A Flexible Architecture for Context-Aware Physical Hypermedia *

Cecilia Challiol[1], Andrés Fortier[1,2], Silvia Gordillo[1,3], Gustavo Rossi[1,4]
[1]LIFIA. Facultad de Informática. Universidad Nacional de La Plata, Argentina.
[2]DSIC. Universidad Politécnica de Valencia, Valencia, España.
[3]Also at CICPBA, [4]Also at CONICET
{ceciliac, andres, gordillo, gustavo}@lifia.info.unlp.edu.ar

Abstract

In this paper we present the rationale and the main components for a modular and extensible architecture for building and deploying physical hypermedia software. We show that this kind of software systems poses strong requirements on supporting software (such as Web browsers) because they involve complex context-aware navigation semantics. By using some simple archetypical examples we also show how to provide context-aware assistance to the mobile user, as he explores the physical world.

1. Introduction and Background

A physical hypermedia application (PH from now on) is a kind of mobile, ubiquitous application in which the user navigates through digital and physical objects (e.g. with a Web browser) using the well-known hypermedia paradigm. Many authors have already formalized the ideas behind PH as a way to integrate the Web and the world [8], to provide assistance to the traveler [9], or to implement augmented reality applications [10]. PH supposes a basic location sensing mechanism that allows a user to perceive digital information corresponding to real world objects while he stands in front of them. These objects can also "offer" navigation links either in the traditional way, or pointing to other physical objects. The user can therefore navigate virtually (as in the Web) or physically by "walking" the corresponding link [9]. In traditional hypermedia applications, clicking a link implies opening the target page; meanwhile, clicking on a physical link only shows the *desire* of the user to navigate from one physical object to another.

As a simplified example suppose a PH for a tourist in a city, in our case, La Plata. When he stands in front of a place that is a node in the PH (e.g. a monument, church, etc), he receives information about the place together with digital and physical links. If he chooses a physical link, he will receive orientation to traverse the physical space to the target. During his travel he will pass by other physical nodes and surely other physical objects (traffic lights, ATMs, etc). A challenging problem (see Figure 1) is how to provide the user some help in his task, for example by making these objects active assistants (e.g. behaving as a travel guide).

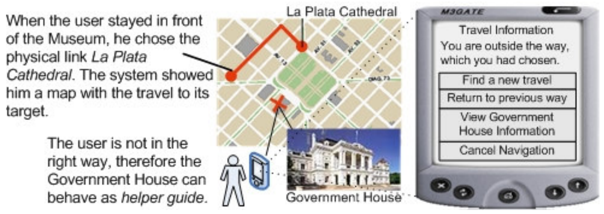

When the user stayed in front of the Museum, he chose the physical link *La Plata Cathedral*. The system showed him a map with the travel to its target.
La Plata Cathedral
The user is not in the right way, therefore the Government House can behave as *helper guide*.
Government House
M3GATE
Travel Information
You are outside the way, which you had chosen.
Find a new travel
Return to previous way
View Government House Information
Cancel Navigation

Fig. 1. Assisting the user in his journey.

We have been working in different aspects of PH development such as design issues [4], server-side support [2] and assistance services [11]. In this paper we present an open architecture for supporting different kinds of context-aware behaviors in PH applications (particularly those which involve assisting the user). Our contribution is twofold: in the field of PH, we show how to provide dynamic assistance to the PH user and, from a more general perspective, we present a set of software abstractions which are useful in the most general field of context-aware applications.

The rest of the paper is organized as follows: In Section 2, we survey the requirements posed by PH software; in Section 3 we describe our architecture. Section 4 discusses some related work and in Section 5 we conclude and present some further work.

2. Requirements for a PH Architecture

In this section we summarize the driving forces which have shaped our architecture. Our main aim was

* *This paper has been partially supported by the SeCyT under the project PICT 13623*

to provide support for powerful, context-aware Web-like browsing, allowing new context types and context-aware behaviors to be seamlessly added to a running system. For the sake of comprehension we separately explain the most important requirements.

2.1. Context-Aware Browsing

In Figure 2 we show the interface of a PH application while the user is standing in front of the Cathedral in the archetypical tourist application. The first difference with "conventional" Web software is that some hypermedia nodes (e.g. the Cathedral) open as the result of a change in the user context (in this case his location). The Web page exhibits information, services, digital and physical links. While the semantics of digital navigation should be preserved, physical links generate a new kind of requirement: how to guide the user to reach the link's target. This implies defining the response of the link itself and which assistance services should be provided.

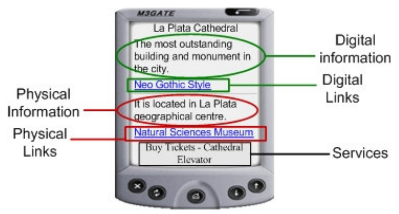


Fig. 2. Interface of a PH application using a web browser.

2.2. Software Engineering Requirements

PH applications are a particular kind of ubiquitous software and as a consequence their evolution pattern differs from traditional software. Besides functionality arising from new user's requirements, PH software might evolve according to technological progress. For example, by adding new kind of sensors, the city can incorporate new on-line services, like querying which roads are closed. This kind of evolution implies a challenge to the supporting architecture: we need to be able to accommodate not only new sensing devices, but also different services and contextual features. We next describe the most important design choices we took to support the previously described behaviors and to make evolution seamless.

3. The architecture in a nutshell

Context is decomposed in a set of features, as shown in Figure 3. These context features affect certain application objects, those that must be aware of context changes (as explained in Section 3.1).

Besides, instead of conceiving context and sensors as two tightly coupled type of software artifacts, we consider sensing as a cross cutting concern. We have shown elsewhere [5] that we can manage the evolution in sensing devices without polluting the context model. For the sake of simplicity, we will assume from now on that IR or Bluetooth beacons are used to tag physical objects. As the same context feature can be used to perform different kind of context-dependent behaviors (e.g. the *Location* feature can be used to provide location-based services or to support physical navigation), we built different combinations of environments and handlers that perform custom adaptation based on those relevant features.

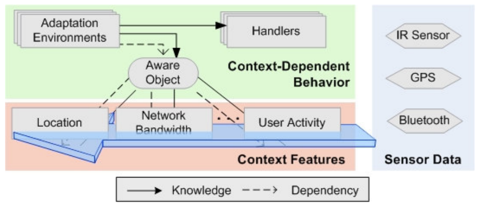


Fig. 3. An architecture for handling context-dependent behavior.

The Web browser is a key component of our architecture. In conventional applications, when a link is clicked an http request is issued and the response is shown to the user. In PH applications this behavior must be slightly modified: first some hypermedia nodes are opened, not because the browser issued an http request, but because the user moved in front of a node; additionally, traversing a physical link is not an atomic operation, since different situations may arise while the user is walking to his target. To cope with these requirements, our browser is considered as a view (in the MVC sense) of a context feature that keeps track of the user's navigation (named *Navigation* feature). This feature holds the current url and the user's navigation history. Each time this feature changes (e.g. as the result of a change of the user's context), the browser receives a notification. When this happens the browser will issue an http request and display the appropriate information as described in detail in Section 3.2 and 3.3. By considering the *Navigation* feature as a part of the context model, we can perform different kinds of customization based on the user's navigational behavior and history.

3.1. Representing Context-Aware Objects

An aware object is an application object (e.g. the user, a monument) that needs to be aware of its context and therefore becomes an Observer [3] of one or more context features. A context feature is a part of the application context (including the user context) that is relevant to be modeled because it will be used to perform some kind of customization. Context features can range from low-level abstractions of data gathered by sensors (e.g. the temperature in a room) to the output of complex machine-learning algorithms (e.g. to infer the user's situation according to his location and the time of the day). In our mobile tourist guide application, we could define the user as an aware object viewing context features such as location, visited points of interest and preferences.

When an aware object is interested in having a specific kind of adaptation, it has to be registered to the desired environment, which in turn becomes an Observer [3] of the aware object. For example, if a user is visiting a new city, he will be interested in registering to the PH environment in order to receive context-aware assistance during his trip. To provide different kinds of context-dependent behaviors, we use *handlers* that are attached to the *adaptation environment*s. Handlers are responsible of performing the concrete customization as a response to a context change.

When a context feature changes (e.g. a new beacon signal has been sensed), it notifies all interested parties. The aware object that observes the feature will receive this notification and it will notify all environments that it is registered to of that context change. The environments will then select those handlers that are interested in this context change and give them the chance to perform their specific adaptation behavior.

As an example, consider assigning services to physical objects to be delivered to the user's device when he is in front of one of those objects. To support this behavior, the user is modeled as an aware object (containing a context feature that keeps track of his location) and registered to a PH environment which includes a ServicesHandler. When the user stands in front of a physical object the ServicesHandler is notified so that old services are removed and new ones added. Suppose that now the application evolves and we want to add time-dependent constraints to the available services. Instead of rewriting the code in the ServicesHandler class, we add a new handler that applies a filter over the full set of services that are available. Finally, we could also want to sort the services in descending order, taking into an account the most used ones. To do so, we create a new context feature that keeps track of the frequency in which the services are used and a handler to perform the sorting of services. By encapsulating this behavior in small grained objects, the application can be modified with small or null impact. In Figure 4 we present a simplified class diagram of these aspects of our architecture.

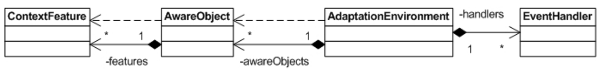


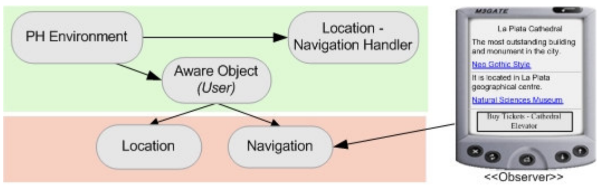
Fig. 4. A simplified view of the architecture's core classes.

In the following sections, we will show how an aware object (e.g., the user) can be enhanced in an incremental fashion to accommodate new requirements and support PH behavior like the one described in the introduction.

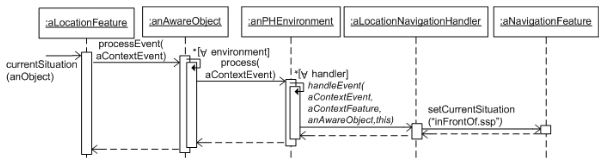
3.2 Supporting Location-Aware Behaviors

In the PH architecture we consider the user's location as the physical object he is standing in front of. In this way, locations are semantically rich and take an active role instead of being just data like in geometric models; transformation issues such as mapping data gathered by sensors into objects are outside of the scope of this paper, but can be read in [5]. In case the user is not in front of a physical object, his location is represented as a special object (*anywhere*) that acts as a null object [12].

The first relevant behavior in a PH application is to display information about the physical object the user is facing. To provide this kind of location-aware behavior, we define a handler (called *Location-Navigation*[1]) that is triggered each time the *Location* feature changes, and that as a response, updates the *Navigation* feature with the url corresponding to the object's information. As described before, since our modified Web browser depends on the *Navigation* context feature, when this feature changes (e.g. a new url is indicated) the browser will trigger an http request to this new url and will show the response in the mobile device. As a response of the http request, the user perceives a PH node similar to the one in Figure 2. This situation is depicted in an instance diagram in Figure 5, while Figure 6 shows the message flow started by a change in the location feature.


PH Environment
Location - Navigation Handler
Aware Object (User)
Location
Navigation
M3GATE
La Plata Cathedral
The most outstanding building and monument in the city.
Neo Gothic Style
It is located in La Plata geographical centre.
Natural Sciences Museum
Buy Tickets - Cathedral Elevator
<<Observer>>
Fig. 5. An aware object with location and navigation features.

[1] We use a naming convention for handlers with the form <ObservedFeature>-<FeatureToChange>


:aLocationFeature
:anAwareObject
:anPHEnvironment
:aLocationNavigationHandler
:aNavigationFeature
currentSituation (anObject)
processEvent(aContextEvent)
*[∀ environment] process(aContextEvent)
*[∀ handler] handleEvent(aContextEvent, aContextFeature, anAwareObject,this)
setCurrentSituation ("inFrontOf.ssp")
Fig. 6. An interaction diagram showing a location change.

3.3 Dealing with Physical Navigation

The main difference between digital and physical links is the kind of response the user receives when acting on them. When the user clicks on a digital link he navigates to another digital node just like in standard hypermedia. On the other hand, when the user selects a physical link, a map showing the path to the target object will appear. As the user walks, he incrementally completes his navigation, which is considered finished when he arrives to the destination.

To provide navigation assistance, we need to take into an account the activity the user is performing, in terms of the PH application. For this reason we added a new context feature called *Activity*. The activities we are actually considering are *standing in front of* a point of interest, *walking in* or *outside the path*. It should be noticed that the framework is open-ended regarding activities, but for the sake of comprehension we only discuss the previously mentioned one. At this point, the user aware object will be configured with the *Location*, *Navigation* and *Activity* features.

The *Activity* feature will be updated every time the *Location* feature changes. Activities are modeled as first class objects (as a variant of the State pattern [3]), and the new activity is determined by analyzing the new location according to the intended user's path. To provide this behavior we define a handler (called *Location-Activity*) that is triggered each time the *Location* feature changes and that re-evaluates the activity according to the new location.

The activity must also be re-evaluated when the user clicks on a physical link, since this determines his intention to perform a physical navigation. In this case the user's activity is set to *walking in the path*, and the activity itself records the itinerary from his current position to his destination. In order to implement this behavior we define a new handler called *Navigation-Activity*, which is triggered each time the *Navigation* feature is updated with a physical link. As a result, the new activity is set and a map is showed in the browser.

By introducing activities, we can improve the system's response by returning different urls according to the current user activity. To accomplish this, the url of the *Navigation* feature is set: *inFrontOf.ssp*[2] (if the user is in front of an object), or *inFrontOfTarget.ssp* (if the user has reached his target), or *passedBy.ssp* (if the user has passed by a point of interest).

At this point the user aware object is configured with three features (*Location*, *Navigation* and *Activity)* and three handlers (*Location-Navigation, Location-Activity* and *Navigation-Activity*).

[2] *ssp* stands for Smalltalk Server Pages. Smalltalk counterpart of JSP

3.4 Providing Smart Navigation Assistance

We aim to provide navigation assistance to the user during his travel. To achieve this, physical objects in the user itinerary should play different roles. Depending on the role a physical object is playing, the information and services shown to the user will vary because roles change the current physical object behavior. We implemented roles by wrapping physical objects with a role object (as a Decorator [3]).

We have already identified four major roles to be used in PH. When the user is traversing a link, and he is in the path, physical objects in his itinerary play the role of *information guides,* indicating him that he is not lost, and providing extra information. Meanwhile, if the user gets lost and passes by a physical object, it must take the role of a *helper guide,* warning the user that he is out of the scheduled path, and providing him with services to return to his previous trail or to recalculate a new path. The other two roles we implemented are: the *query guide* role which is used to search points of interest, and the *alert guide* which notifies the user about an urgent event (e.g. a fire taking place near his location). These roles don't depend on the user activity.

To provide different assistance behaviors according to roles, a new handler named *Location-Role* has been defined. This handler is triggered each time the user location changes. When the location changes, the *Location-Role* handler passes the control to an instance of the *RoleBuilder* class (which acts as a Facade [3]), to decide the role that must be assigned to the physical object the user is in front of. Since the activity is modeled as an object, the *RoleBuilder* delegates this decision by performing a double-dispatch between the user and his current activity. As a result the intended role is created and used to wrap the user's location.

4. Related Work

The term PH has been coined in [6]. In [8] the authors present an object-oriented framework (HyCon) whose goal is to extend the Hypermedia paradigm with the manipulation of real world objects. This framework has been used to create context-aware hypermedia systems and supports the classical mechanisms of

Hypermedia. The metaphor of “walking” a link has been presented in [9], where the authors augment Hypertext with physical relationships present in the real world. Finally, in [13] the authors define the set of roles that physical objects may play and take those roles to the Web to support the navigation of handicapped users. Our research has been certainly inspired by these seminal research projects; we further characterize real-world objects according to the role they can play to assist the user’s travel.

While we agree with the motivation and requirements presented in [7] (transparent monitoring of context, decoupled communication, scalability etc), our architecture goes a step further that context sensing abstraction, and not only provides a context model but also a set of abstractions to configure the context-dependent behaviour. From an architectural point of view, our work has been inspired in [1]: the sum of our micro-architectural decisions (such as using decorators or dependencies) also generates a flexible architecture. Our approach emphasizes a clear separation of concerns, decoupling the context model, the sensing mechanisms and the context-dependent behavior in different layers. Thanks to this decoupling changes in these three main concerns don’t impact on the others.

5. Concluding Remarks and Future Work

In this paper we have presented a scalable architecture to build context-aware physical hypermedia applications. The main flexibility in our design is given by the notion of context features and by configuring the system response to context changes by means of context event handlers. We consider that the most important strength of our approach is that it supports incremental development and can be easily modified to accommodate unforeseen requirements.

The prototype to test our architecture has been implemented in VisualWorks Smalltalk, which runs on different platforms an operating systems. We are currently researching in how other hypermedia metaphors can be applied to a PH application (e.g. the semantics of the *back* button) in order to provide a behavior that is consistent with the user’s intuition. We are also planning to enhance the role assignment logic to accommodate other context features, such as time or user preferences.

6. References

[1] K. Beck, and R. E. Johnson, “Patterns Generate Architectures”, *In Proceedings of the ECOOP'94*, Springer-Verlag Berlin, 1994, pp. 139-149.

[2] C. Challiol, G. Rossi, S.E. Gordillo, and V. De Cristófolo, “Designing and Implementing Physical Hypermedia” Applications, *In Proceedings of the ICCSA (4),* Springer-Verlag Berlin Heidelberg, 2006, pp. 148-157.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Software*, Addison-Wesley, 1994.

[4] S. Gordillo, G. Rossi, and D. Schwabe, “Separation of Structural Concerns in Physical Hypermedia Models”, *In Proceedings of the CAiSE 2005*, Springer-Verlag Berlin Heidelberg, 2005, pp. 446-459.

[5] J. Grigera, A. Fortier, G. Rossi, and S. Gordillo, “A Modular Architecture for Context Sensing”, *To be presented in PCAC-07*, Niagara Falls, Canada, May 21-23, 2007.

[6] K. Gronbaek, J. Kristensen, and M. Eriksen, “Physical Hypermedia: Organizing Collections of Mixed Physical and Digital Material”, *In Proceedings of the Hypertext 2003*, ACM Press, 2003, pp. 10-19.

[7] G. Hackmann, C. Julien, J. Payton, and G-C. Roman, “Supporting Generalized Context Interactions", *In Proceedings of the SEM 2004*, Springer-Verlag, 2005, pp. 91-106.

[8] F. Hansen, N. Bouvin, B. Christensen, K. Gronbaek, T. Pedersen, and J. Gagach, “Integrating the Web and the World: Contextual Trails on the Move”, *In Proceedings of the Hypertext 2004*, ACM Press, 2004, pp. 98-107.

[9] S. Harper, C. Goble, and S. Pettitt, “proximity: Walking the Link”, *Journal of Digital Information (JODI)*, British Computer Society and Oxford University Press, 2004, Volume 5.

[10] L. Romero, and N. Correia, “HyperReal: A Hypermedia model for Mixed Reality”, *In Proceedings of the Hypertext 2003*, ACM Press, 2003, pp. 2-9.

[11] G. Rossi, S. Gordillo, C. Challiol, and A. Fortier, “Context-Aware Services for Physical Hypermedia Applications”, *In Proceedings of the CAMS 2006*, Springer-Verlag Berlin Heidelberg, 2006, pp. 1914-1923.

[12] B. Woolf, “Null object”, *In Pattern languages of program design 3*, Addison-Wesley, 1997, pp. 5–18.

[13] Y. Yesilada, R. Stevens, and C. Goble, “A foundation for tool based mobility support for visually impaired web users”, *In Proceedings of the WWW '03*, ACM Press, 2003, pp. 422–430.