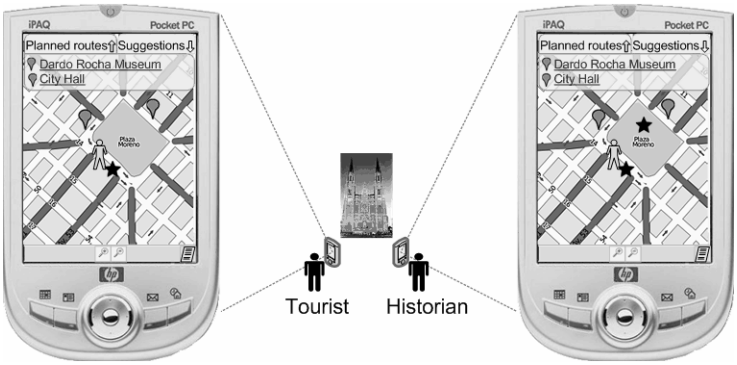

Figure 2: The planned path of the Cathedral for the two users.

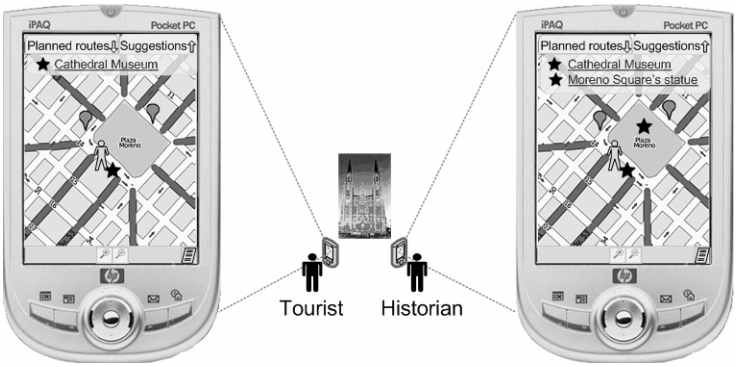

Figure 3: Both users are seeing their suggestion paths.

This simple scenario summarizes some of the underlying issues we need to solve to successfully build a mobile hypermedia application. Though we will review them thoroughly later, we briefly summarize the most important ones in the context of the example, focusing mainly in architectural and modeling issues. Even though technological aspects (e.g. how to sense the position of the user) or algorithms (e.g. how to

compute the path between two addresses in a city) should not be neglected; most of these issues have been widely discussed in the literature, so we omit them in this discussion.

The first set of problems refers to the architectural organization of this kind of software, specifically to which are the main components of a mobile hypermedia application and how we cope with variability issues, e.g. when technology evolves or varies, or when we need to accommodate different technologies for the same kind of functionality, such as sensing the position of the user (using global positions with GPS, code bars or similar technologies, etc). In this set of problems, we also include how to relate contextual information (e.g. the position of the user, his actual activity or interest) to the way it is captured and the corresponding context-aware reaction. For example, when the user faces the Cathedral, the application reacts by activating a hypermedia node; the content of this node might depend on different contextual data (e.g. his profile, the day of the week, etc.). The way in which we decide to organize this set of components has a strong impact in the application's modularity and therefore on how easy or difficult it is to support its evolution.

The second set of problems is related to modeling issues. In particular, which modeling approach is the best one for coping with the inherent complexity of the kind of applications represented by the example above? While well known hypermedia modeling approaches (for instance OOHDM (Schwabe & Rossi, 1998) or UWE (Koch & Kraus, 2002)) might help us to express the main abstractions in the example using a high level notation (e.g. in the style of UML), there are still some open issues such as, representing physical links or indicating in a clear way which are the different application concerns the user might be interested in (e.g. history, architecture, sports, shopping, etc), or how to relate these concerns to other hypermedia functionality such as links and paths, etc.

In the following sections we analyze these issues and other and describe some of the solutions we have developed in this field.

3. RELATED WORK

Separation of concerns has been a constant objective of modern architectures for context-aware software, particularly in the mobile hypermedia field. The seminal ideas of the Context Toolkit (Salber, Dey, & Abowd, 1999), in which the context sensing concern was clearly separated from the components that processed the context in (Salber, Dey, & Abowd, 1999) aggregators, interpreters and applications, were later applied with variations in many other frameworks and systems. Particularly, the HyCon system (Hansen, Bouvin, Christensen, Grønbæk, Pedersen, & Gagach, 2004), aimed at extending the hypermedia paradigm with the manipulation of real world objects, also realized these ideas. The authors use this systems to create applications in different areas, such as Mobile Urban Drama (Hansen, Kortbek, & Grønbæk, 2008), Mobile Learning in Context (Hansen & Bouvin, 2009) and Social web information in the city (Hansen & Grønbæk, 2008). In summary, the usual strategy to separate architectural concerns in mobile hypermedia consists in decoupling the sensing components from

the context model (specifically the user's location) and these two from the components which use the context to adapt the application's functionality. However, as discussed in the following sections, we have found some limitations in this approach, particularly when treating context as a monolithic entity.

In a similar way, most hypermedia modeling approaches have promoted some kind of separation of the "paradigmatic" (i.e. content, navigation and interface) concerns in hypermedia. Modeling approaches such as OOHDM (Schwabe & Rossi, 1998), UWE (Koch & Kraus, 2002) or OOWS (Fons, Pelechano, Albert, & Pastor, 2003) decouple content from navigation and presentation modeling. In this way, different facets of the same application can evolve without making an impact on others; for example by changing the presentation model, we can adapt the hypermedia application to different browsing devices (from desktop to mobile phones). In UWA (UWA Project), this strategy has been complemented with additional context and rule models, the former for representing the user's context and the latter for specifying the behavioral reactions to context changes. In Section 7, we show how we extended these ideas by adding an additional separation of application concerns to further improved modularity and therefore evolution and maintenance.

4. SUMMARY OF ARCHITECTURAL AND DESIGN PROBLEMS

As a result of building different prototypes of context-aware applications and in particular mobile hypermedia ones, we found several outstanding concerns that should be separated in order to build scalable applications.

The main challenge that a context-aware application has to face is how to adapt itself according to the current context. In this statement there are two key issues that must be analyzed. First of all, to build an application that adapts itself to the current context we must have an internal representation of what is considered the context (i.e. we need a context model). The second issue that we must tackle is how to perform the required adaptation, especially because the word adaptation is generally used in the context-aware community in a very wide sense (e.g. changing the application's presentation (Eisenstein, Vanderdoncki, & Puerta, 2000), the displayed content (Pascoe, 1998), performing proactive (Leonhardt, 1989) or reactive actions (Lamming & Flynn, 1994)). Finally, we must also cope with the myriad of sensing technologies used to gather context information, both at the hardware (GPS, ARTags (ARTags, 2009), IR (Schilit, Adams, & Want, 1994), etc.), and software levels (e.g. a weather web service) and even with legacy applications that must be "upgraded" to support context-dependent behavior. In the next sub-sections we will describe the most relevant concerns, we will outline the difficulties and main requirements associated with them. For the sake of conciseness, we omit the discussion on sensing issues. Later, in Section 5 we will present our architecture and explain how these problems are tackled.

4.1 Context models

Context-aware applications can be developed in very different domains, like health caring (Bricon-Souf & Newman, 2007), tour guides (Cheverst, Mitchell, & Davies, 2002) and learning environments (Jones & Jo, 2004) just to name a few. However, designing a context model is not only difficult because of all the domains it can be applied to, but because of the set of requirements that it must fulfill. In particular, we have found that:

a) The context model should be changeable in run-time. If we consider that the user of a mobile context-aware application is exposed to constant social changes, it follows that what is contextually relevant at a given time will also be constantly changing. Thus, the context model should be designed to change its structure according to the current situation.
b) Context models should allow for different technologies to tackle different context domains. While a simple XML document may be enough to encode a user's location, enhanced RDBMs (Bolchini, Curino, Quintarelli, Rossato, Schreiber, & Tanca, in press; Stefanidis, Pitoura, & Vassiliadis, 2005) are better suited for context-dependent data retrieval and ontologies are a good choice for reasoning about concepts relationships. Thus, while a unified context model is desirable, it should be open ended so that different context elements (e.g. location, activity, user's mood, etc.) can be modeled with different technologies.
c) Already built and tested context models should be reusable in different applications. For example, a well designed location model can be used in applications like friend finders, tourist guides, route planners, location-based services, etc. Thus, it is extremely important that context models can be reused.

From these requirements we can see that context modeling is not a trivial issue and that is why the subject has been studied so hard. In Section 5.1 we will present our context model, which aims at solving the presented requirements.

4.2 Decoupling Adaptation from Context Models

Even though the context-aware behavior could be coupled with the context model, this would seriously threaten reuse and scalability. As an example, consider a framework or library used to manage hybrid location models (Leonhardt, 1989), which can be used for many context-aware applications, such as friend finders, tourist guides or location-based services. If we bind the context model to the specific context-dependent behavior, not only we cannot share the context model between applications, but it would also be a nightmare to share improvements to the context model, since we should first remove all the context dependent behavior in order to port the context improvements to another system. For these reasons, we consider a key issue to decouple the context model from the way it is used to provide context-dependent behavior. To do so, we use the notion of an *adaptation domain* to characterize the context-

dependent behavior of an application. Thus, a friend finder would define a specific adaptation domain, while a tourist guide would have a completely different one.

To effectively cope with adaptation domains in context-aware systems we define two new requirements:

a) Different domain-specific adaptation should coexist. As a general rule, adaptations should be modeled and developed without taking care of other adaptations present in the system at the same time. Of course, if resource sharing is required (e.g. accessing a database), some kind of synchronization will be required, but this should be the exception, not the rule.

b) Domain-specific adaptation should be decoupled from the base model. Whether we are building an application from scratch, or improving an existing one to support context-aware features, the application model should be independent of the proposed adaptation, since the forces that drive the changes in each case are orthogonal.

5. SUPPORT ARCHITECTURE

In this section we review the issues presented in Section 4 and explain our solutions. We first explain our ideas in an abstract way, and then show our concrete architecture and implementation. Even though our ideas can be realized in any OO language, we decided to implement them on VisualWorks Smalltalk, due to the language simplicity and its natural reflective capabilities.

5.1 Context Models

Since context modeling is a sensible issue, we put special emphasis on this part of the architecture. Our approach takes as a starting point that the context model will be based on the pure definition of the object oriented paradigm (Kay, 1993), where an application can be seen, in a reductionist way, as a set of objects collaborating with each other by sending messages. As a result, the behavior of an application is scattered in a set of objects, which are responsible for implementing certain responsibilities (Wirfs-Brock & McKean, 2002). This basic statement, which may seem trivial at first, is actually one of the cornerstones of our approach, since we do not consider context as a piece of data that is floating around our program, but as information that is attached to an object. Thus, one of our core guidelines for context modeling is that *context information is always bound to an object*. This means that in order to manage context information, we must first define *whose* context it is. Notice that this first guideline relates to the phenomenological view of context presented in (Dourish, 2004), where the activity is considered as the originator of context, and it is stated that context and activity are not separable. If we translate this statement to the OO paradigm (where objects are the core modeling abstraction), we would see that the object and its context are not separable.

This first statement about context modeling has a direct impact on how context information is managed, since the idea of context as a monolithic structure is not possible. Instead, what is generally referred to as "the context" is actually the aggregation of each object's context.

At the architectural level, we based our context model in two main abstractions: *aware objects* and *context features*. An aware object is the context-aware counterpart of an application object, which resides in the context layer. To actually model context information, we decided to split an object's context in features, where each feature represents a specific context domain. As an example, suppose that to provide location-based services (LBSs) we need to track a user's location and his current activity. In order to do so, we create an aware object that represents the user and associate two context features to it, one for tracking the user's location and other for his activity. In Figure 4, we show an instance diagram depicting this scenario.

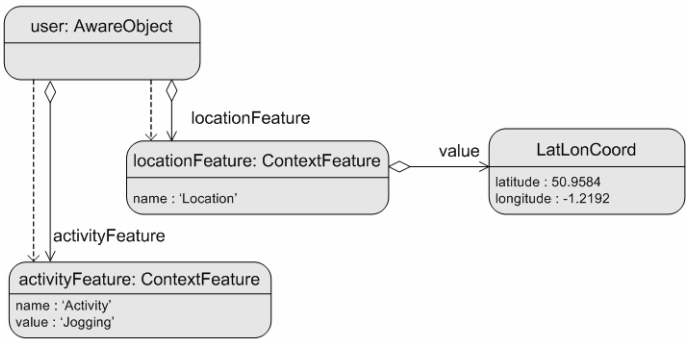

Figure 4: A simple aware-object configuration for LBSs.

As shown in the figure, the aware object not only has a knowledge relationship with each context feature, but also acts as an Observer (Gamma, Helm, Johnson, & Vlissides, 1995) of each of them. By setting this relationship, the aware object gets a notification each time any of its context feature changes its value (we will later show how this enables us to program context-aware behavior).

As discussed in Section 4.1, the context structure should be changeable at run time; this implies that we must provide a mechanism to dynamically attach (or detach) context information to an object. In our architecture, this means that we should be able to dynamically add or remove a context feature to an aware object. To do so the `addFeature(feature: ContextFeature)` and `removeFeature(feature: ContextFrature)` messages, implemented in the `AwareObject` class are used. These messages not only perform the expected behavior (e.g. adding a new context feature to the aware object's feature collection), but also add new behavior to the aware object. In our implementation, we make a heavy use of Smalltalk's reflective capabilities, to dynamically add behavior according to the aware object's context features. In particular, by adding a new feature (e.g. a *location* feature) an aware object understands two new messages:

a) `locationFeature()`, which returns the context feature named "location".

b) `currentLocation()`, which returns the value of the context feature named "location".

Notice that, besides providing syntax sugar (for example the user.currentLocation() message could be implemented as user.getFeatureNamed('location').value()), these two messages have a theoretical reason to exist, since we consider that a context feature extends the object it belongs to. As we will show in the next section, this issue becomes more relevant when we are extending an existing application to support context-dependent behavior.

A key aspect of our design is the fact that each context feature can be engineered separately. This allows the developer to focus on one context domain at a time, and to compose them by means of an aware object. This ability aims at solving the second requirement presented in Section 4.1, and it is due to the fact that no constraints are place regarding what can be the value of a context feature. Also, the very nature of our solution allows the developer to extend the context feature hierarchy to add more complex features if required. As a matter of fact, during the development of different case studies, we have identified a set of specialized context features, which are provided as part of our framework:

a) Model Based Feature: In many cases a feature's value is meaningless without a supporting model. Suppose that we are tracking the user's location in a symbolic map (Leonhardt, 1989), and for that purpose we have a context feature that holds the symbol that identifies the room where the user is standing. Without the map itself, the symbol is not of much use, since we cannot connect it to other locations. The model based feature extends a context feature by adding a reference to a model.
b) Tracked Feature: Some applications require a history of the situations in which the user has been involved, being mobile hypermedia one of those cases. For this purpose, a tracked feature extends the basic context feature with the capacity to keep a log of the context feature history.
c) Derived Feature: Context features represent small-grained pieces of context information. In some cases, we can combine that information by means of a transformation to produce derived context information. As a simple example, suppose we need to present weather information to the user, so that he knows if the conditions are favorable for going cycling. If we have context features like temperature, pressure, wind, etc., we could use a decision tree (Quinlan, 1986) whose inputs are the current weather conditions and the output is the suggestion made by the program. An important characteristic of a derived feature is that it automatically updates its current situation (and triggers a change event), when any of its input features change.

Finally it is important to notice that tracking the changes of a context feature, or requiring an underlying model are not mutually exclusive. For these reason the extensions presented earlier are implemented as Wrappers (Gamma, Helm, Johnson, & Vlissides, 1995) and, if needed, the developer can add its own wrappers.

In Figure 5 we present a class diagram of the `ContextFeature` hierarchy, where the root of the hierarchy is an abstract class that can be extended to add new custom

context features. Notice that a derived feature can be configured with a policy that states when it should trigger a change in its current situation (namely, every time any of its input features changes, only when its new situation is not identical to the previous one, or when its new situation is not equal to the previous one).

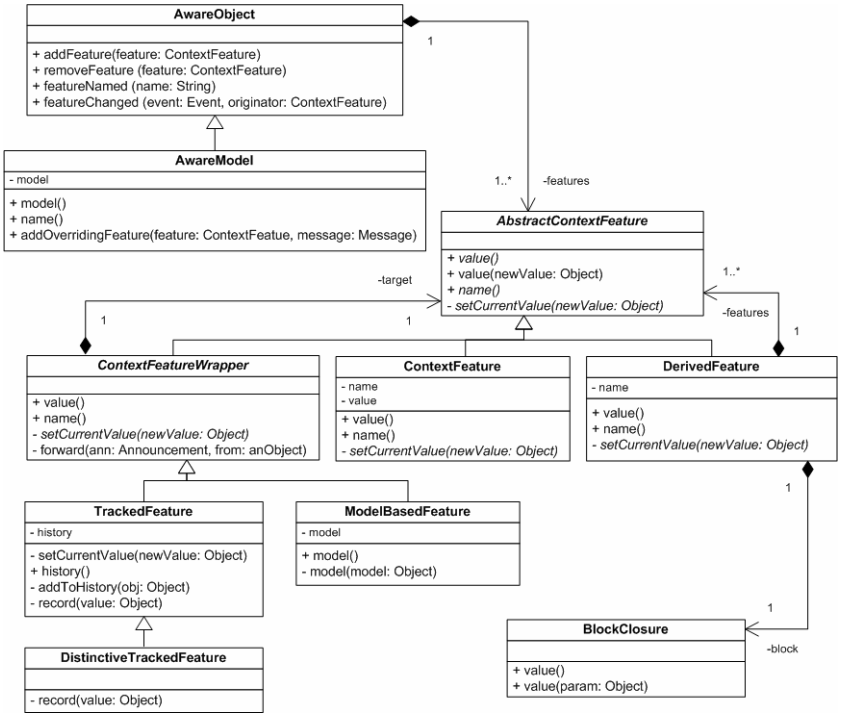

Figure 5: A simplified class diagram of our context model.

By using aware objects and context features, what we are indirectly doing is isolating the different context-domains. This allows each context domain to be designed, implemented and tested independently from the others. Of course, we still have to run integration tests, but this is definitely a major step towards reusability compared to considering context as a monolithic entity.

Finally, we must address the last two requirements related to context models, which are context sharing and privacy. However, since we make a clear distinction between context model and how that model is used by the applications these requirements will be discussed in the next section.

5.2 Decoupling Adaptation from Context Models

As we shown in the previous section, one of our goals is to separate the context model from the application domain to get a clear separation of concerns, resulting in better and more flexible designs. We consider that the same idea should be applied between context models and the context-dependent behavior. To clarify this separation, con-

sider the task of providing LBSs to a mobile user: in this scenario we can identify the minimum required context (the user's location) and the expected behavior (according to the user's location, let him access a set of services). However, the service's domain (e.g. health-caring, traffic monitoring or finances) is, in principle, not relevant. The important thing is to have an infrastructure to provide LBSs, not the services themselves.

We have aimed at separating the application domain from the adaptation domain, so that developers can build specific infrastructures (e.g. for LBSs) that can be reused across different application domains (e.g. health-caring, finances, etc.). To do so we decided to treat adaptation domains as first class objects, which are called *adaptation environments.* Each adaptation environment references a set of aware objects in order to "listen" to their context changes. When a context change is triggered, a notification is delivered to the environment so that it can perform a specific action, like showing a new service in the case of LBSs, or turning off a light in the case of a smart home.

To illustrate the use of the adaptation environments we will explain a subset of the functionality of a mobile hypermedia application. Suppose that the user is standing in the Cathedral and has chosen to walk to the City Hall. This path is shown to the user, and recorded as part of the application's internal state. However, the user can make a wrong turn and head toward the University instead of the City Hall. However, since the environment is user-dependent (which is modeled as an aware object), each time his location changes, the environment gets a notification, which can be checked against the recorded route, to see if the user is just to the right direction. In case he is not, the system can warn him by means of a visual cue, or a sound alarm.

In our approach, the behavior related to the path calculation and route checking would be encapsulated in a specific mobile hypermedia adaptation environment, whereas the user's context model remains adaptation-agnostic.

By encapsulating the context-dependent behavior, we can solve both of the requirements presented in Section 4.2, since many adaptation environments can coexist independently.

6. MODELING ISSUES

Mobile Hypermedia applications are complex as they involve many different problems; their evolution (as explained in Section 5) can be affected by technology changes and also by the evolution of the applications' requirements themselves. As a consequence, we found it mandatory to face them using a high level approach in which modeling and design decisions are clearly recorded. In this section, we briefly summarize which are the most important problems which may influence the way in which we model a mobile hypermedia application. For the sake of conciseness we emphasize those problems which have not been dealt elsewhere.

Analogously to the broader field of hypermedia applications, we can notice that an application may involve information and services belonging to different concerns, and at the same time, different user profiles (e.g. the historian or the tourist) might be interested in different information.

As an example, when a tourist is standing in front of the Cathedral he may want to receive tourist information about the place (opening times, for example). But additionally, the Cathedral may have a shop which can be explored to buy some presents for the family; this last information (and the related services) belongs to a completely different concern than the opening times or the architectural details of the building. Additionally, as mobile devices have small screens, it is clear that we cannot pollute a hypermedia page with dozens of information items (e.g. like in Amazon.com). As also shown in the motivation, the user may want to explore the "physical" view (See Figure 2). A naïve view of these problems might conclude that we are facing "just" an interface issue (i.e. how to accommodate information in small screens to make the application usable). Whereas, from our point of view the problem is somewhat more complex as it does not involve only the look and feel of the application, but also the way in which we bundle information and services together (which pertains more to the navigational design stage) and, considering evolution, the way in which they can change.

As a summary, we found it necessary to use or adapt a modeling approach which allows us to:

a) Specify different user profiles to better organized navigation.
b) Incorporate the "physicality" of this kind of software in a seamless way and
c) Be able to accommodate multiple "thematic" concerns, in such a way that we can modularize the specification of a running system.

Additionally, we aimed to generate a modeling approach which could be automated in the model-driven software development way i.e. that allows producing running applications directly from the design models.

7. OUR MODELING APPROACH

To solve the above mentioned problems, we developed a light extension of the OOHDM (Schwabe & Rossi, 1998) design framework. To be concise, in the following sub-sections, we focus particularly on the conceptual and navigational models. A thorough description of requirement issues can be found in (Nanard, Rossi, Nanard, Gordillo, & Perez, 2008).

7.1 Conceptual Model

Assuming that during the requirements modeling process we have identified the most important application concerns, the conceptual model contains one package for each concern. We focus here on "navigational" concerns, i.e. those which affect hypermedia navigation. Other ones, such as persistence, security, etc. (though important) are ignored in this chapter. In each package, the designer specifies those classes and relationships pertaining to the specific concern. In our example, we identified two clear concerns: the historical and the physical one. Additionally, we could define other concerns such as tourist, architectural, e-commerce, etc., a class may appear in more

than one concern. In each concern, the class represents the specific "view" of its objects when accessed in that concern.

The physical concern contains those classes which can be explored physically by the user. These classes define the objects' location and other geographical aspects. In Figure 6, we show a conceptual model with different concerns emphasizing the two concerns of the motivation example.

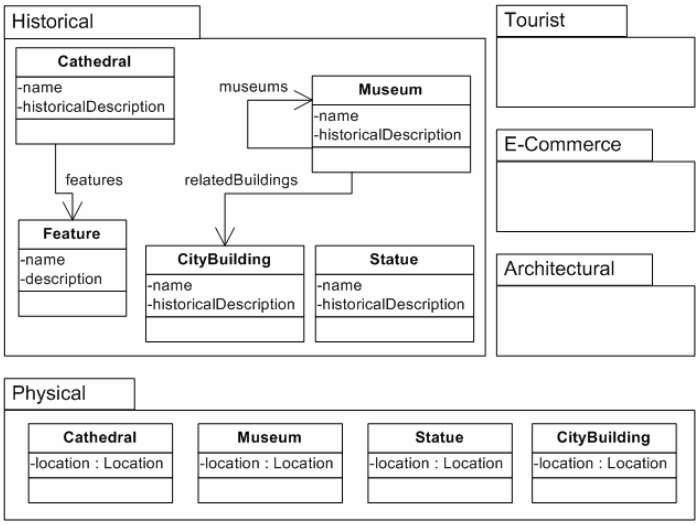

Figure 6: Conceptual model for the motivation example.

At the end of the conceptual modeling activity we weave the partial concern models onto a unified conceptual model. The designer needs to select the "core" concern of the application to be derived. After that, classes which appear in more than one concern (besides the core) are modeled as roles of each core class following the approach in (Rossi, Nanard, Nanard, & Koch, 2007). For the sake of conciseness we do not include a discussion on the possible use of aspects as a weaving approach for the multiple concerns in the conceptual model. Figure 7 shows a unique conceptual model in which the core of the application is the historical concern. The physical concern appears now as a role of the corresponding classes.

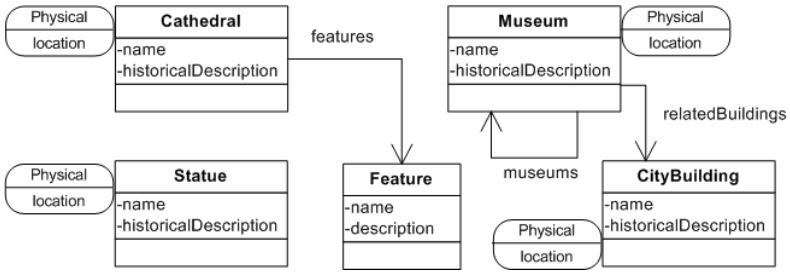

Figure 7: A unique conceptual model.

7.2 Navigational Model

From the conceptual model we can derive many navigational models (e.g. for different user profiles). To do so, classes are mapped onto nodes and relationships onto links. As the unique conceptual model is the base to create the navigational model, the core concern in navigation is the same one that the designer selected to build the unique conceptual model. The same applies to the others concerns in the unique conceptual model, which are expressed as roles of the core navigation nodes. The navigational model can have new links not mapped from relationships, and, of course, some relationships which are not meaningful for navigation might not be mapped to links.

Additionally, in the navigational model, the designer needs to specify the physical links. These links connect two physical objects (in fact two physical roles) expressing that there is a path between the corresponding nodes, and we intend the user to traverse them by walking. When the user selects one of these links, the system should provide him with instructions to move to the target of the physical links.

A node's physical role will contain static physical links shown in the navigational model and implicit physical links. These latter links are computed dynamically according to the current state of navigation. If a user is navigating in a digital (not physical) concern C_k, the physical concern can be enriched to show implicit physical links from C_k. The implicit physical links for a node M are represented by a function $f_M(Core, C, N_c)$, where *Core* is the core application model, C is the concern that the user is currently navigating and N_c is the current node in the navigation of the C concern. As an example, let us suppose that we want to calculate the implicit physical links to (the physical role of) *The Cathedral*. The function $f_{TheCathedral}(Core, C, N_c)$ is calculated with these values: *Core* is the historical concern and the user is currently navigating on this concern and facing the *Dardo Rocha Museum*. From this node, there are three digital links*: Natural Science Museum*, *La Plata* and *Dardo Rocha.* However, the *Natural Science Museum* is the only one with a physical representation. As the *Natural Science Museum* is not included in the static physical link of *The Cathedral*, then it is included as implicit physical links. In other words, the result of the $f_{TheCathedral}(Core, C, N_c)$ is the *Natural Science Museum* which is included as the target of the implicit physical link.

Figure 8 shows that the navigational model may add or remove information. As an example, the *CathedralNode* adds two links: one to the *MuseumNode* and another to the *StatueNode.* The physical role of the *CathedralNode* adds two static physical links (*MuseumNode* and *CityBuildingNode*). The *MuseumNode* removes the relationships *relatedBuilding* which is mapped to a link in the physical role of the *MuseumNode*. In this case, the designer decides to move the relationship to the physical concern because he considers it more useful when appearing as a static physical link.

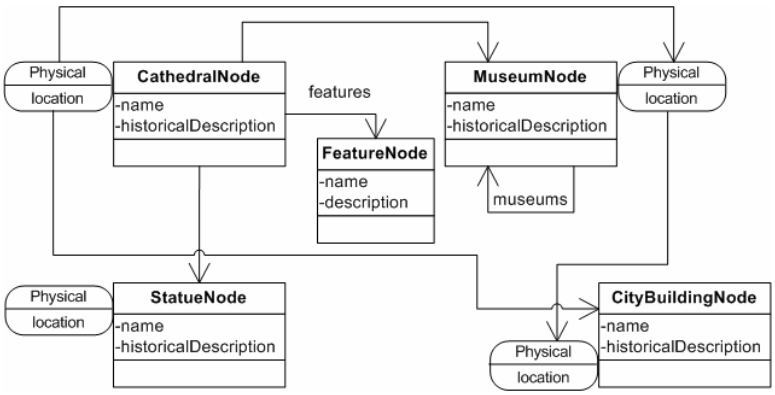

Figure 8: Navigational model.

An example of two nodes related to the motivation example is shown in Figure 9. The *CathedralNode* (La Plata Cathedral) is the source of four digital links one of them to the *MuseumNode* (Dardo Rocha Museum). The physical role of the *CathedralNode* has a static physical links to the physical role of the same *MuseumNode*. Notice that, the semantic of these links are different: some of them are standard digital links and the others are static physical links.

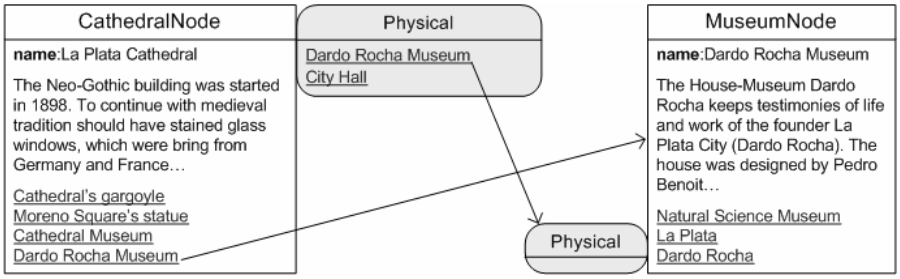

Figure 9: An example of two nodes and their links.

7.3 From Design Models to Mobile Hypermedia Applications

As explained in Section 4.2, we aimed to decouple the application model from the type of proposed adaptation. In this sense, it is important to indicate that both the conceptual and navigational models are part of the application model; these UML classes are mapped into classes (in our current implementation VisualWorks classes), where as links are mapped into knowledge relationships between corresponding objects. In terms of our MVC implementation architecture, the browser is also considered part of the "application model", we describe this in detail in Section 8.

To add adaptation facilities, as discussed in Section 5.2, we have built an adaptation environment for the mobile hypermedia adaptation domain. This environment

provides the necessary functionality to provide dynamic adaptation, e.g. to compute paths between two real world objects.

Mobile hypermedia applications allow the user to explore real world objects, to locate the user. We represent him as an aware object with a context feature for his actual position.

Each time the user changes his location, the environment is notified and provides the corresponding behavioral response, e.g. allow him to view information on the actual object in front of him. To achieve this functionality the environment notifies the browser, which is updated with the corresponding hypermedia node. Figure 10 shows the adaptation elements which are generated to provide hypermedia adaptation and their relationship with hypermedia nodes, populated from the navigational model.

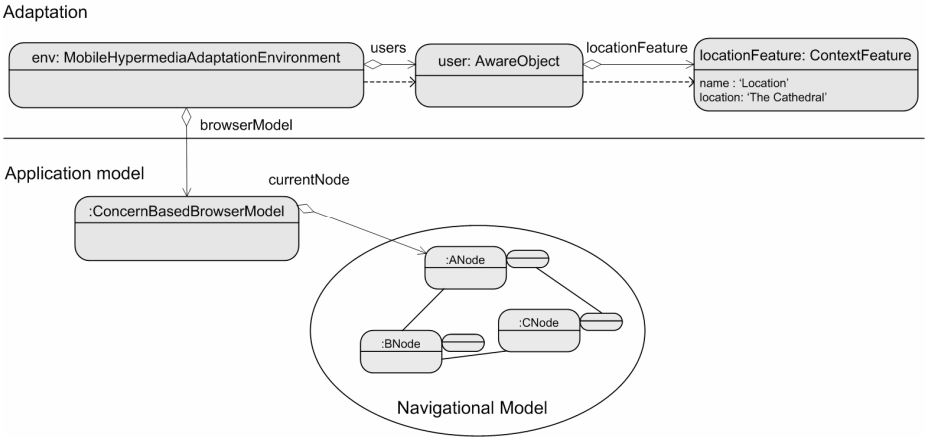

Figure 10: Instance diagram of the Mobile Hypermedia Application.

8. BROWSING THROUGH APPLICATION CONCERNS

In this section, we explain how to support the required Browser functionally for mobile hypermedia applications. In our previous works (Challiol, Muñoz, Rossi, Gordillo, Fortier, & Laurini, 2007), we have shown how to decouple a web browser from its model, using it as a view in the MVC (Krasner & Pope, 1988) sense. Thus, from an architectural point of view the web browser becomes the "renderer" of our hypermedia model.

8.1 Browser Basics

To effectively decouple the browser from the model, we created the *IBrowserModel* interface, which defines the minimum protocol that any hypermedia model must implement. A possible implementation of the *IBrowserModel* interface is the *BrowserModel* class. The navigational model decouples the navigation semantics by means of

the *NavigationStrategy* class. Thus, we can create different subclasses of *Navigation-Strategy* to implement different semantics for the *back* and *next* actions, according to the navigation behavior that we need to use. This design is shown in Figure 11.

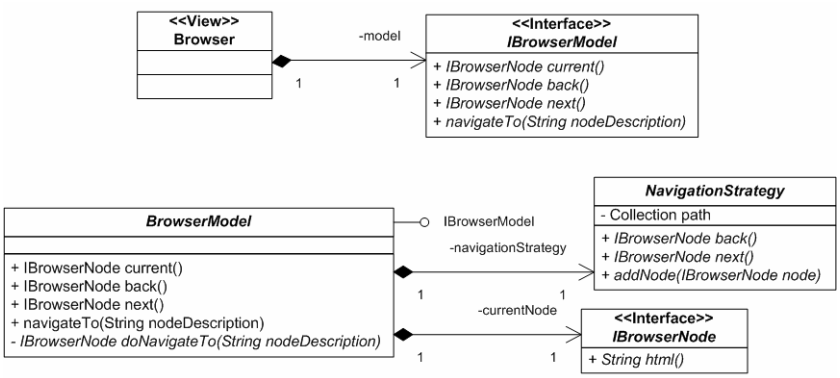

Figure 11: Definition of the BrowserModel.

As we have shown in the motivation example, the user can navigate digitally or physically. For this reason, we need to manage two browser models, one for the historical information (*HistoricalBrowserModel*) and another one for the physical information (*PhysicalBrowserModel*) as shown in Figure 12. Notice that, each browser model has its corresponding navigation strategy, since the digital and physical semantics of the back and forward actions are different (for more detail see (Challiol, Muñoz, Rossi, Gordillo, Fortier, & Laurini, 2007)).

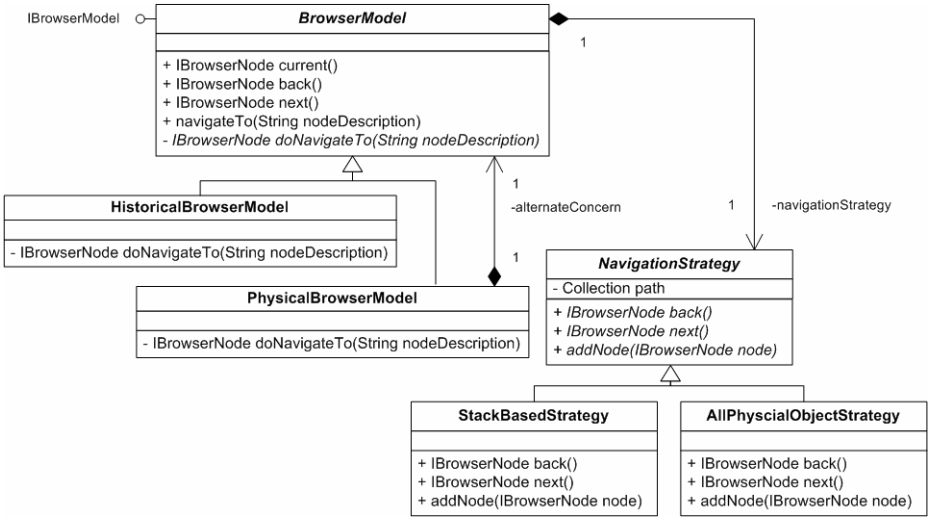

Figure 12: Historical and physical browser models.

The *PhysicalBrowserModel* defines the variable named *altenateConcern* which references the historical model, used to calculate the implicit physical links (*Suggested*

Links for the user). To generate these links, the physical role of the node needs to define what function is used to calculate them. This function is used by the *Physical-BrowserModel* and takes as parameters the *altenateConcern* and its current node. One example of implicit (physical) link functions has been detailed in Section 7.2.

8.2 Supporting concern-oriented browsing

In the sub-previous section we outlined how to create different browsing models, and how to use different navigation strategies for changing the browsing semantics. In this sub-section, we go one step further and show how different concerns can be coordinated to provide concern-sensitive browsing to a mobile hypermedia user.

To extend our model, we created the *ConcernBasedBrowserModel,* which implements the *IBrowserModel* interface (see Figure 13). Additionally, the *ConcernBased-BrowserModel* is associated with a collection of browser models, one of them being the currently active (*currentModel* relationship). Also, in the same way that a navigation strategy can be defined for each navigation model, we can specify a "macro" navigation strategy that applies to the whole composite concern.

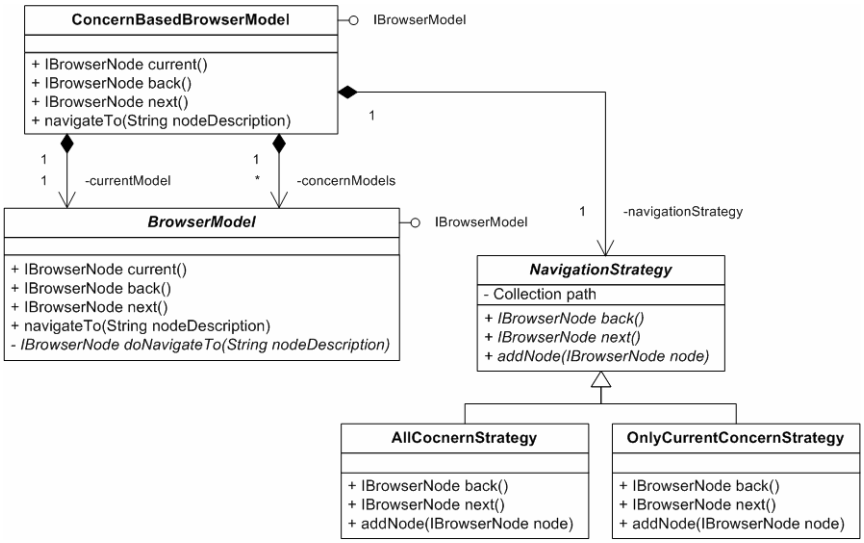

Figure 13: The concern browser model.

To clarify our architecture in Figure 14 we present an instance diagram of a scenario where the user is browsing the historical view of the cathedral.

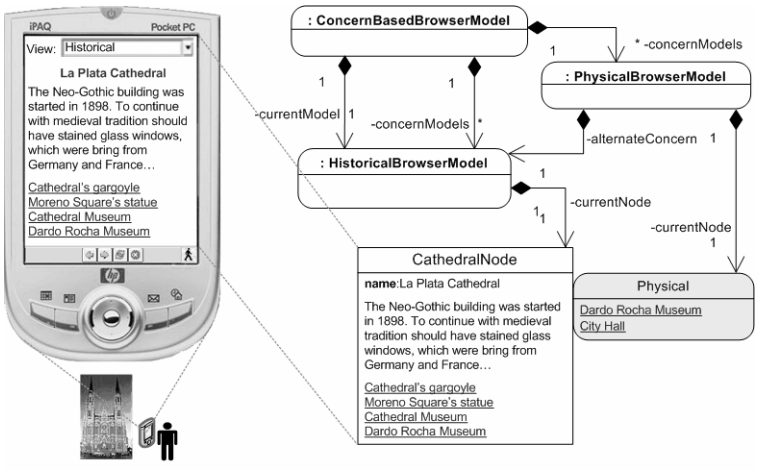

Figure 14: The user is sensed facing the Cathedral.

When the user changes to the physical view, the implicit physical links are computed according to the *alternateConcern*. The planned routes are the static physical links defined in the physical role of the node (the planned routes). On top of that, a set of suggested links are presented to the user (see Figure 15). The *Suggestion* tabs show two links (*Cathedral Museum* and *Moreno Square's statue*) which have been calculated from the links of the *currentNode* of the *alternativeCocnern.* By analyzing the figure, we can see that this *currentNode* has four links, but only the *Cathedral Museum* and *Moreno Square's statue* have a physical role.

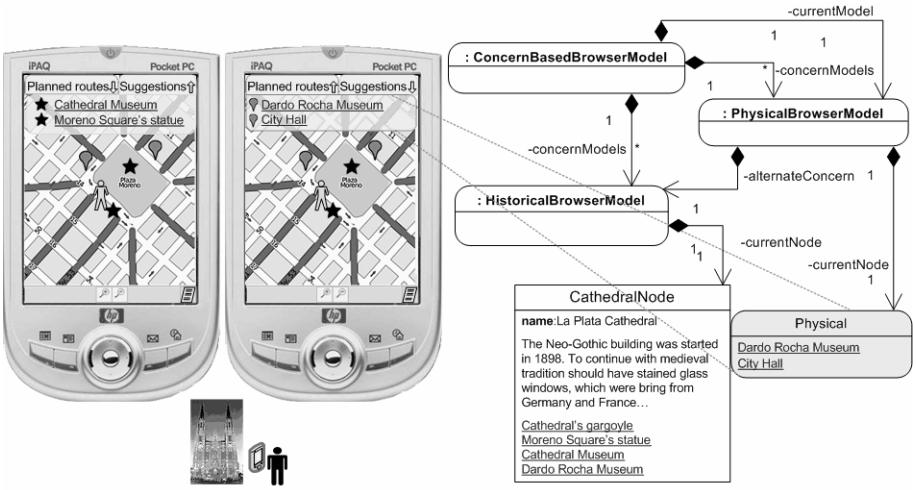

Figure 15: The user seeing the physical concern.

To see how our design supports navigation through different concerns in a homogeneous way, suppose that the user changes to the historical concern and decides to

navigate (digitally) to the *Dardo Rocha Museum*. The currentNode of the *Historical-BrowserModel* has changed and it is now associated to the *Dardo Rocha Museum* node. If the user wants to change to the physical concern view, he will see the same "*Planned Route*". But the Suggestion is computed according to the *alternateConcern* which has changed its *currentNode.* Notice that, the *Suggestion* only adds the links to the *Natural Science Museum,* which has a physical role and is not included in the set of static physical links (*Planned Route*). This scenario is depicted in Figure 16.

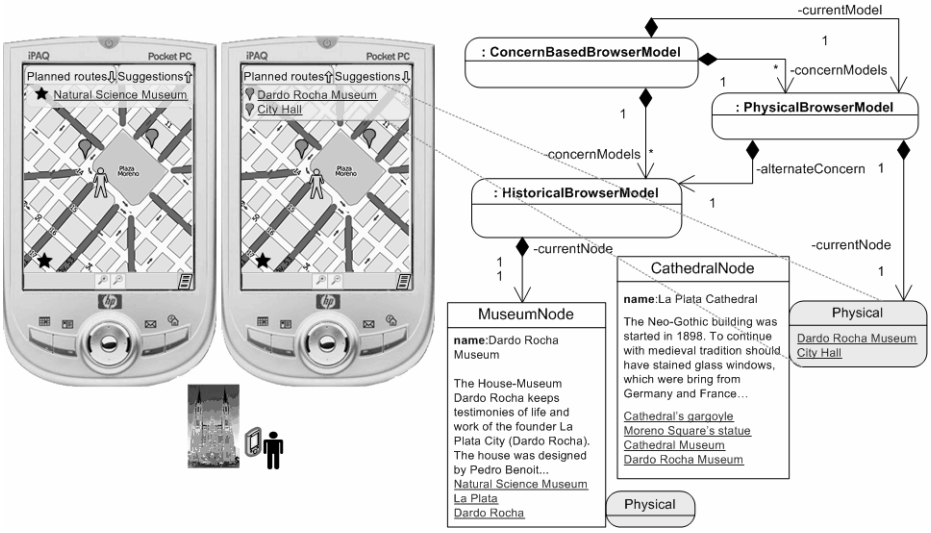

Figure 16: The user is seeing the physical concern of the Cathedral (before the digital navigation).

9. CONCLUSIONS AND FUTURE WORK

In this chapter we have presented our experience in the development of a design model and support architecture for mobile hypermedia software. In particular, we have focused on the most relevant decisions related to separation of concerns in our project. We have shown the need to decouple different architectural concerns, such as adaptation, sensing and core functionality, and have presented a simple set of micro-architectural constructs which not only provide the needed set of behaviors also what is more important, can be seamlessly extended to other application or adaptation domains, can be configured to new technological advances, to support more complex context models, etc. We have briefly presented the basic ideas behind our modeling and design approach, a light extension of the OOHDM design framework. We have shown our strategy to separately model different navigational concerns, and how we use this separation to improve navigation in the final application.

We are currently working in several research topics related to the problems described in this paper.

a) As mentioned in the paper, evolution is a key problem in this kind of software. In this context, we are exploring variability issues in different product families (e.g. mobile hypermedia), to improve maintenance and evolution, by building customized development frameworks for specific domains.
b) Automating the development process of mobile hypermedia should improve productivity of the software development team. We are working to improve the process of generating a running application from design models by applying well-known techniques of model-driven software development (Moreno, Romero, & Vallecillo, 2008; Sami, Matthias, & Volke, 2005; Seidman & Ritsko, 2006) such as model to model transformations (Koch, 2006).
c) Related with the previous problem we are studying how to generate applications that may run in different run-time environments.
d) Finally, we are working on different applications of the mobile hypermedia paradigm. Particularly we are exploring educational applications following the ideas in (Hansen, Kortbek, & Grønbæk, 2008) adapting them to our underlying models.

References

ARTags. Retrieved Feb 21, 2009, from http://www.artag.net/

Bolchini, C., Curino, C. A., Quintarelli, E., Rossato, R., Schreiber, F. A., & Tanca, L. (in press). And what can context do for data?. *Communications of the ACM*.

Bricon-Souf, N., & Newman, C. R. (2007). Context awareness in health care: A review. *International Journal of Medical Informatics, 76*(1), 2-12.

Challiol, C., Rossi, G., Gordillo, S. E., & De Cristófolo, V. (2006). Designing and Implementing Physical Hypermedia Applications. In M. Gavrilova, O. Gervasi, V. Kumar, C.J. K. Tan, D. Taniar, A. Laganà, Y. Mun, & H. Choo (Eds.), *Computational Science and Its Applications - ICCSA 2006: Vol. 4*. (pp. 148-157). Springer Berlin / Heidelberg.

Challiol, C., Muñoz, A., Rossi, G., Gordillo, S. E., Fortier, A., & Laurini, R. (2007). Browsing Semantics in Context-Aware Mobile Hypermedia. In G. Kotsis, D. Taniar, E. Pardede, and I. Khalil (Eds.), *MoMM '08: Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia* (pp. 211-221). ACM.

Challiol, C., Fortier, A., Gordillo, S. E., & Rossi, G. (2007). A Flexible Architecture for Context-Aware Physical Hypermedia. In G. Kotsis, D. Taniar, E. Pardede, & I. Khalil

(Eds.), *DEXA '07: Proceedings of the 18th International Conference on Database and Expert Systems Applications* (pp. 590-594). IEEE Computer Society.

Cheverst, K., Mitchell, K., & Davies, N. (2002). The role of adaptive hypermedia in a context-aware tourist GUIDE. *Communications of the ACM, 45*(5), 47-51.

Dourish, P. (2004). What we talk about when we talk about context. *Journal of Personal and Ubiquitous Computing, 8*(1), 19-30.

Eisenstein, J., Vanderdoncki, J., & Puerta, A. (2000). Adapting to Mobile Contexts with User-Interface Modeling. *WMCSA '00: Proceedings of the Third IEEE Workshop on Mobile Computing Systems and Applications* (pp. 83-92). IEEE Computer Society.

Fons, J., Pelechano, V., Albert, M., & Pastor, O. (2003). Development of Web Applications from Web Enhanced Conceptual Schemas. In G. Goos, J. Hartmanis, & J. van Leeuwen. (Eds.), *Conceptual Modeling – ER 2003* (pp. 232-245). Springer Berlin / Heidelberg.

Fortier, A., Cañibano, N., Grigera, J., Rossi, G., & Gordillo, S. E. (2006). An Object-Oriented Approach for Context-Aware Applications. In W. De Meuter (Ed.), *Advances in Smalltalk* (pp. 23-46). Springer Berlin / Heidelberg.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (Ed.). (1995). *Design Patterns*. Addison-Wesley Professional.

Gordillo, S. E., Rossi, G., & Schwabe, S. (2005). Separation of Structural Concerns in Physical Hypermedia Models. In O. Pastor, & J. Falcão e Cunha (Eds.), *Advanced Information Systems Engineering* (pp. 446-459). Springer Berlin / Heidelberg.

Grønbæk, K., Kristensen, J. K., Orbæk, P., & Eriksen, M. E. (2003). Physical Hypermedia: Organizing Collections of Mixed Physical and Digital Material. *HYPERTEXT '03: Proceedings of the fourteenth ACM conference on Hypertext and hypermedia* (pp. 10-19). ACM.

Hansen, F. A., Bouvin, N. O., Christensen, B. G., Grønbæk, K., Pedersen, T. B., & Gagach, J. (2004). Integrating the Web and the World: Contextual Trails on the Move. *HYPERTEXT '04: Proceedings of the fifteenth ACM conference on Hypertext and hypermedia* (pp. 98-107). ACM.

Hansen, F. A., & Bouvin, N. O. (2009). Mobile Learning in Context - Context-aware Hypermedia in the Wild. *International Journal of Interactive Mobile Technologies, 3*(1), 6-21.

Hansen, F. A., & Grønbæk, K. (2008). Social web applications in the city: a lightweight infrastructure for urban computing. *HT '08: Proceedings of the nineteenth ACM conference on Hypertext and hypermedia* (pp. 175-180). ACM.

Hansen, F. A., Kortbek, K. J., & Grønbæk, K. (2008). Mobile Urban Drama - Setting the Stage with Location Based Technologies. In U. Spierling, & N. Szilas (Eds.), *Interactive Storytelling* (pp. 20-31). Springer Berlin / Heidelberg.

Jones, V., & Jo, J. H. (2004). Ubiquitous learning environment: An adaptive teaching system using ubiquitous technology. In R. Atkinson, C. McBeath, D. Jonas-Dwyer, & R. Phillips (Eds.), *Beyond the comfort zone: Proceedings of the 21st ASCILITE conference* (pp. 468-474). Perth.

Lamming, M., & Flynn, M. (1994). Forget-me-not: Intimate computing in support of human memory. *FRIEND21: International symposium on next generation human interface* (pp. 125-128).

Leonhardt, U. (1989). *Supporting Location-Awareness in Open Distributed Systems*. Unpublished doctoral dissertation, Department of Computing, Imperial College London, UK.

Kay, A. C. (1993). The early history of smalltalk. *ACM SIGPLAN Notices*, *8*(3), 69-95.

Koch, N. (2006). Transformation Techniques in the Model-Driven Development Process of UWE. In N. Koch, & L. Olsina (Eds.), ICWE *'06: Workshop proceedings of the sixth international conference on Web engineering* (Article No. 3). ACM.

Koch, N., & Kraus, A. (2002). The expressive power of UML-based web engineering. In D. Schwabe, O. Pastor, G. Rossi & L. Olsina (Eds.) *Second International Workshop on Web-oriented Software Technology (IWWOST02)* (pp. 105-119). Springer-Verlag.

Krasner, G. E., & Pope, S. T. (1988). A Cookbook for Using Model-View-Controller User Interface Paradigm in Smalltalk-80. *Journal of Object Oriented Programming*, *1*(3), 26-49.

Moreno, N., Romero, J. R., & Vallecillo, A. (2008). An Overview Of Model-Driven Web Engineering and the Mda. In G. Rossi, O. Pastor, D. Schwabe, L. Olsina (Ed.), *Web Engineering. Modelling and Implementing Web Applications,* (pp. 353-382). Springer.

Nanard, J., Rossi, G., Nanard, M., Gordillo, S. E., & Perez, L. (2008). Concern-Sensitive Navigation: Improving Navigation in Web Software through Separation of Concerns. In Bellahsène Z., & Léonard M. (Eds.), *Advanced Information Systems Engineering* (pp. 420-434). Springer Berlin / Heidelberg.

Pascoe, J. (1998). Adding generic contextual capabilities to wearable computers. *ISWC '98: Proceedings of the 2nd IEEE International Symposium on Wearable Computers* (pp. 420-434). IEEE Computer Society.

Quinlan, R. J. (1986). Induction of Decision Trees. *Machine Learning, 1*(1), 81-106.

Rossi, G., Gordillo, S.E., Challiol, C., & Fortier, A. (2006). Context-Aware Services for Physical Hypermedia Applications. In R. Meersman, Z. Tari, & P. Herrero (Ed.), *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops* (pp. 1914-1923). Springer Berlin / Heidelberg.

Rossi, G., Nanard, J. Nanard, M., & Koch, N. (2007). Engineering Web Applications Using Roles. *Journal of Web Enginnering, 6*(1), 19-48.

Salber, D., Dey, A. K., & Abowd, G. D. (1999). The Context Toolkit: Aiding the Development of Context-Enabled Applications. *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 434-441). ACM.

Sami, B., Matthias, B., & Volke, G. (Ed.). (2005). *Model-Driven Software Development*. Springer Press.

Schilit, B. N., Adams, N., & Want, R. (1994). Context-Aware Computing Applications. *WMCSA '94: Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications* (pp. 85-90). IEEE Computer Society.

Schwabe, D., & Rossi, G. (1998). An object-oriented approach to web-based application design. *Theory and Practice of Object Systems, 4*(4), 207-225.

Seidman, D. I., & Ritsko, J. J. (2006). Model-Driven Software Development. In IBM Systems Journal, *45*(3). IBM Corp.

Stefanidis, K., Pitoura, E., & Vassiliadis, P. (2005). On Supporting Context-Aware Preferences in Relational Database Systems. In W. Mansoor, M. Khedr, D. Benslimane, Z. Maamar, M. Hauswirth, & K., Aberer (Eds.), *Proceedings of the First International Workshop on Managing Context Information in Mobile and Pervasive Environments*.

UWA (Ubiquitous Web Applications) Project. http://www.uwaproject.org

Wirfs-Brock, R., & McKean, A. (Ed.). (2002). *Object Design: Roles, Responsibilities and Collaborations*. Addison-Wesley Press.

KEY TERMS & DEFINITIONS

Hypermedia: Hypermedia is a paradigm for organizing and accessing to information by organizing it in a network, in which *nodes* contain multime-

dia data and are connected with *links*. The user traverses the information space by navigating from node to node using links.

Mobile Hypermedia: Mobile hypermedia extends the concept of hypermedia. The user, carrying his mobile devices traverses not only the virtual space but also can receive information from the real-world places (physical nodes) that he visits during his detour.

Physical Link: A physical link connects two physical objects. The user traverses a physical link by moving from the source to the target physical objects, instead of waiting that the underlying system "brings" him the information. Navigation in these links is not atomic as may take time, be interrupted, deferred or canceled by the user who changed his decision and moves to another place.

Concern: In the context of this chapter, a concern comprises a set of coherent application requirements; it can therefore encompass an important application theme or domain area. By separating concerns we achieve modularity as we can reason and operate on them independently.

Context: Context is any information that is considered relevant to characterize the interaction between a user and the application, for example the user's location, identity, role, current activity, network or device's features, etc.

Context-Aware: An application is said to be context-aware when it considers the actual context to provide better information and/or services to the user.

OOHDM: The Object-Oriented Hypermedia Design Method is a mature approach for building hypermedia and Web applications by describing different design models which are then mapped onto a running application.

Navigational Model: In OOHDM and other design methodologies, the navigational model specifies in an abstract way, the application's navigation topology, i.e. the node types and their contents, and the links and other navigation structures (such as indexes) which connect nodes.