Journal of Mobile Multimedia, Vol.4, No.2 (2008) 118-138
© Rinton Press

ARCHITECTURAL AND IMPLEMENTATION ISSUES FOR A CONTEXT-AWARE HYPERMEDIA PLATFORM

CECILIA CHALLIOL

LIFIA. Facultad de Informática. Universidad Nacional de La Plata, Argentina.

Also at CONICET
ceciliac@lifia.info.unlp.edu.ar

ANDRÉS FORTIER

LIFIA. Facultad de Informática. Universidad Nacional de La Plata, Argentina.

DSIC. Universidad Politécnica de Valencia, Valencia, España.

Also at CONICET
andres@lifia.info.unlp.edu.ar

SILVIA GORDILLO

LIFIA. Facultad de Informática. Universidad Nacional de La Plata, Argentina.

Also at CICPBA
gordillo@lifia.info.unlp.edu.ar

GUSTAVO ROSSI

LIFIA. Facultad de Informática. Universidad Nacional de La Plata, Argentina.

Also at CONICET
gustavo@lifia.info.unlp.edu.ar

Received January 9, 2008
Revised April 23, 2008

In this paper we present the rationale and the main components of a modular and extensible architecture for building and deploying mobile hypermedia software. Using some simple archetypical examples we show how to provide context-aware assistance to the mobile user, as he explores the physical world. We also show that this kind of software systems poses strong requirements on supporting software (such as Web browsers) and we explain how to provide a modular software substrate to support these requirements. This paper shows how to use some simple concepts to develop complex context-aware systems, which are evolvable and easy to extend.

Key words: Mobile Hypermedia, Context-aware application, Mobile Computing

1 Introduction

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 [10], to provide assistance to the traveler [11], to implement augmented reality applications [16] or to provide context-aware software [9]. 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 (digital) way, or pointing to other physical objects. The user can therefore navigate virtually (as in the Web) or physically by “walking” the corresponding link [11]. One of the main differences between traditional hypermedia and PH applications is related to the atomicity of the actions. In standard hypermedia, clicking on a link implies opening the target page, or, in other words, that the user receives information immediately about the target node. On the other hand, clicking on a physical link only shows the *intention* of the user to navigate from one physical object to another and thus we must assign a different response to that action. In this case the system shows the user a map with the travel from his current position to the target object. Since obviously the system can’t physically move the target object, its aim in this (physical) navigation is to assist the user to reach his selected target. As the user walks the suggested path, he will come across other physical objects that will give him additional information in order to help him to complete his physical navigation. Once the user gets to the right path which will lead him to the target object, the physical navigation is completed and the browser information is updated, showing both physical and digital information about the actual object.

We have been working on different aspects of PH development such as design issues [5], server-side support [3] and assistance services [17]. 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). In this context, our contribution is twofold:

- In the field of PH, we show how to provide dynamic assistance to the PH user.
- From a more general perspective, we present a set of software abstractions which are useful in the field of context-aware applications.

The rest of the paper is organized as follows: In Section 2, we present a motivation example. In Section 3 we survey the requirements posed by PH software; in Section 4 we describe our architecture. In Section 5 we describe some details about our implementation and an evaluation. Section 6 we discuss some related work and in Section 7 we conclude and present some further work.

2 Motivation

As a concrete case study we will focus on a tourist visiting a foreign city, in particular the city of La Plata, Argentina. When the user stands in front of a place that is a node in the PH (e.g. a monument, a church, etc.), he receives information about it together with digital and physical links; for example, if the user stands in front of Dardo Rocha Museum he will receive information as shown in Figure 1. In this stage the user can decide between navigating in the digital or physical world. Navigating the digital world is just as navigating in standard hypermedia. However, navigating the physical world involves the user moving around, physically interacting with different objects which our application must be aware of. As an example, if the user chooses the physical link La Plata Cathedral, the system will show him a map that displays the path he should follow from where he is standing in order to get to the Cathedral.

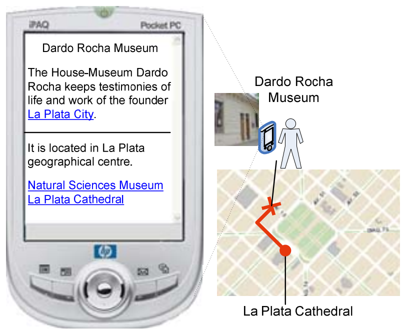

iPAQ
Pocket PC
Dardo Rocha Museum
The House-Museum Dardo Rocha keeps testimonies of life and work of the founder La Plata City.
It is located in La Plata geographical centre.
Natural Sciences Museum
La Plata Cathedral
Dardo Rocha Museum
La Plata Cathedral

Figure 1. The user standing in front of Dardo Rocha Museum

Suppose now that once the user has completed his visit to the museum, he walks to reach the Cathedral and makes a wrong turn, heading in a different direction. In his erroneous path he passes in front of the City Hall which, reacting to this situation, plays the role of helper guide. As a helper guide, the City Hall warns the user that he is out of his intended route and gives him a set of options according to the situation (see Figure 2).

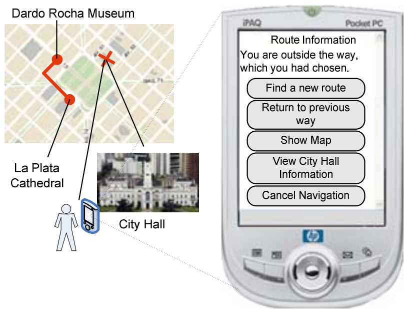


Figure 2. City Hall plays the role of helper guide.

In order to keep the example, let us suppose that the user decides that he still wants to visit the Cathedral and thus asks the system to find a new route. As a result, he receives a new path from his current position (the City Hall) to La Plata Cathedral (his selected target). As the user walks toward his intended target, he will pass by a Statue (placed in the center of Moreno Square) which is also a node in our PH network. Since the user is in the right direction to reach his target, the Statue will play the role of an information guide and give him a description about his route and a set of services the user may need (see Figure 3).

Once the user eventually gets to La Plata Cathedral he will not only receive the standard PH information about that node, but also a sign that he has reached his target. In this moment the physical navigation is considered to be over.

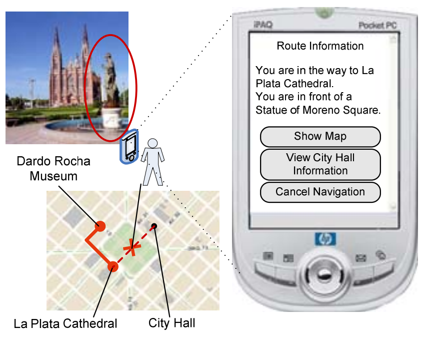

Route Information
You are in the way to La Plata Cathedral.
You are in front of a Statue of Moreno Square.
Show Map
View City Hall Information
Cancel Navigation
Dardo Rocha Museum
La Plata Cathedral
City Hall
iPAQ
Pocket PC

Figure 3. Statue plays the role of information guide.

In this simple example, as the user walked, he has passed by different physical objects, such as the City Hall, a Statue and the Cathedral. Depending on the user's activity and his situation (e.g. whether he is lost or not) those physical objects had played different roles. As a final example, let us suppose that the user has not selected any physical links and he is just walking around the Moreno Square. If he stands in front of the Statue, it will take the role of a decision guide [footnote 1], giving him information about the nearest physical objects to visit as shown in Figure 4.

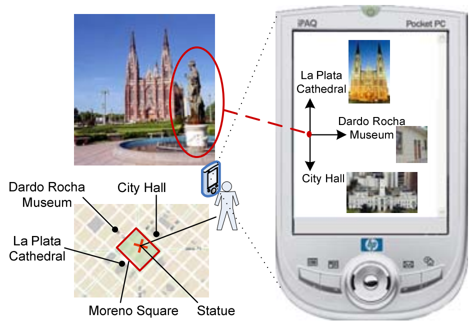


Figure 4. Statue plays the role of decision guide.

3 Requirements for a PH Architecture

In this section we summarize the driving forces which have shaped our architecture. Our main aim is 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 will explain the most important requirements separately.

3.1 Context-Aware Browsing

In Figure 5 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

[1] This role is currently under development.

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: to guide the user to reach the link's target. This involves defining the semantic of the link itself and which assistance services should be provided. Furthermore, the system needs to analyze which is the best role (or roles) to assign to the physical object depending on the user needs.

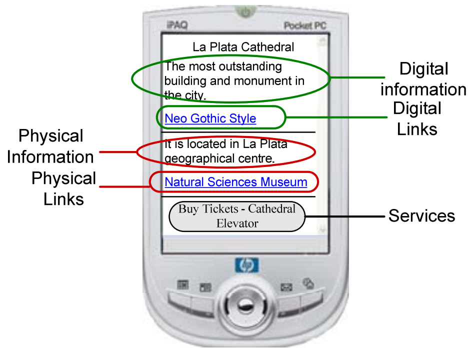


Figure 5. Interface of a PH application using a web browser.

Another feature of this application is that when the user remains in the same physical location, he can navigate digitally and the physical information won't change, effectively reflecting the fact that he is still standing in the same physical node. Finally, both the information and links (digital or physical) can change according to the user's context. As an example, let us consider a user in a room inside a museum, where a set of objects are displayed. If the room is, at the physical level, the smaller-grained object, all the other objects will display the same information about their locations (i.e., the room that contains them). However, as the user stands in front of each object, the digital content displayed in the browser will change, showing information about that particular object. Thus, physical and digital information can be displayed in the same browser at different levels of granularity.

3.2 Software Engineering Requirements

PH applications are a particular kind of ubiquitous systems and as a consequence their evolution pattern differs from traditional software. While we still have to care about changes or new requirements, ubiquitous applications evolve "organically" as wisely posed by Abowd [1]; applications are built many times as prototypes which grow together with both users' needs and new technological possibilities. Using our motivating example we might discover that by adding new kind of sensors, the city can incorporate new on-line services, like querying which roads are closed to improve path computing. This kind of evolution implies a challenge to the supporting architecture at several levels of abstraction: at the lowest levels we need to be able to incorporate new kind of sensing devices; as an immediate consequence, we might need to improve or extend the context model (if sensing devices provide a new kind of context information). Using this new kind of data we need to add, at the highest level of abstraction, new services and context-aware behaviors. Notice that, while this latter kind of modification is equivalent to adding new requirements (like in "conventional" software), the former ones might further stress the supporting architecture and the resulting design models.

We next describe the most important design choices we took to support the previously described behaviors and to make evolution seamless at all levels of abstraction.

4 The architecture in a nutshell

A key issue when devising context-aware applications is how to model the context in a way that it can support the application evolution and adding new features without having to redesign the whole application. In our architecture we decided to decompose the context in a set of *features*, each one representing a specific aspect of the context of a given application object. When a "standard" object is enhanced with one or more context features it becomes an *aware object* (i.e. an object that is aware of its context). Also, 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 [6] 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. Finally, since 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 use the concept of an *environment* to model that specific adaptation. For example, we could device an environment for context-aware groupware services and other for location based services, both of them using the location feature of the user to provide a different kind of adaptation. As we will show in section 4.1, an environment has a set of handlers that are in charge of reacting upon context changes to implement its specific context-aware behavior. This basic decomposition is shown in Figure 6.

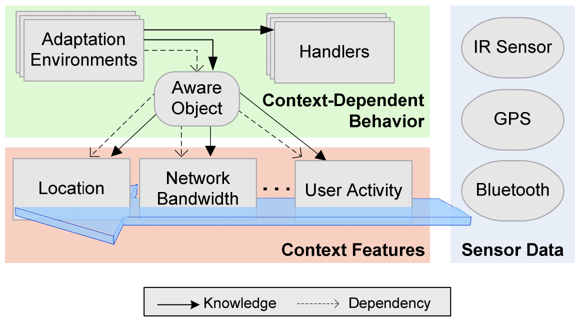


Figure 6. An architecture for handling context-dependent behavior.

To complete the description of our approach to PH applications we must describe the way we use the web browser, since it is a key component of our architecture. In "conventional" web applications, when a link is clicked an HTTP request is issued and the response is shown to the user. Even though a communication failure may arise or a web server may be down, the user considers web navigation as an atomic action; a link is clicked and the requested information is presented. In PH applications this behavior must be slightly modified in two ways: 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 [13] 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. because the user is standing in front of a different monument), 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 4.2 and 4.3. 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.

4.1 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 [4] 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. For example, if we wanted to model a simple smart-home application that turns on the heating according to the current temperature and time of the day we would model the specific room as an aware object with two context features: the room's temperature and the current time. In our architecture, these 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, the current date and the time of the day). However, since these features are modeled as independent software components, any required redesign should not affect the other. In our mobile tourist guide application, we define the user as an aware object that knows context features such as location, visited points of interest and preferences.

As we have previously mentioned, the combination of aware objects and context features are used to model the notion of context. However, since the same context information can be used to provide different kinds of adaptation, we consider them as adaptation-agnostic and we put this responsibility on the environments. Thus, when an aware object is interested in having a specific adaptation (such as PH or Location-Based Services), it has to be registered to the desired environment, which in turn becomes an Observer [4] of that 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.

Even though a first approach would be for modeling the complete behavior in the environment itself we found out that many times we would like to enhance a given environment with behavior that is independent from the facilities already provided by that environment. At the same time, as the application evolves we may encounter new services to provide to the user. In order to cope with this evolution we decided to encapsulate this behavior in small-grained objects (the handlers) that implement the specific behavior. These handlers are attached to the environment and are configured to listen to specific context changes. Thus, handlers are responsible for 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), a context event is triggered to notify all interested parties. The aware object that observes the feature will receive this notification and will, in turn, notify all interested environments about 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 them. 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. Let us 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 available services. Finally, we could also be willing 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. As a result, by encapsulating this behavior in small grained objects, the application can be modified with small or null impact. In Figure 7 we present a simplified class diagram of these aspects of our architecture.

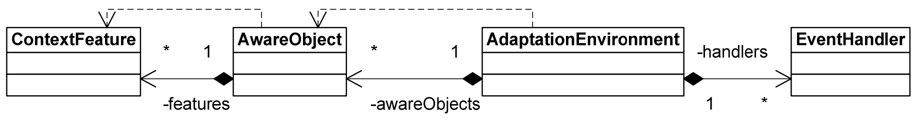


Figure 7. A simplified view of the architecture's core classes.

In Figure 8 we show an interaction diagram depicting the sequence of messages sends that is triggered when a context feature changes. Each time a context feature receives the setCurrentSituation() message, it triggers a change that is captured by the corresponding aware object. As a response, a context event is created and passed to all the adaptation environments the user is registered to by means of the process() message. The environment evaluates all its event handlers to see if they should be triggered (canHandle() message) and if so, it will forward the corresponding notification (handleEvent()).

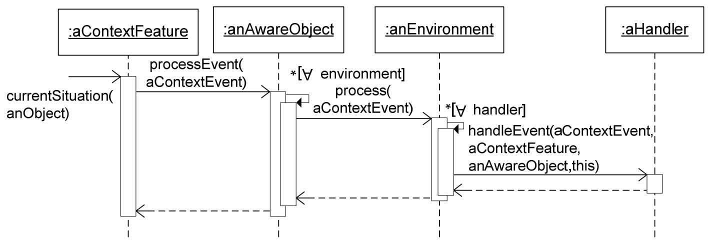

:aContextFeature
:anAwareObject
:anEnvironment
:aHandler
currentSituation(anObject)
processEvent(aContextEvent)
*[∀ environment] process(aContextEvent)
*[∀ handler]
handleEvent(aContextEvent, aContextFeature, anAwareObject,this)

Figure 8. An interaction diagram showing the event propagation.

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

4.2 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 will be briefly covered in section 5 and can be read in more detail in [6]. 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 [18].

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 [footnote 2]) 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. 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 5. This situation is depicted in an instance diagram in Figure 9, while Figure 10 shows the message flow started by a change in the location feature.

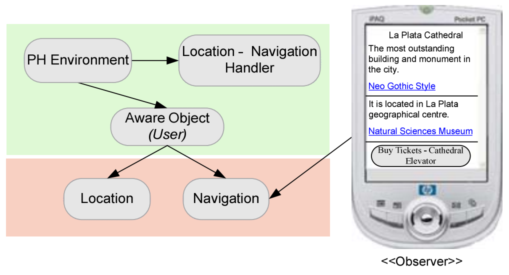

PH Environment
Location - Navigation Handler
Aware Object (User)
Location
Navigation
La Plata Cathedral
The most outstanding building and monument in the city.
Neo Gothic Style
It is located in La Plata geographical centre.
Natural Sciences Museum
Buy Tickets - Cathedral Elevator
<<Observer>>

Figure 9. An aware object with location and navigation features.

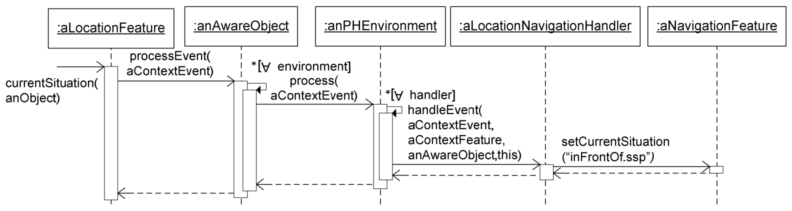

:aLocationFeature
:anAwareObject
:anPHEnvironment
:aLocationNavigationHandler
:aNavigationFeature
currentSituation(anObject)
processEvent(aContextEvent)
*[∀ environment] process(aContextEvent)
*[∀ handler] handleEvent(aContextEvent, aContextFeature, anAwareObject,this)
setCurrentSituation ("inFrontOf.ssp")

Figure 10. An interaction diagram showing a location change.

4.3 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

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

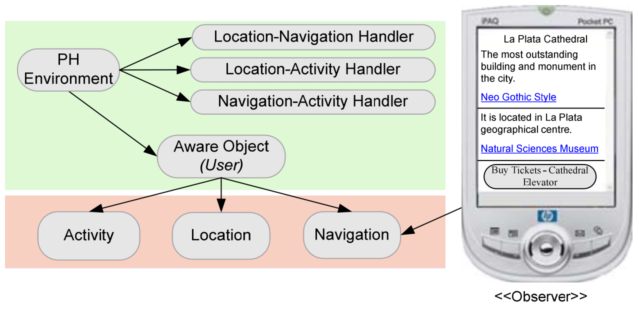
standard hypermedia. On the other hand, when the user selects a physical link, a map showing the path to the target object will appear. As the user walks, he incrementally completes his navigation, which is considered finished when he arrives to the destination.

To provide navigation assistance, we need to take into an account the activity the user is performing, in terms of the PH application. To do so a new context feature called Activity is added to our existing context model. The activities we are actually considering are standing in front of a point of interest and walking inside or outside the planned 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.

Activities are modeled as first class objects (as a variant of the State pattern [4]), and the new activity is determined by analyzing the new location according to the user's intended path. Also, the Activity feature will be updated every time the Location feature changes. To provide this behavior we define a handler (called Location-Activity) that is triggered every time the Location feature changes and that re-evaluates the activity according to the new location. Finally, activities 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 a 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 URL's according to the current user situation. To accomplish this, the URL of the Navigation feature is set to inFrontOf.ssp [footnote 3] (if the user is in front of an object), inFrontOfTarget.ssp (if the user has reached his target) or passedBy.ssp (if the user has passed by a point of interest when he is in the chosen path).

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), as shown in Figure 11.


PH Environment
Location-Navigation Handler
Location-Activity Handler
Navigation-Activity Handler
Aware Object (User)
Activity
Location
Navigation
iPAQ
Pocket PC
Buy Tickets - Cathedral Elevator
<<Observer>>

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

Figure 11. An aware object with three features and three handlers.

[3] SSP stands for Smalltalk Server Pages, Smalltalk counterpart of JSP.

4.4 Providing Smart Navigation Assistance

Since one of our aims is to provide navigation assistance to the user during his trip, we will assign different roles to the physical objects the user may pass by. Depending on the role the relevant physical object is playing, the information and services shown to the user will vary because the roles change the behavior of the physical object. We implemented roles by wrapping physical objects with a role object (as a Decorator [4]).

We have already identified four major roles to be used in PH. When the user is traversing a link, and he is in the right path, physical objects in his itinerary play the role of information guides, indicating him that he is not lost and providing extra information. Conversely, 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 showing him services to return to his previous trail or to recalculate a new path. The other two roles we implemented are the query guide, 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). Notice that these last two 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 every time the user's 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 [4]), 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. This situation is depicted in an instance diagram in Figure 12.

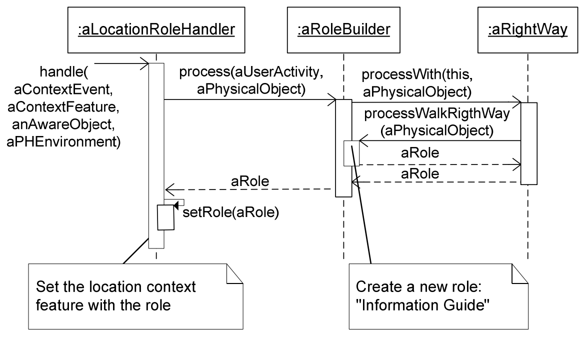

:aLocationRoleHandler
:aRoleBuilder
:aRightWay
handle(aContextEvent, aContextFeature, anAwareObject, aPHEnvironment)
process(aUserActivity, aPhysicalObject)
processWith(this, aPhysicalObject)
processWalkRigthWay (aPhysicalObject)
aRole
aRole
aRole
setRole(aRole)
Set the location context feature with the role
Create a new role: "Information Guide"

Figure 12. Determining the role to be played by a physical object.

4.5 Integrating all concepts

In the previous sections we mentioned a set of elements (as context features, aware objects, environments and handlers) needed to represent a PH application with our architecture and the interaction between them. In Section 4.2, 4.3, 4.4 we show how to incorporate new elements to represent the main concepts underlying a PH system. In Figure 13 we show a diagram that integrates all these concepts and how handlers are triggered by specific features. As can be seen in the diagram, when the location feature changes three handlers are triggered, causing the Activity, Navigation and Location features to be updated. Also, when the Navigation feature changes, the Activity feature is updated.

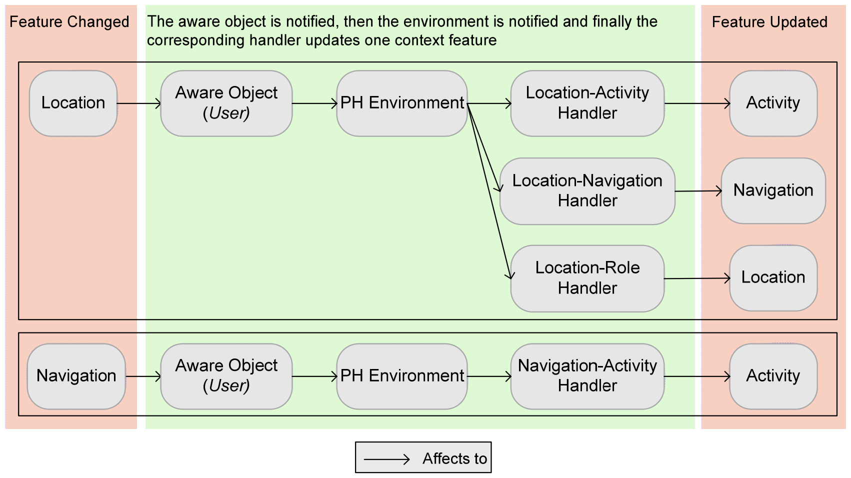

Feature Changed
The aware object is notified, then the environment is notified and finally the corresponding handler updates one context feature
Feature Updated
Location
Aware Object (User)
PH Environment
Location-Activity Handler
Activity
Location-Navigation Handler
Navigation
Location-Role Handler
Location
Navigation
Aware Object (User)
PH Environment
Navigation-Activity Handler
Activity
Affects to

Figure 13. A complete diagram with the relations when contexts feature changes.

5. Implementation and Evaluation

In this section we will show some details about the implementation of our prototype. Since we don't have the required infrastructure in the city to perform real tests, we devised a simulator to do indoor tests. In this simulator we use the city's cartography as the basis to situate a set of beacons, eventually each one having a different range. When the user is standing within the range of a certain beacon, we send the ID to a simulated hardware, which is in turn captured by a sensing concern and injected into the context model.

5.1 Implementation details

Our generic architecture and the prototype application to test the proposed PH concepts have been implemented in VisualWorks Smalltalk 7.4.1. VisualWorks has some interesting characteristics which make it suitable for the development of this kind of applications: first of all it is supported in all major operating systems and can be used in PDAs running windows mobile. Also since Smalltalk is dynamically typed and supports full reflective capabilities we can implement our concepts in a clearer way and add behavior in a dynamic fashion.

5.1.1 Basic setup

As a first example we will consider a user with a single feature that is used to track his location. Since a context feature models a piece of context (in this case, the user's context) it must always have a *current situation*. To simplify things we will assume that we are using a symbolic location model [14]. A first approach is shown in Code 1:

```
| user |
user:=AwareObject new.
user addFeature:
          (ContextFeature
                       named: 'Location'
                       currentSituation: 'City Hall').
```

Code. 1. Creation of an aware object and a single feature.

Since the aware object has been “extended” with a location feature it now supports, via reflection, two new messages:

- `locationFeature`, which answers the ‘Location’ context feature. This would be the same as evaluating `user featureNamed: ‘Location’`
- `currentLocation`, which returns the current situation of the ‘Location’ feature. This would be the same as evaluating `(user featureNamed: ‘Location’) currentSituation.`

Now that the user has been created with a location feature, we need to find a way to capture the beacon received by the IR port of the PDA and transform it into context information (see Code 2):

```
| ir pushPolicy beaconLookup |
ir:=IRPort new.
pushPolicy:=PushPolicy new.
beaconLookup:=LookupTransformation on:‘IR2Location.xml’.
sa:=SensingConcern
          policy: pushPolicy
          transformation: beaconLookup
          receiver: user locationFeature.
```

Code. 2. Creating a sensing concern to map IR beacons to locations.

In this example we are creating a sensing concern that will use a push policy to gather data from the IR port of the PDA. To do so, we have developed an IRPort class that abstracts the hardware and presents a simple accessing protocol. In order to make the mappings between the beacon’s ID’s and the location they represent, we use the class LookupTransformation which can read an XML file of key-value pairs in order to apply them to incoming sensor values. Finally, we specify which feature will be the receiver of the incoming data (i.e. the location feature).

The final step is to create the adaptation environment with its required handlers and register the aware object to it. The first step is accomplished by subclassing the AdaptationEnvironment class and defining the collaboration and responsibilities needed for this domain. In the case of PH applications we will need the graphs of physical and digital objects that the user will visit and to define a set of messages to query the environment about those objects.

The next step is to define a set of handlers that perform the required adaptation. To keep things simple we will show how the LocationNavigation handler is implemented (i.e. the handler that is in charge of changing the browser’s URL according to the place where the user is standing). To do so, we must subclass the Handler class and define the handle:from:on:environment: message. This message will be triggered every time the location feature changes (see Code 3):

```
LocationNavigation>>handle: aContextEvent
                 from: aContextFeature
                 on: anAwareObject
                 environment: aPHEnvironment

    anAwareObject currentNavigation:
                 anAwareObject currentState activityURL.
```

Code. 3. Defining a Location-Navigation handler.

The activityURL is a message implemented in the UserState class as explained in section 4.3, returning the URL that corresponds to the user’s activity. The final step is to add the handler as part of the environment and to register the handler to the PH environment (we assume that the graphs of physical and digital objects are already created), as shown in Code 4:

```
| env pushPolicy beaconLookup |
env:=PHEnvironment
          digitalNodes: digitalNodesCollection
          physicalNodes: physicalNodesCollection.
user addEnvironment: env.
env addEventHandler:
          (LocationNavigation
                       forFeature: user locationFeature).
```

Code. 4. Instantiating an environment and adding the navigation handler.

5.1.2 The Web Browser

As we explained in section 4 we adapted the web the behavior of the browser to suite our needs. To do so, we use WithStyle, which is a Smalltalk view that renders XML and CSS and that can be used as an HTML widget inside a VisualWorks application. To dynamically generate the page contents we use Smalltalk Server Pages (SSP) which allows communicate with Smalltalk classes. The SSP pages can be coded using JSP or ASP semantics but with Smalltalk code and also supports Custom Tags to display information. When the Navigation feature changes, a registered handler is triggered and an HTTP request is sent to the server (by means of the Net.HttpClient class) with the new URL. As previously explained, according to the user’s state a different URL will be sent, identifying how a given object should be presented (*inFrontOf.ssp*, *inFrontOfTarget.ssp*, *passedBy.ssp*, etc).

Regarding the server’s side, since we are in a prototyping stage, we use a built-in Smalltalk HTTP Server that can serve common request incoming from our web browser. When a HTTP requests asking for a SSP is received the server just finds the page, evaluate any Smalltalk code and returns the resulting page. Thanks to this built-in server and the fact that pages are located in relative directories we can package the full server application in a single environment and test it in different machines without having to reconfigure it or to depend on other software infrastructure.

5.1.3 Advanced Topics

In section 5.1.1 we presented simple example to start a basic PH application. In this section we will cover some issues that appeared while developing our PH prototype. The first issue has to do with different requirements in context features; some features must be tracked as they change to have a history (e.g. the visited nodes), others need an underlying model to make sense (e.g. a symbolic location needs a symbolic map to be related to other locations) while others may be derived from existing features (e.g. the weather conditions tagged as *good* or *bad* can be derived from a set of simple features like temperature, humidity and pressure). Thus, we need a model that supports these scenarios and that allows combining them (e.g. a derived feature that can be tracked and that has an underlying model). To do so we use a hierarchy of context features (see Figure 14) that models an atomic feature as the most simple one, a set of wrappers to add models or tracking capabilities and the notion of a derived feature that is computed as an aggregation of one or more other features. This derived feature also collaborates with an updating policy which determines when a change in any of its subcomponents must be forwarded to its container.

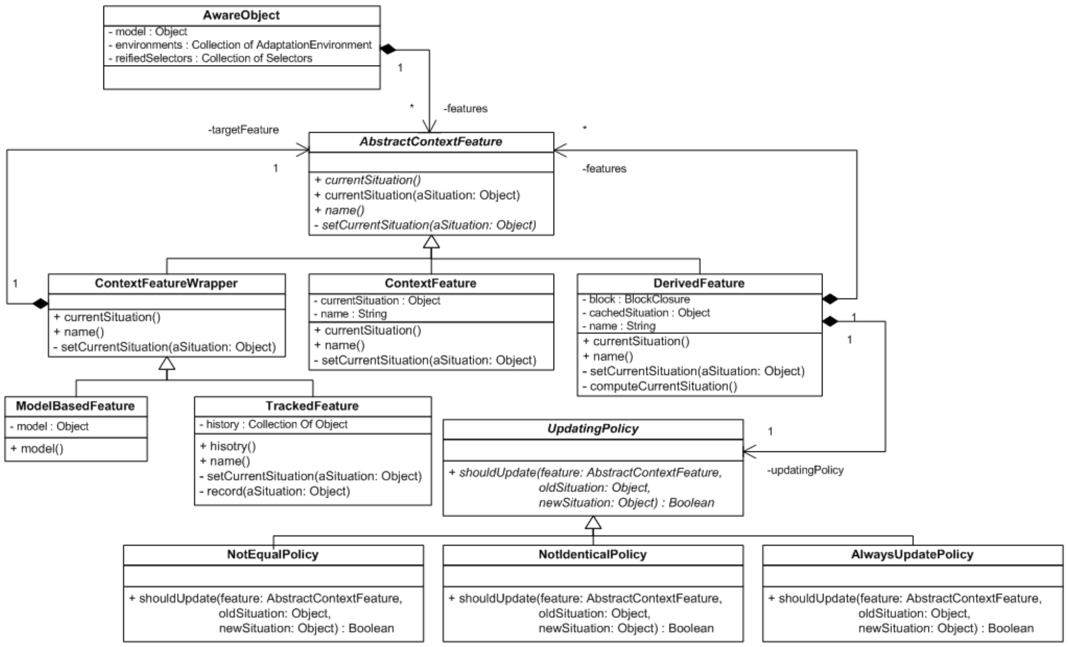

AwareObject
- model : Object
- environments : Collection of AdaptationEnvironment
- reifiedSelectors : Collection of Selectors
1
*
-features
-targetFeature
1
*
-features
AbstractContextFeature
+ currentSituation()
+ currentSituation(aSituation: Object)
+ name()
- setCurrentSituation(aSituation: Object)
1
ContextFeatureWrapper
+ currentSituation()
+ name()
- setCurrentSituation(aSituation: Object)
ContextFeature
- currentSituation : Object
- name : String
+ currentSituation()
+ name()
- setCurrentSituation(aSituation: Object)
DerivedFeature
- block : BlockClosure
- cachedSituation : Object
- name : String
+ currentSituation()
+ name()
- setCurrentSituation(aSituation: Object)
- computeCurrentSituation()
1
1
ModelBasedFeature
- model : Object
+ model()
TrackedFeature
- history : Collection Of Object
+ hisotry()
+ name()
- setCurrentSituation(aSituation: Object)
- record(aSituation: Object)
UpdatingPolicy
+ shouldUpdate(feature: AbstractContextFeature, oldSituation: Object, newSituation: Object) : Boolean
1
-updatingPolicy
NotEqualPolicy
+ shouldUpdate(feature: AbstractContextFeature, oldSituation: Object, newSituation: Object) : Boolean
NotIdenticalPolicy
+ shouldUpdate(feature: AbstractContextFeature, oldSituation: Object, newSituation: Object) : Boolean
AlwaysUpdatePolicy
+ shouldUpdate(feature: AbstractContextFeature, oldSituation: Object, newSituation: Object) : Boolean

Figure 14. The ContextFeature hierarchy.

Notice that, even though we have only modeled the most usual requirements for context features, since the delegation mechanisms are fully implemented in the ContextFeatureWrapper class, new variations can be added if necessary with almost no effort.

Another interesting issue is how handlers are triggered; in section 5.1.1 we created a handler that was triggered every time a specific feature changed. However, if this was the only option, the mechanism would be too restrictive since we couldn't specify a handler to be triggered when any feature of an aware object changes or to be dependent of changes of a given feature class. To solve this, an event handler implements the message canHandle: aContextEvent by delegating it to a hierarchy of policies (currently we have implemented the most common patterns). In case none of these satisfy the programmer requirements, a matching policy can be instantiated with an arbitrary block of code that returns true or false indicating whether the handle should be triggered by that event. In Figure 15 we show a class diagram of the MatchingPolicy hierarchy. Below, we show an example of a handler that is triggered only when a Navigation event arrives and its current link is digital (see Code 5).

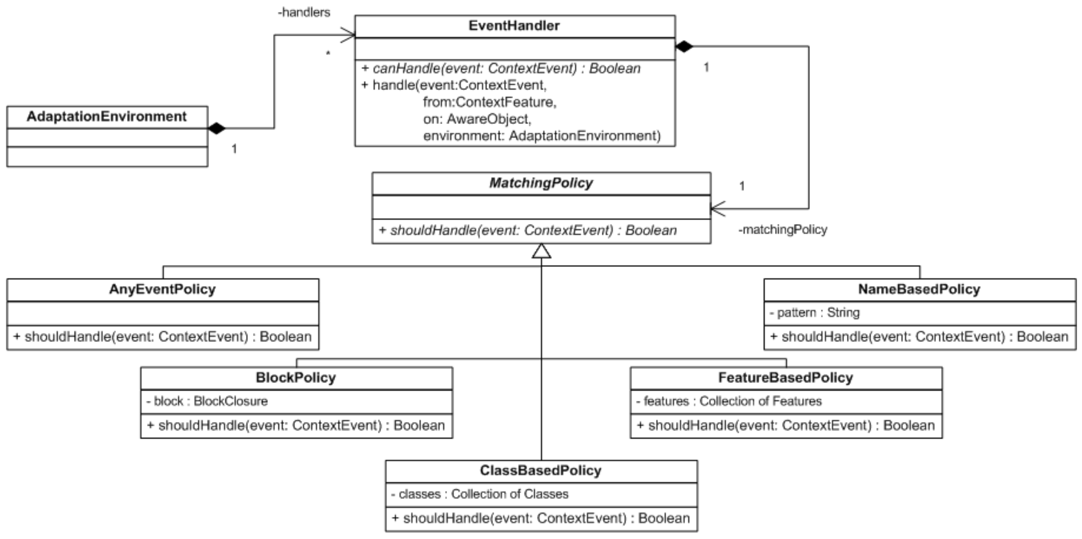

-handlers
EventHandler
+ canHandle(event: ContextEvent) : Boolean
+ handle(event:ContextEvent, from:ContextFeature, on: AwareObject, environment: AdaptationEnvironment)
AdaptationEnvironment
MatchingPolicy
+ shouldHandle(event: ContextEvent) : Boolean
-matchingPolicy
AnyEventPolicy
+ shouldHandle(event: ContextEvent) : Boolean
NameBasedPolicy
- pattern : String
+ shouldHandle(event: ContextEvent) : Boolean
BlockPolicy
- block : BlockClosure
+ shouldHandle(event: ContextEvent) : Boolean
FeatureBasedPolicy
- features : Collection of Features
+ shouldHandle(event: ContextEvent) : Boolean
ClassBasedPolicy
- classes : Collection of Classes
+ shouldHandle(event: ContextEvent) : Boolean

Figure 15. The MatchingPolicy hierarchy.

```
| policy |
policy:=BlockPolicy on:[:anEvent |
                          (anEvent name = 'Navigation')
                          and:['*digitalLink*'
                                   match: anEvent currentSituation]].

env addEventHandler:
       (DigitalNavigationHandler matchingPolicy: policy).
```

Code. 5. Creation of a block policy to match digital links when the Navigation feature changes.

Finally, we may encounter situations where we need to use multiple devices to gather context information. In our system we use two kinds of beacons to detect the user's location, IR and Bluetooth. Which one is used in a particular situation depends on the coverage area and whether we want the user to explicitly point to the target or not. As en example consider a user inside a museum: if we want to tag specific items in a gallery we would use an IR beacon for each, so that if the user wants to get information about the object he must point his PDA to it. On the other hand, if we need to know on which room the user is, Bluetooth beacons would be more appropriate, since they have a larger coverage area and are non-directional. Since in our approach sensing is a cross cutting concern, using two devices to inject information into the same context feature is straight forward: we just have to repeat the process shown previously for the IR beacon, but this time with the Bluetooth port. However, if we use this configuration, the last received signal will "win" over the other, which is not necessarily what we expect. As an example, let us suppose we have a Bluetooth beacon to cover a whole room (e.g. RoomA) and an IR beacon inside that room that tags a specific painting (e.g. PaintOne). In this case we would expect the user's location to be RoomA if he is inside the room and not in front of the painting, and PaintOne if the user is standing in front of the painting. Thus, when receiving two beacon signals, we expect the location with lower granularity to win. If we don't impose this rule before the data arrives to the context feature, our application may get erroneous information. To solve this, we add and intermediate object between the sensing concern and the target object, that works as a composite sensing concern. This new sensing concern is an aggregation function performed between its children. To solve the two beacons issue, when a new signal arrives, the composite function checks which of them has a smaller granularity (by asking the symbolic map that was added to the context feature) and forward it as the current situation of the location feature.

5.2 Evaluation

In order to test PH applications in a real setting a minimum infrastructure is required: at least we must tag specific objects and areas with some identification mechanism (IR, Bluetooth, Code bars, etc) and have access to a wi-fi access point. We currently don't have the required infrastructure in the physical sense, so we built a simulator to manage the user as he moves in the city. This simulator was built based on the city's cartography and can be used in two modes: to move the user trough the streets (i.e. only allowing him to move in open spaces) or in free form (i.e. letting the user get inside a museum). Below, there is a screenshot of the simulated environment (see Figure 16).

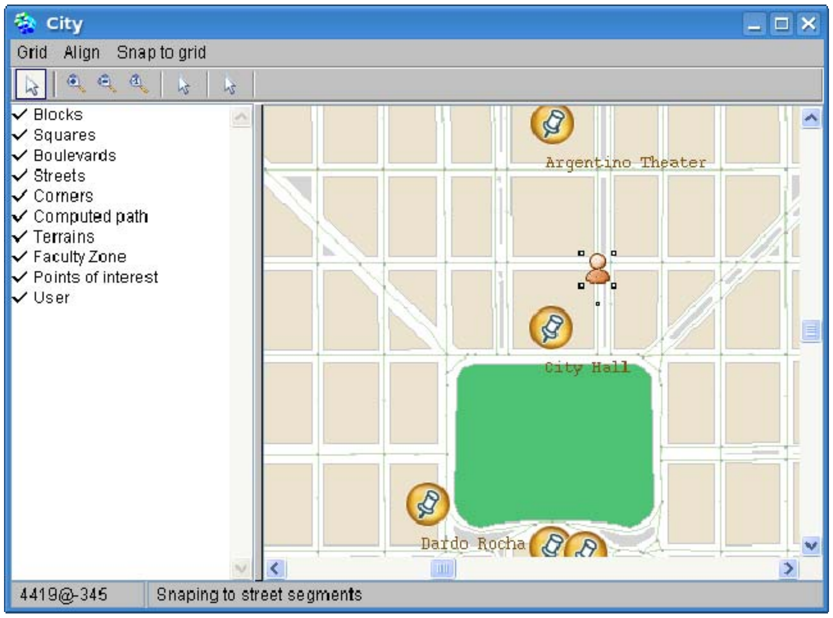


Figure 16. A screenshot of the simulator used to test our prototype

In this example, the user is approaching the City Hall. When he arrives, the system shows both digital and physical information about it. If he then decides to walk to the Government House a map is displayed showing the route he should follow (see Figure 17).

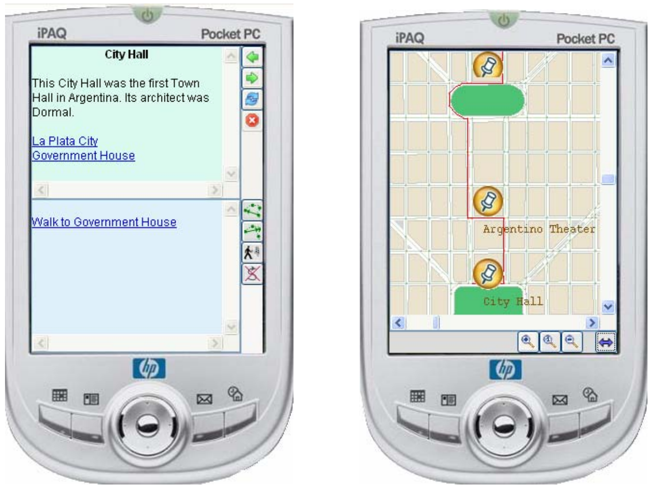


Figure 17. HTML and Map information presented to the user.

Finally we are working on providing better assistance to the user. We are using the simulator shown in Figure 16 to also give the user a 3D view of the city, from the user's perspective (see Figure 18). In this 3D scene we will merge P.H. (where physical objects will be emphasized) with augmented reality. We are also planning to incorporate photographs of the blocks and streets to provide a better experience.

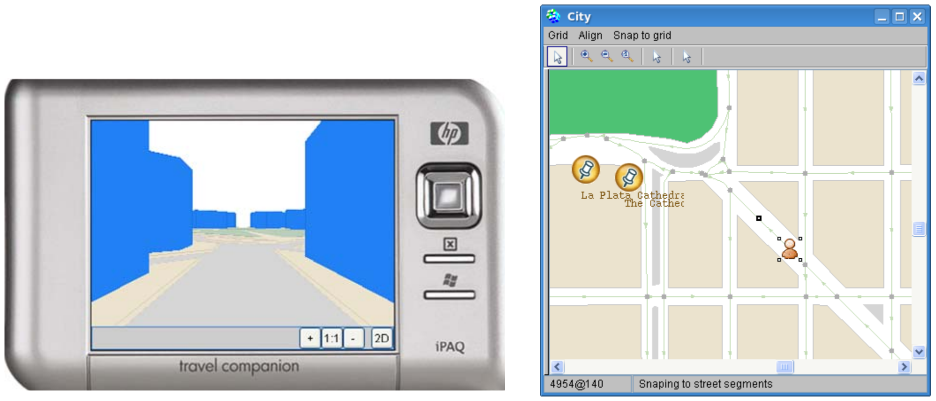


Figure 18. A screenshot of the simulator with 3D view

In our simulated environment the system performed as expected and so did the architecture. As we developed the PH application new ideas appeared and we were able to implement them by adding new handlers or refactoring existing behavior. However, as we will explain in section 7, there are still lots of areas to improve and more research to do.

6 Related Work

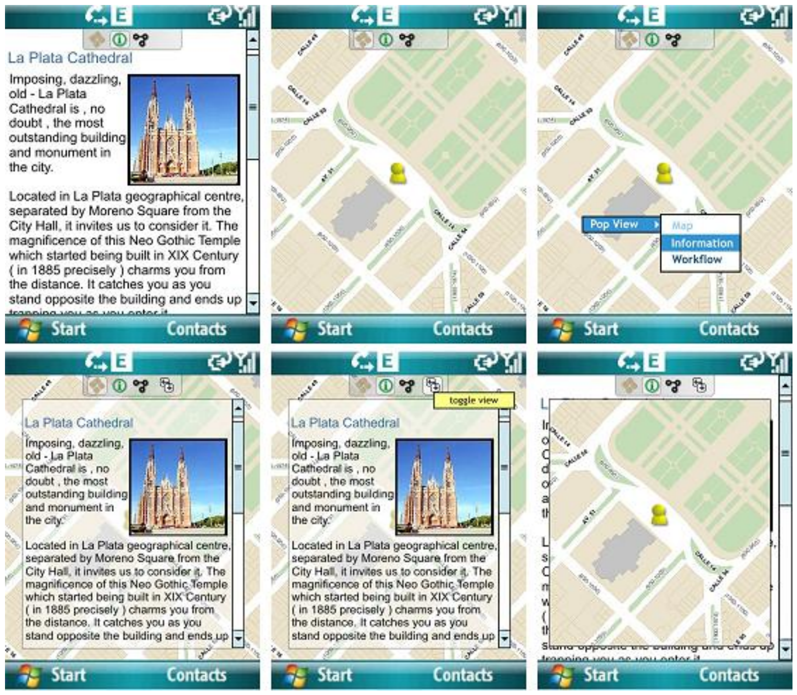
The term PH has been coined in [7]. In [10] the authors present an object-oriented framework (HyCon) whose goal is to extend the Hypermedia paradigm with the manipulation of real world objects. This framework has been used to create context-aware hypermedia systems and supports the classical mechanisms of Hypermedia. The author specialized this work in [9]. In [16] meanwhile, an object-oriented framework called HyperReal, allows building augmented reality applications. HyperReal uses concept from adaptive and spatial hypermedia to integrate, in the same setting, virtual documents, 3D environments and the physical world to build mixed reality applications. A Mobile Tourist information Systems (TIP) is presented in [12], where the authors mention different user's interactions, like "Browsing by walking" or going "To last location". Even though 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 metaphor of "walking" a link has been presented in [11]. The main goal of proXimity is to extend the metaphor of links to the real world and to show how the basic ideas behind adaptive hypermedia can be applied to physical hypermedia spaces. The idea of "walking" the link in proXimity inspired our physical navigation. In [19] 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 journey. Finally, in [15] the authors present an approach to identify pedestrian flows and to attach landmark information into navigation services for pedestrians. In our current realization of context-aware hypermedia applications we only consider to be landmarks those physical objects which the user faces in his detour. Improving the architecture to introduce landmarks which are far from the user's position is a challenge for further work.

While we agree with the motivation and requirements presented in [8] (transparent monitoring of context, decoupled communication, scalability etc), our architecture goes a step further than context sensing abstraction, and not only provides a context model, but also a set of abstractions to configure the context-dependent behavior. From an architectural point of view, our work has been inspired by [2]: 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. By decoupling these three concerns (context model, sensing and context-dependent behavior) changes are not propagated between them.

7 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 adapt to unforeseen requirements. As was mentioned in section 5.1, the architecture met our requirements in terms of scalability and flexibility. However, there are many areas where the prototype can be improved. The first thing we noticed was that by splitting the browser in two halves (i.e. digital and physical information) the available space to show information was very small. On top of that, if the user moves while he switches views (as shown in Figure 17) he may loose important information. For this reason we think that a different approach must be taken to show different aspects (e.g., physical and digital) of the same object. Our idea is that the user can visualize a main aspect of the object (e.g. its location on a map) and that he is able to put emergent information on top of that, without loosing his context. At any time the user can choose to close the emergent view or switch the views so that the emergent information is swapped with the one displayed in the background (a sketch of the GUI is presented in Figure 19).


La Plata Cathedral

Imposing, dazzling, old - La Plata Cathedral is , no doubt , the most outstanding building and monument in the city.

Located in La Plata geographical centre, separated by Moreno Square from the City Hall, it invites us to consider it. The magnificence of this Neo Gothic Temple which started being built in XIX Century (in 1885 precisely) charms you from the distance. It catches you as you stand opposite the building and ends up trapping you as you enter it.

Figure. 19. An interaction sketch of the UI we are developing.

We are currently researching in how other hypermedia metaphors can be applied to a PH application (e.g. the semantics of the back button) and how to define a navigation model that can be used to display a different type of information in a consistent way with the user’s intuition. So far we have detected four different type of information that can be useful: digital information (HTML), physical information (both HTML and as a map) and workflow information (e.g. the different places to visit in a pre-planned tour). All this models present an interaction that is compatible with the basic hypermedia commands, like back and next buttons.

At the assistance level we are planning to enhance the role assignment logic to incorporate other context features (such as time or user preferences) and providing more specific assistants (like the decision guide role). We are also planning to merge our PH application with other prototypes we have developed for augmented reality, providing a virtualization of the city. Our thesis is that if the user can see in his PDA a view that resembles his current view of the city, where the tagged physical objects are marked in a distinctive way, he will be able to easily distinguish them and ask for guidelines in case he is lost or needs a special service. Finally, we are building a small tool to automatically extract digital models and relationships with physical objects based on existing sites. In this way we would be able to automate most of the process of arranging the information of the PH application.

References

1. Abowd, G. D. (1999), “Software Engineering Issues for Ubiquitous Computing”, In Proceedings of the ICSE'99, IEEE Computer Society Press, Los Angeles, California, United States, pp. 75-84.
2. Beck, K. and Johnson, R. E. (1994), “Patterns Generate Architectures”, In Proceedings of the ECOOP'94, Springer-Verlag Berlin, Bologna, Italy, Vol. 821, pp. 139-149.
3. Challiol, C., Rossi, G., Gordillo, S.E. and De Cristófolo, V. (2006), “Systematic Development of Physical Hypermedia Applications”, Intl J of Web Info. Sys. (IJWIS), Vol.2, No.3/4, pp232-246.
4. Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1994), Design Patterns: Elements of Reusable Software, Addison-Wesley, Reading, Massachusetts.
5. Gordillo, S., Rossi, G. and Schwabe, D. (2005), “Separation of Structural Concerns in Physical Hypermedia Models”, In Proceedings of the CAiSE 2005, Springer-Verlag Berlin Heidelberg, Porto, Portugal, Vol. 3520, pp. 446-459.
6. Grigera, J., Fortier, A., Rossi, G. and Gordillo, S. (2007), “A Modular Architecture for Context Sensing”, In Proceedings of the PCAC-07, IEEE Computer Society Press, pp. 147-152 .
7. Gronbaek, K., Kristensen, J. and Eriksen, M. (2003), “Physical Hypermedia: Organizing Collections of Mixed Physical and Digital Material”, In Proceedings of the Hypertext 2003, ACM Press, Nottingham, UK, pp. 10-19.
8. Hackmann, G., Julien, C., Payton, J. and Roman, G-C. (2005), “Supporting Generalized Context Interactions", In Proc. of SEM 2004, Springer-Verlag Berlin Heidelberg, Vol. 3437, pp. 91-106.
9. Hansen, F.A. (2006), “Context-aware Mobile Hypermedia: Concepts, Framework, and Applications”, Ph.D. Dissertation, Department of Computer Science, University of Aarhus.
10. Hansen, F., Bouvin, N., Christensen, B., Gronbaek, K., Pedersen, T. and Gagach, J. (2004), “Integrating the Web and the World: Contextual Trails on the Move”, In Proceedings of the Hypertext 2004, ACM Press, Santa Cruz, California, USA, pp. 98-107.
11. Harper, S., Goble, C. and Pettitt, S. (2004), “proximity: Walking the Link”, Journal of Digital Information (JODI), British Computer Society and Oxford University Press, UK, Vol. 5, No 1.
12. Hinze, A., Malik, P., Malik, R. (2006), “Interaction design for a mobile context-aware system using discrete event modeling”. In Proceedings of the ACSC’06, Australian Computer Society, Hobart, Australia, pp. 257–266.
13. Krasner, G. and Pope S. (1988), “A Cookbook for Using Model-View-Controller User Interface Para-digm in Smalltalk-80”. Journal of Object Oriented Programming, SIGS Publications, Denville, NJ, USA, Vol. 1, No 3, pp. 26-49.
14. Leonhardt, U. (1998), “Supporting Location-Awareness in Open Distributed Systems”, Ph.D. Thesis, Department. of Computing, Imperial College, London.
15. Millonig A. and Schechtner K. (2005), “Developing landmark-based pedestrian navigation systems”. In Proceedings of ITSC’05. IEEE Press, Vienna, Austria, pp. 197-202.
16. Romero, L. and Correia, N. (2003), “HyperReal: A Hypermedia model for Mixed Reality”, In Proceedings of the Hypertext 2003, ACM Press, Nottingham, UK, pp. 2-9.
17. Rossi, G., Gordillo, S., Challiol, C. and Fortier, A. (2006), “Context-Aware Services for Physical Hypermedia Applications”, In Proceedings of the CAMS 2006, Springer-Verlag Berlin Heidelberg, Montpellier, France, Vol. 4278, pp. 1914-1923.
18. Woolf, B. (1997), “Null object”, In Robert, C. M., Riehle D. and Buschmann F. (Ed.), Pattern languages of program design 3, Addison-Wesley, Boston, MA, USA, pp. 5–18.
19. Yesilada, Y., Stevens, R. and Goble, C. (2003), “A foundation for tool based mobility support for visually impaired web users”, In Proceedings of the WWW '03, ACM Press, Budapest, Hungary, pp. 422–430.