Browsing Semantics in Context-Aware Mobile Hypermedia

Cecilia Challiol[1,3], Agustin Muñoz[1], Gustavo Rossi[1,3], Silvia E. Gordillo[1,4] ,
Andrés Fortier[1,2,3], Robert Laurini[5]
[1] LIFIA. Facultad de Informática. UNLP. La Plata, Argentina
{ceciliac,agustinm,gustavo,gordillo,andres}@lifia.info.unlp.edu.ar
2 DSIC. Universidad Politécnica de Valencia. Valencia, España
[3] Also CONICET, [4] Also CICPBA
[5] LIRIS. Universite de Lyon. France. robert.laurini@insa-lyon.fr

Abstract. Mobile hypermedia applications combine the well-known advantages of the navigational paradigm of the Web with the capabilities of location-aware software. However, there are some subtleties to integrate them synergistically. In this paper we analyze different aspects related with navigation semantics in mobile hypermedia; in particular we discuss the problems which arise in the use of the familiar backward and forward operations when physical navigation in the real world is involved. Using a motivating example, we present a simple model to handle physical and digital navigation in a cohesive way. We also describe a modular implementation of our ideas in an architecture which support context-aware services.

1 Introduction and Motivation

In the last years there has been a growing interest to integrate the navigation paradigm of the Web with the capabilities which are usual in mobile, context-aware software [8,9]. In these systems, the mobile user navigates physically by traversing the real world or digitally by following links. The underlying hypermedia network is therefore formed out of a set of physical and digital nodes; while digital links are just the "old" Web links, physical ones require moving in the physical space.

Suppose for example a tourist in a city: when he stands in front of a remarkable place (e.g. a monument) he receives digital information about that place in his mobile device. This location-aware behavior can be thought as equivalent to opening a Web page; however, in this case "opening" means accessing physically. This page may contain links to other pages, which can be pure digital (i.e. conventional hyperlinks) or may point to other physical locations. When the user chooses a digital link, he explores the hyperspace and does not need to move. Meanwhile, if he selects a "physical" link he receives information on how to move to the target of the link (a physical place). In this case, traversing the link means moving to another location, what is referred to as "walking" the link [10]. However, during this trip, the user might get lost or decide to visit other monuments. Navigation in the real world is not atomic as we are used to in the virtual world. In this context, browsing semantics such as the

interpretation of the *back* and *forward* operations are an important issue for the mobile user.

Figure 1 shows a sketch of the city our tourist is traversing and some relevant points of interest. At the beginning of his tour the user is in front of the Museum, and therefore he is accessing the corresponding digital node. One of the (physical) links points him to the Theatre; he selects it and gets a map to initiate his trip. The user walks according to the map and then he arrives to the Theatre. In that moment, the user selects the (physical) link to the Cathedral and gets a map with the travel. While walking, he passes by the Football Stadium, stops in front of it, gets some information for a future visit and decides to continue to the Cathedral.

Figure 1 also shows a simple graph indicating this trajectory. It is easy to see that his "navigation history" includes (in order): Museum, Theatre, Football Stadium and Cathedral. According to this sequence it is natural to think that the default implementation of the back button would be to help him return to the previous place, while the next button should give him cues regarding the rest of the trip.

The situation might get more complicated if during this trip he faces other physical objects that behave as assistants [4], either correcting his tour, giving further information or encouraging to follow his trip. Should these objects be part of the navigation history? Moreover, suppose that he arrives to the Cathedral and wants to return to the Museum. Is it necessary that he traverses the same path even when he can use a shortcut? (see Figure 1). In addition, suppose that in each stop the user navigates the digital hyperspace; how should the (digital) back operation behave? The situation might be more complicated if we try to provide context-aware behavior, such as eliminating a place from the history if it is closed, not accessible or not in the user's preferences.

Fig. 1. A simple city tour scenario.

These are just a set of the problems one faces when combining the physical and digital worlds in a hypermedia setting. Some of the problems might depend on the application domain, others on the user's context, while others are just the consequence of mapping the hypermedia metaphor to the real world. While it is not possible to find a solution suitable for all cases, we aim to provide a conceptual and application framework to provide different navigation behaviors according to the user's needs.

In this paper we analyze the problem of dealing with different browsing semantics in a coherent way. We show that a mobile user requires varied strategies for backward

and forward navigation, both according to his actual context and intended task; we also describe an architectural approach and its associated implementation to deal with these issues modularly.

The main contributions of our paper are the following:

- We characterize the problem raised by browsing semantics in mobile hypermedia applications.
- We outline a model to reason on (physical and digital) navigation in this kind of software.
- We present a novel approach to decouple the navigation semantics from the underlying browsing software in order to improve application's modularity.

The rest of the paper is organized as follows: In Section 2 we present a model for dealing with forward and backward navigation. In Section 3 we describe our architectural support. In Section 4 we discuss some related works and we conclude in Section 5 describing some further work we are pursuing.

2 Forward and Backward Navigation in Mobile Hypermedia

We will use the standard graph representation both for digital and physical hypermedia and we will assume that there are two different graphs, one encompassing the physical objects the user can visit and the other corresponding to the digital documents he can navigate with his mobile device. As we will explain below, some of these digital nodes are the counterpart of physical objects. Regardless the nature of the graph (digital or physical) we represent the user's path as the list of nodes he has traversed; each time a new node is visited, it is added at the end of the user's path. The next step is adding the *back* and *next* functionality, which in principle should match the standard browser semantics (i.e. Stack-Based Navigation [2]). To do so we define:

$back : (Path \times Index) \rightarrow Node$

$back(p,i) = element(p, i-1) \qquad \forall\ i\ ,\ 1 < i <= size(p)$

$next : (Path \times Index) \rightarrow Node$

$next(p,i) = element(p, i+1) \qquad \forall\ i\ ,\ 1 <= i < size(p)$

It is important to notice that these functions do not modify the user's current situation; they just return, based on a specific criterion, what the previous or next node is, given a path and the current active node.

2.1 Physical Navigation

In the case of physical navigation, there is an important issue in terms of visited nodes, since navigation is not atomic and therefore a user can visit a physical node without explicitly asking for it. In our example the user decides to physically navigate from the Museum to the Theatre and then to the Cathedral. In his way he passes by different physical objects of which the system is aware (e.g. ATMs, traffic lights, etc.). Thus, even though the user has not explicitly asked to navigate to those intermediate objects,

we can think of them as part of his navigation history and we can use them to improve user guidance [13]. In this case, when the user walks from the Museum to the Cathedral he had passed by four physical objects (a traffic light, the Theatre, an ATM and the Football Stadium).

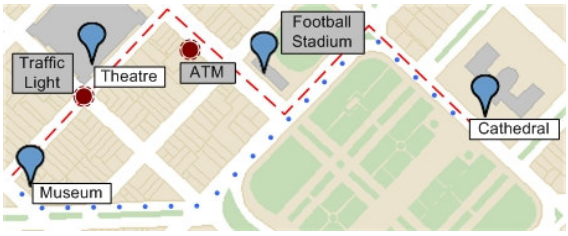

Fig. 2. An extended map showing points of interest and physical objects.

From this scenario (Figure 2), we could think of applying two different back (and next) functions: one that takes into an account all visited objects ($Back_{Full}$) and other that only looks at objects that were explicitly requested by the user ($Back_{Simple}$). Assuming that the user is standing in the Cathedral, this scenario would be modeled as:

Path={Museum, TrafficLight, Theatre, ATM, FootballStadium, Cathedral}

$Back_{Full}(path,6) \rightarrow FootballStadium$

$Back_{Simple}(path,6) \rightarrow Theatre$

The situation might get more complicated if we want to filter the objects according to contextual information, such as the user activity or actual task.

2.2 Integrating Physical and Digital Navigation Semantics

Mobile hypermedia combines physical and digital navigation. To show how to integrate the physical and digital models, we will refer to the set of digital hypermedia nodes as N_{dig} and to the set of physical hypermedia objects as N_{ph}. Using this notation we can define a function that maps physical objects to their digital counterparts:

$Map_{ph\text{-}dig} : N_{ph} \rightarrow (N_{dig} \cup \{\theta\})$

where θ corresponds to those physical objects without a digital counterpart

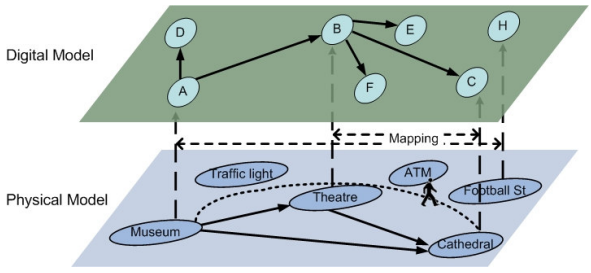

Fig. 3. Mapping physical and digital models.

In the example of Figure 3 we have:

- N_{dig}={A,B,C,D,E,F,H}
- N_{ph}={Museum, Theatre, Cathedral, TrafficLight, ATM, FootballStadium}
- $Map_{ph\text{-}dig}$={(Museum,A), (Theatre,B), (FootballStadium,H), (Cathedral,C)}

We can now characterize physical objects according to their relationship with digital objects and vice versa:

- PurePhysicalObejcts={n | n ∈ N_{ph} ∧ $Map_{ph\text{-}dig}$(n) = θ }
- PointsOfInterest={n | n ∈ N_{ph} ∧ $Map_{ph\text{-}dig}$(n) ≠ θ }
- PureDigitalObjects={d | d ∈ N_{dig} ∧ ~ (∃ n ∈ N_{ph}) ($Map_{ph\text{-}dig}$(n) = d)}

By using this characterization we can offer the user a wider range of functions to give back and next semantics. As an example, a user may find useful to include in the back functionality the objects that he has encounter along his path, but only if they are points of interest. As a result, we would have now a new back function for Figure 2:

Path={Museum, TrafficLight, Theatre, ATM, FootballStadium, Cathedral}

$Back_{PoI}$ (path, 6) → FootballStadium *(consider only objects in PointsOfInterest)*

This idea can be extended as far as needed, adapting the back and next functionality according to different parameters. As an example consider these two scenarios:

- We could tag points of interest according to different characteristics. For example the Museum, Theatre and Cathedral would be tagged as *cultural* points of interest, while the Football Stadium would be tagged as *recreational.* A "smart" back could inspect tags to decide weather a node is taken into an account or not. In this case, since the football stadium was not explicitly asked by the user and it is the only one that is not tagged as *cultural*, when asked for the previous point of interest, $Back_{Tagged}$ (path, 6) would answer Theatre.
- If we have a function that retrieves opening times of points of interest, the back function could take into an account only those points of interest that are open.

These examples show that we should be able to create as many back functions as we need. Of course not all of them are well suited for every situation: while $Back_{PoI}$ seems reasonable for a tourist that knows where he is, it does not apply for a tourist that is lost (and especially if the visited points of interest are far away from each other). Moreover, this kind of functionality might be application dependent, e.g. if the user is a postman delivering letters, he may need a back function that shows the places in which letters could not be delivered (e.g. because there was nobody at home). As a consequence, an architecture that supports the development of mobile hypermedia applications should allow this variability, which means that it should clearly separate these aspects to simplify the development process. In the next sections we describe the main design decisions in our architecture to support varied browsing strategies.

3 Architectural Support

We have built a substrate for developing mobile hypermedia applications as an extension of our previous work on context-aware services [4, 13]. The first and most impor-

tant decision to achieve flexibility in browsing semantics is to decouple the browser functionality from the model it is rendering. Generally, a Web browser is seen as an application that sends HTTP requests and renders the response. In the next sub-section we will present a different perspective for using a Web browser in a mobile setting, and show how navigation functionality can be configured according to the application's need. Next we will briefly explain our base architecture for context-aware behavior and how we used it to provide context-aware browsing.

3.1 Decoupling Navigation from the Browser

In order to achieve the required functionality we consider and use the Web browser as a medium that encompasses the view and the controller of a typical MVC architecture [12] (see Figure 4). As a result the navigational model (including its browsing semantics) is decoupled from the browser, which gives us great flexibility.

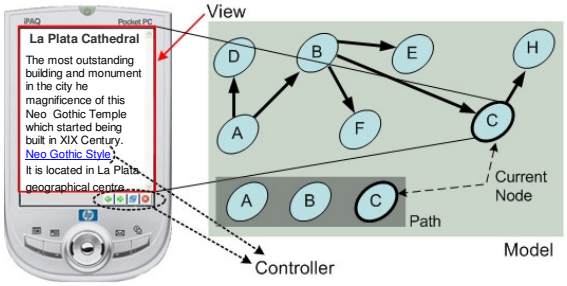

Fig. 4. The browser as the view component of an MVC application.

In this approach the MVC components will be materialized as:

- **View:** an object's view is just a standard HTML document that is generally created in a dynamic fashion (e.g. as a result of processing a remote jsp). In this scenario, the web browser takes the role of an HTML renderer.
- **Controller:** in hypermedia applications we are mainly interested in navigation events. These events result from the user clicking on a link anchor or using the standard toolbar functionality (*back*, *next*, *home*, and *update* buttons). However, the interpretation of these behaviors should be provided by the model and not by the browser; the controller just "captures" them and delegates their execution. To support this view we have used callbacks to intercept these events and redirected them to the browser's model. Even though we implemented this schema in Twoflower, a simple Smalltalk browser, most actual web browsers provide callbacks to capture navigation events.
- **Model:** the browser model is in charge of adapting the underlying application model to comply with the browser required interface. Internally the browser model maintains the static structure (i.e. the nodes and the links), keeps track of the navigational state (i.e. the path and current node) and defines the navigational behavior by means of the back and next functions.

By defining a clear message protocol for the model, we can use a Web browser for any application type in which the underlying domain can be "navigated", such as Web sites, mobile hypermedia or workflows. To do so, the web browser's model should comply with the IBrowserModel interface (see Figure 5). To ease the development process we have created an abstract class (BrowserModel) that implements almost all the required behavior; the only abstract message that must be redefined by its subclasses is the *doNavigateTo(String nodeDescription)* that takes a string that identifies an object and should return the corresponding IBrowserNode object. As an example, implementing a mobile Web browser using this small framework is straightforward: the WebBrowserModel interprets the nodeDescription as a url and just creates a WebBrowserNode. When asked for its html representation, the WebBorwserNode issues an http request and answers its result.

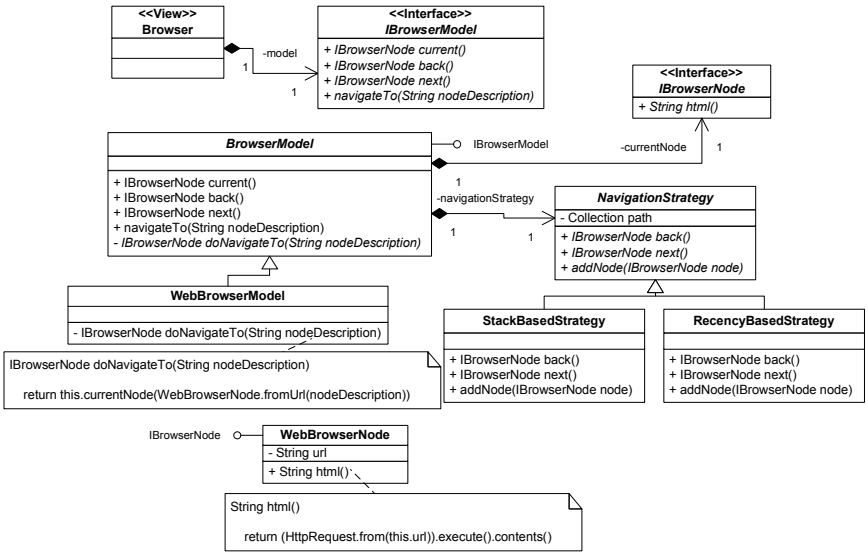

Fig. 5. Class diagram showing how to decouple the browser from its model.

Finally, notice that the BrowserModel class not only provides a stub for the IBrowserModel interface, but also decouples the back and next functionality by delegating it to a navigation strategy [5]. By default the StackBasedStragety is used, but it can be changed according to the application needs. To do so, a subclass of NavigationStrategy must be defined and override the *back()* and *next()* abstract messages. As an example, the class diagram shows a subclass that implements a recency based strategy [6]. Finally, if we have a mapping function between different browser models (such as the $Map_{ph\text{-}dig}$ function presented before), the user is able to switch between different views of the same object (e.g. the digital description of the Museum and a map showing where it is located).

3.2 Context-dependent Behaviors

In this section we will show how to take advantage of the gained flexibility of decoupling the browser's model from the view by changing the back and next functionality according to the user's behavior and context. By default, the system is initialized with the $Back_{PoI}$ strategy, which only takes into an account the points of interest visited by the user. If, by analyzing the way the user moves around the city, the system "realizes" that he can not easily follow the paths, the strategy is replaced by the $Back_{Full}$, which shows all the physical objects the user has passed by.

Following the approach presented in [4] we used a three-layered approach to decouple those concerns that evolve independently (a complete instance diagram is presented in Figure 6):

- **Application Model:** this is the model that has to be enhanced with new, context-aware functionality. In mobile hypermedia software, this role is played by the hypermedia application (i.e. the "Model" in the MVC triad described in 3.1).
- **Context Model:** it is in charge of modeling the information that is contextually relevant. Since we must reason about the way the user traverses his path, we model him as an aware object with two context features: his location and the target of his actual navigation. The *location* feature will be updated when a location sensor changes (e.g. receiving a beacon signal or a GPS coordinate), while the *navigationTarget* will be changed by a software sensor that is triggered every time the browser sends the *navigateTo* message to its model (for more information regarding decoupling sensing mechanisms from context model see [7]).
- **Context Adaptation:** a context handler is created to act as an Observer [5] of an aware object to get notified of context changes. This handler will be triggered each time the user asks for a new navigation or changes his current position. If by analyzing his behavior the system concludes that the user is lost, the handler will change the navigation strategy of the browser model. Since this paper is not concerned with describing how to infer if the user is lost, we assume that the handler collaborates with a reasoning engine, which can have data structures of its own.

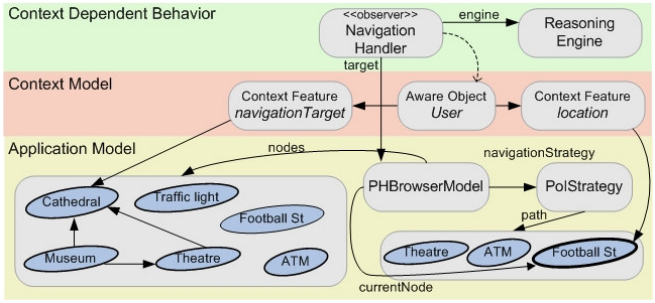

Fig. 6. Instance diagram of a simple context-aware mobile hypermedia application.

As a result, by decoupling the aspects that are inherently independent (such as the context model, the context-dependent behavior and the sensing mechanism) we can add specialized behavior with almost no impact in the original application model. For example, to skip closed points of interest from the path we would have to create an aware object for each point of interest, with a *schedule* context feature. Then a new navigation strategy would be defined to filter out those points of interest that are closed. Thus, the required modifications to the application model to achieve this functionality are minimal.

4 Related Work

Our work has some points in common with [3], where the authors present a model and infrastructure for *browsing the world* with added context information. In contrast with our work, they design and implement an infrastructure based on a 4-tuple context model (Who, What, Where and When) while our context model is configurable by adding or removing context features. Also, the system in [3] is oriented towards querying a tuple space, while we take an event-based approach. We could mimic this behavior by imposing 4 context features to the aware objects and defining one handler that reacts to a context change by adding the aware object to a tuple space.

In [6] a recency-based back behavior (an alternative to stack-based back functionality) is presented. In [1] the authors present task-based backtracking, a technique for backtracking within the various logical tasks a user may be working on at any given time. Since the back and next functionalities in our approach can be configured, they could easily match the previous criteria.

A Mobile Tourist information Systems (TIP) is presented in [11], where the authors mention different user interactions, like *Browsing by walking* or going *To last location*. Even tough our work has some points in common with TIP (since the user can switch between a browser and map view while he is walking), we include the concept of physical navigation that is not present on TIP. The back button offers the user the chance to return to the last information that was delivered, before he started browsing and/or walking around, which can be compared with our digital navigation. Finally in [9] the author presents a framework for context-aware mobile hypermedia (HyCon) which has four different context-aware techniques: browsing, search, annotation, and linking. The users can “browse information with their feet”, simply by moving about in the world. In our system the user can also browse digital information by moving in the physical world, but we attempt to go a step further and extend the browser metaphor to other domains in which navigation is used.

5 Concluding Remarks and Further Work

We have presented a characterization of browsing semantics in mobile hypermedia applications. We showed that different browsing strategies should be applied to physical navigation according to user preferences, contextual information or actual user behavior. The key point in our approach is to decouple the underlying model from the

browser software and to configure it by means of the back and next functions as object strategies. In this way, we were able to create as many back and next functions as necessary according to different parameters or to the semantics of the. We have also shown that any model that complies with the IBrowserModel interface (e.g. web sites, mobile hypermedia or workflows) can be navigated with the standard Web browser semantics.

So far we have defined browser models to navigate the Web and a network of physical objects. We are currently working on adding workflows to support task-oriented physical hypermedia, and to fully characterize the notion of "navigable" models. We are also working on interface issues to easily manage related models, so that the user can view different aspects of the same object (e.g. digital information or the tasks related to it) without loosing context information (e.g. where the user is physically standing). As a long term goal we expect to have a dedicated framework which supports the most used models and navigation strategies.

References

1. Bieber M., Jiangling W.: Backtracking in a Multiple-window Hypertext Environment. In Proceeding of ECHT '94, ACM Press, 1994, pp. 158-166.
2. Bieber M., Vitali F., Ashman H., Balasubramanian V., Oinas-Kukkonen, H.: Fourth generation hypermedia: some missing links for the World Wide Web. In Journal of Human-Computer Studies, 1997, pp. 31-65.
3. Castelli G., Rosi A., Mamei M., Zambonelli F.: A Simple Model and Infrastructure for Context-Aware Browsing of the World. In Proceeding of PERCOM '07, IEEE Computer Society, 2007, pp. 229-238.
4. Fortier A., Challiol C., Rossi G., Gordillo S.: Physical Hypermedia: a Context-Aware approach. In Proceedings of the CAiSE'07, 2007, pp. 499-513.
5. Gamma E., Helm R.. Johnson R., Vlissides J.: Design Patterns. Elements of reusable object-oriented software. Addison Wesley 1995.
6. Greenberg S., Cockburn A.: Getting Back to Back: Alternate Behaviors for a Web Browser's Back Button. In Proceeding of the 5th Annual HFWeb, 1999.
7. Grigera J., Fortier A., Rossi G., Gordillo S.: A Modular Architecture for Context Sensing. In Proceedings of PCAC-07, IEEE Computer Society, 2007, pp.147-152.
8. Gronbaek K., Kristensen J., Eriksen M.: Physical Hypermedia: Organizing Collections of Mixed Physical and Digital Material. In Proceedings of Hypertext 2003, 2003, pp. 10-19.
9. Hansen F. A.: Context-aware Mobile Hypermedia: Concepts, Framework, and Applications. Ph.D. Dissertation, Department of Computer Science, University of Aarhus, 2006.
10. Harper S., Goble C., Pettitt S.: proXimity: Walking the Link. In Journal of Digital Information, Volume 5, Issue 1, Article No 236.
11. Hinze A., Malik P., Malik R.: Interaction design for a mobile context-aware system using discrete event modeling**.** In Proceedings of the ACSC'06, Australian Computer Society, 2006, pp. 257-266.
12. Krasner G., Pope S.: A Cookbook for Using Model-View-Controller User Interface Paradigm in Smalltalk-80. In Journal of Object Oriented Programming, 1988, pp. 26-49.
13. Rossi G., Gordillo S., Challiol C., Fortier A.: Context-Aware Services for Physical Hypermedia Applications. In Proceedings of the CAMS 2006, Springer-Verlag Berlin Heidelberg, 2006, pp. 1914-1923.