Service Platform for Rapid Development and Deployment of Context-Aware, Mobile Applications

Stanislav Pokraev[1], Johan Koolwaaij[1], Mark van Setten[1], Tom Broens[2], Patrícia Dockhorn Costa[2], Martin Wibbels[1], Peter Ebben[1], and Patrick Strating[1]

[1]*Telematica Instituut*
P.O. Box 589
NL-7500 AN Enschede, The Netherlands
TEL:+31 53 485 0490
FAX:+31 53 485 0400
{firstname.lastname}@telin.nl

[2]*Centre for Telematics and Information Technology,*
University of Twente. P.O. Box 217,
NL-7500 AE Enschede, The Netherlands.
TEL:+31 53 489 4454
FAX:+31 53 489 5477
{broens, dockhorn}@ewi.utwente.nl}

Abstract

In this paper we present a web services-based platform that facilitates and speeds up the development and deployment of context-aware, integrated mobile speech and data applications. The platform is capable of handling different types of context and offers sophisticated personalization mechanisms. To illustrate the usefulness of the platform and to validate the claim that cross-platform application development, in particular mobile, context-aware applications, is easier and faster with web services technologies, we present a demonstration application. It serves tourists with interesting information and services in their specific context, and contributes to the achievement of their current goals. Finally, we present a number of problems that we experienced in the implementation process as well as the feedback that we received from real users who tested our application.

1. Introduction

The development and deployment of context-aware applications is driven by the increased mobility of the end-users. This mobility urges applications - typically running on mobile phones - to act situation-dependent, or in other words to become aware of the end-users context and to adapt to it. Take an emergency button as an example: for a patient living in a nursery home pushing the button may always trigger the same action: call a nurse. But for someone moving freely, the action triggered by pushing the button might be situation-dependent: call the neighbor when an accident happens in a home environment, call an emergency team when jogging around in the woods, or alert a nearby police officer when walking in the city center. In short, mobile applications have a natural need for context information to make decisions for or on behalf of the end-users. In addition, the actions, which follow from these decisions, might again be context dependent.

Although the concept of context-aware applications has been around for more than a decade, the first serious, commercial applications are brought out just now. All these applications share the functionality to retrieve context information from heterogeneous sources, to reason with about this information, and to offer services and information that are relevant in this context.

This paper reports on a platform, whose first version was developed in the Freeband WASP project [5] and is now being improved in the Freeband AWARENESS project[4]. The platform facilitates and speeds up the development and deployment of context-aware, integrated mobile speech and data applications. It is tailored towards reuse of components and developer convenience. All components are exposed as web services to the network. One of the goals of the project was to validate the claim that cross-platform application development, in particular regarding flexible, mobile, context-aware applications, is easier and faster when using web services technologies such as SOAP, WSDL, UDDI, XML, etc. are used.

There are several aspects that make our work distinct from the previous work in the field. First, the platform that we present provides *dynamic* way to develop, deploy and integrate mobile, context-aware services. Second, it handles *many, different types of context*. Finally, the platform offers sophisticated *personalization* mechanisms to tailor the output of the different services to the current user need.

This paper is organized as follows: Section 2 presents a simple scenario to visualize the capabilities of our platform and the demonstration application. Section 3 presents the architecture of our platform. Section 4

presents a demonstration application that shows the usefulness of the platform. Section 5 presents web services-related problems that we experienced while implementing the platform and the demonstration applications. We also describe our workarounds of these problems. Section 6 discusses the feedback that we received from real users who tested our application. Section 7 presents related work in this area. Finally, Section 8 summarizes our contributions.

2. Scenario

The service platform, developed within the project, is domain agnostic. That is to say using the platform, context-aware capabilities can be added to new or existing applications in domains, from business-to-employee services to wireless services in tourism. However, to bring the capabilities of the service platform to life we developed a scenario of an American family with two children, Liddy and Luke, that visits the Netherlands for holidays. The scenario describe a context-aware application that assist tourists in navigation in unfamiliar environments, in suggesting interesting places to visit, and in communicating with people and services in their vicinity. This scenario guides the demonstration application COMPASS (COMPASS is an acronym for COntext-aware Mobile Personal ASSistant) that is presented in Section 4. In the scenario an American family with two children, Liddy and Luke, visits the Netherlands for holidays.

Dynamic navigation: On a Thursday afternoon, the Stephenson family arrives in Enschede using a small size MPV rented from Schiphol Airport. The rental car had the top rank in the suggestion list of the automatic car rental and reservation system, and really suits the needs and wishes of the Stephenson family. They have booked a hotel in the city center of Enschede. Since Mr. Stephenson has never been in the eastern part of The Netherlands before, he uses an on-line navigation guide to get to his destination. The suggested route matches the usual driving style of Mr. Stephenson during holidays: use main routes, but suggest touristic detours. The guide correctly informs him about construction works in the south of the city center and proposes an alternate route without any obstacles.

Meeting friends: During the drive to Enschede, his son Luke has found out that the daughter of one of his father's old friends, Mr. Vaneden, is also in the Netherlands, because she is on his 'buddy list' and he received a notification once they were in each other's vicinity. He calls her by simply selecting the call option that comes with the notification, and she tells him that her family is staying in a hotel in the picturesque town of Ootmarsum, about 20 km north of Enschede. Luke proposes to his family to dine with them. Since they all like the idea, Luke suggests a date in the schedule of the Vaneden family. The place to dine will be decided by Luke later, but he promises that it will be a restaurant that meets the expectations and tastes of all of them.

Interesting Places: The hotel rooms are comfortable and at the quiet side of the building. Although it is late in the afternoon, Liddy wants to have a 'sneak preview' of the touristic attractions of Enschede. It was her father's idea to visit Enschede because of the Aviation Museum and some local railroads with running steam engines, but she hopes that there are also some other attractions as well. To search for interesting information, Liddy uses a terminal provided by the hotel, because it has better interaction capabilities than her PDA. Her portal offers a personalized guide to the touristic attractions of Enschede. To tailor the guide to the visitor's interests, the site asks the visitor to explicitly describe his interests or to give permission to obtain (parts of) the profile of the visitor. Liddy chooses for the last option. Taking into account the information from her profile, the portal provides Liddy with an interactive map with all relevant points of interests in Enschede and the surrounding area. Liddy chooses for all museums within walking distance. A map is shown with a route, starting from Liddy's current position, along three museums: the Rijksmuseum Twente (Art), the Natuurmuseum (Natural history) and the Jannink museum (Textile). The maps shows estimations of the walking time based on her average walking speed as well as the visiting time for the different museums. Liddy is also notified that there will be an art market on Saturday at the Van Heek Plein starting at 10:00.

3. System architecture

The overall architecture of the platform and the demonstration application is shown in Figure 1. Four main groups can be identified in the architecture: *third party services*, the *platform*, the *demonstration application* and the *recommendation service*.

3.1 Third party services

The *3G (GPRS, UMTS) network services* provide network access capabilities, such as user identification, call setup, messaging, charging, etc. These network capabilities are accessible via web services interfaces and offered by mobile network operators.

The *context services* provide information about the context of a user, e.g. the user status (free or busy), his location, etc. Some of this information is obtained from the 3G network via web services. Context services include both services that provide information about the user such as his shopping list or his schedule, as well as services that are independent from the user but which might be relevant

when selecting a business service, e.g. weather or traffic information services.

Business services offer information and services for applications build on the platform. In the demonstration application these are services that offer so-called points of interest (POI): museums and their catalogues, monumental buildings and historical information associated with them, restaurants and their menus, shops and their current promotions, hotels with reservation services, digitized old postcards, etc.

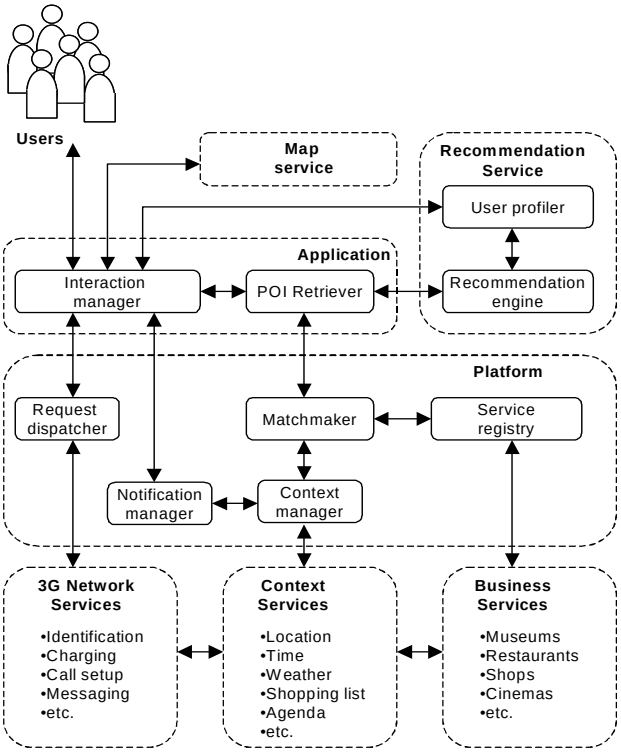

Figure 1

3.2 The platform components

The *Request dispatcher* is a component responsible for forwarding user requests to the appropriate 3G-network platform. This way, users can switch transparently to different network operators or, for instance, use different messaging services.

The *Notification manager* provides functionality for applications to subscribe and receive notifications when the context of a particular user changes. For example, when a user moves around in a city, his location changes. The notification manager notifies all applications that have registered interest in this event and provides the new location of that user. The application can then adapt itself to this change of the user's location.

The *Context manager* retrieves information about the user's context by contacting the appropriate context services. It is also responsible for aggregating the context or deriving new context based on domain specific rules. For example, the context manager can infer whether a user walks or drives given the speed of the user and the geographical properties of his current location (city street, highway, sea or river, etc) or simply from the fact that his phone is attached to a car kit. The context manager is also responsible to update the notification manager on changes in the context.

The *Service registry* contains information about the services provided by third parties. To improve the semantics of the service descriptions we use semantic web technology, notably OWL [12], to create additional annotations of service elements (e.g. service types, data in the input and output messages, etc.).

Most of the standard service search engines retrieve service descriptions that contain particular keywords from the user's query. In most of the cases this leads to retrieval of irrelevant service descriptions while services that are useful for the user are left out. The reason for the first is that the query keywords might be syntactically equivalent but semantically different from the terms in the service description (homonyms). The reason for the second is that query keywords might be semantically similar but syntactically different from the terms in service descriptions (synonyms). In our approach we use ontologies as one possible solution to this problem. This way we enable retrieval based on service types rather than just keywords.

Another drawback of the existing search engines is that the query-service description matching score is calculated taking only the keywords from the user's query and the terms in the service descriptions into account. Thus, regardless of the context of the user and the context of the services, the same list of results is returned in response to a query. By definition, *context* is a situation of an entity (person, place or object) that is *relevant* to the interaction between a user and an application. Therefore, taking the context into account in the query-service description matching process can improve the quality of the search results. However, contextual information is highly interrelated and has many alternative representations, which makes it difficult to interpret and use. In our approach we use again ontologies to specify the interrelations among context entities and ensure common, unambiguous representation of these entities.

The *Matchmaker* uses the service registry to discover the services that match the request received from an application (in this case the point-of-interest (POI) retriever). Once services are discovered based on their types, capabilities and context attributes, the matchmaker component filters out the services that do not match the criteria set by the application. To perform this action, the component uses the context ontology and domain-specific

rules provided by the application. It also interacts with the context manager to retrieve the respective required contextual information. The service registry and the matchmaker components are presented in greater detail in [2].

3.3 The demonstration application

The *Interaction manager* is a server side component responsible for finding the most appropriate way to communicate a user's request and assist the interaction of the user and the client side application (on the mobile phone, PDA or other device). For example, if a user clicks on a POI representing a restaurant, the interaction manager can if available, retrieve the restaurant's menu automatically and present it to the user or prompt to setup a phone call in order for the user to make a reservation.

The *POI retriever* receives a request from the interaction manager when the user context changes or directly by an action of the user. It creates a search request that is sent to the matchmaker component. After the matchmaker component returns the list of POIs matching the issued request and the criteria of the user's context, the POI retriever sends this list together with the user's identity and the context information to the recommendation service, which assigns scores to each POI indicating the predicted relevance of the POI for the user. The POI retriever then sends the list of POIs with scores to the client side application, which displays them.

The demonstration application also uses an external *map service*, such as Microsoft MapPoint [10] for regular maps, a map service providing aerial photographs and a map service providing old cadastral maps. These web services are used to offer dynamic and interactive maps, providing navigation support, etc. The demonstration application allows the user to switch between the various types of available maps, while keeping all other functionality, such as displaying POIs on the map and services associated with POIs.

3.4 The recommendation service

The recommendation engine uses multiple prediction strategies to predict how interesting each POI is for the user. A prediction strategy selects and/or combines multiple prediction techniques by deciding which prediction techniques are the most suitable to provide a prediction based on the most up-to-date knowledge about the current user, other users, the information for which a prediction is requested, other information items and the system itself[16]. Used prediction methods include social filtering [14], case-based reasoning (CBR) [13] and category learning[17]. For different classes of POIs, different prediction strategies can be defined in the engine. As the semantics of POIs are captured in an ontology, the recommendation engine is aware of the class hierarchy of each POI. If a prediction strategy exists for the actual class of a POI that strategy is chosen, otherwise the engine moves up the class hierarchy until it finds a parent class that has a prediction strategy associated with it. In our hierarchy, POI is the root class, which has a default prediction strategy assigned to it.

For the demonstration application, prediction techniques have also been developed that base their predictions on contextual factors; e.g. one technique predicts the relevance based on the time past since the last time the user visited a POI of that class. The more recent the user has visited such a POI, the lower its predicted relevance.

The *user profiler* maintains the profiles of all users. It is used by the recommendation engine to retrieve and store knowledge about users, such as the interests of and ratings provided by users. The interaction manager can also directly access the profile manager; this way, the interaction manager can store user preferences or it can retrieve (parts of) the user profile and present it to the user.

The *recommendation service* is not part of the platform as some prediction techniques are domain dependent or need to be tuned to specific domains, e.g. a similarity function had to be defined for the CBR-based prediction technique that compares two POIs with each other and returns a similarity score. However, the recommendation service is also not part of the demonstration application; this allows other applications in the tourist domain to use the same recommendation service. The recommendation service is described in greater detail in [15].

4. The COMPASS application

The COMPASS application serves a tourist with information and services (ranging from buildings to buddies) needed in his specific context and interesting for him given his goal at that moment. For example, a tourist who has an interest in history and architecture is served with information about nearby monuments built before 1890. A tourist who has wants to find a place for the night gets a list of hotels and campsites in and around town that match his preferences for accommodations.

After start-up, the application shows the user a map of his current location. The location is either obtained from the mobile network or from other devices such as GPS receivers. Depending on the user's profile and goal, a selection of nearby buildings, buddies and other objects is shown on the map and in a list. The map and the objects shown are updated when the user moves or his profile or goal changes. Other context changes might also force the map to change. For example, an increase in the user's

speed by starting to drive a car causes the map to zoom out automatically as the user's notion of nearness can be defined by what he can reach in a certain amount of time. Clicking on objects on the map usually means interacting with services provided by that object, e.g. calling a buddy, reserving a table at a restaurant, or booking tickets for a show.

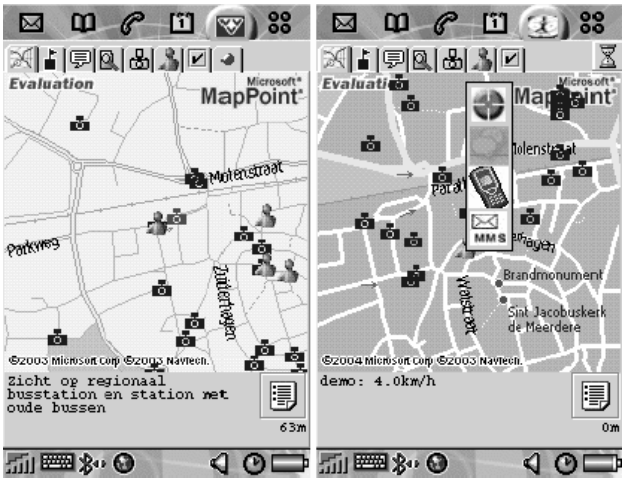

Figure 2

The platform is open, which means that third parties can easily integrate their information and services with the platform; these services can then transparently be found and used by the application users. For example, an organization that owns a collection of digitized old postcards wrapped its database with postcards as an internet-accessible web service, published the web service description in the service registry of the platform and related the web service's interface to the registry's ontology. The net effect is that all application users with an interest in such postcards are now able to view postcards depicting objects near their location instantaneously. Depending on the visualization, they see a map of their environment with icons indicating the location depicted on the old postcards (see the left image in Figure 2) or a thumbnail list of the postcards. Clicking on an icon displays the postcard, the date the picture was taken and a short description. It makes it quite easy to get an impression of the past atmosphere of while walking through a street or neighborhood.

The demonstration application accomplishes this functionality by querying the service registry for search services that are bound to deliver objects related to the user's context. The underlying platform retrieves services matching the criteria of the user's context and goal. For example, for someone located in Enschede and looking for sightseeing attractions it delivers search services for museums, landmarks, architectural buildings, etc. Next, the relevant search services are queried to retrieve the objects matching the context's criteria, e.g. to be within a certain radius from the location of the user. The retrieved objects are then sent to the recommendation engine which scores each object based on the other criteria, such as the user's interests and context. The retrieved objects and scores are then displayed on the map and in the list of objects. To process is depicted in Figure 3.

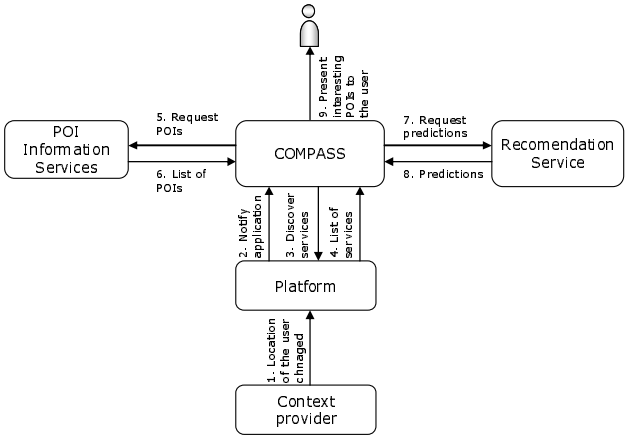

Figure 3

5. Development experience

When implementing the COMPASS application and the WASP platform we encountered a number of problems. Given the scope of this conference, we focus on the problems related to web services technologies and only briefly present the general issues related to the development of the platform and the demonstration application.

The web service related problems come from the fact that MapPoint service is implemented using Microsoft .Net[11] technology while our implementation uses Apache Axis[1] and is Java based.

The first problem we encountered was related to the authentication mechanism that MapPoint service uses. The MapPoint service uses HTTP digest authentication to validate user accounts. Unfortunately, Axis supports only basic authentication. The difference between basic and digest authentication is that in the first method the username and the password are sent as clear text, whereas in the second method hash schemes are used to create an encrypted string sequence combining the two values. To overcome this problem, we replaced the default Axis transport layer with a custom one (`SimpleHTTPSender`) that supports digest communication. The deployment description we used is shown below:

```
<deployment name="defaultClientConfig"
  xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <transport name="http"
    pivot="java:nl.freeband.wasp.mappoint.SimpleHTTPSender" />
  <transport name="local"
    pivot="java:org.apache.axis.transport.local.LocalSender" />
</deployment>
```

Another problem we experienced was that Axis does not generate any java code for the default values from the MapPoint WSDL. For example, the following segment from the MapPoint WSDL file

```
<s:complexType name="MapPointConstants">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1"
      default="6378.2" name="EarthRadiusInKilometers"
      type="s:double" />
    <s:element minOccurs="0" maxOccurs="1"
      default="3.1415926535897931" name="Pi"
      type="s:double" />
      ...
    <s:element minOccurs="0" maxOccurs="1"
      default="0.3048" name="MetersPerFoot"
      type="s:double" />
  </s:sequence>
</s:complexType>
```

resulted in a class that only contains the members defined in the WSDL (like `EarthRadiusInKilometers`) but does not contain their default values. We fixed this by manually modifying the generated java stubs classes.

Finally, we discovered a problem in the way Axis handles simple types and lists. For example, the following segment from the MapPoint WSDL

```
<s:simpleType name="DataSourceCapability">
  <s:list>
    <s:simpleType>
      <s:restriction base="s:string">
        <s:enumeration value="CanDrawMaps" />
        <s:enumeration value="CanFindPlaces" />
        <s:enumeration value="CanFindNearby" />
        <s:enumeration value="CanRoute" />
        <s:enumeration value="CanFindAddress" />
        <s:enumeration value="HasIcons" />
        <s:enumeration value="DataServiceQuery" />
      </s:restriction>
    </s:simpleType>
  </s:list>
</s:simpleType>
```

resulted in an "empty" java stub class. That is, there were no `set` or `get` methods generated for the values from the list. Again we fixed this by manually modifying the generated java stubs classes.

Our conclusion is that, although Web Services aim at improving the interoperability among different applications by defining standards for message exchange, languages for interface definition and composition of compound services, etc., there are still many open issues that have to be solved first in order for web services technology to reach its full potential.

Finally, we briefly present some general issues that we encountered during the development of the COMPASS application. The most important are:

- Most parties that have information about points of interest in a city, like tourist offices and national heritage institutions, do not have enough technical expertise to expose that information via web services.
- Moreover, if their information can be made available their information assets sometimes need heavy post-processing to make them suited for application in location-based services, e.g. by enriching data with latitude-longitude information or by the normalization of old maps.
- We started developing with device independence in mind, using the phone as an internet gateway to our web application. However to enhance the user interaction and experience we switched to a generic Java application. and finally, we ended up with about 10% of our native code to used to work with the phone camera and jog dial, which, unfortunately, does not generalize to other devices.
- Battery life becomes a limiting factor for a P800 phone with active Bluetooth connection, continuously lit screen, open GPRS connection and a few important background processes running.,.

The platform offers functionality that is general enough to support other context-aware applications. Using the same platform, we also developed a "find-a-new-home" application based on CellID positioning techniques as well as some other applications.

6. User experience

We have exposed the COMPASS application to reality tests in Enschede using small test teams, consisting of an employee of the local tourist office, one tourist (or someone who has not seen the application before) and a researcher. Each test team had a P800 phone, the COMPASS application and a bluetooth GPS receiver. Some teams wandered through the city without a specific goal, just like tourist tend to do, other teams were instructed to simulate a young couple with child interested in city architecture, or, to follow a particular predefined city tour. These teams came up with a lot of comments and experiences from the test walks. In general, the inexperienced users were impressed by the new possibilities of the application, but at the same time they judged it as too complex for a normal tourist. In more detail, the perceived complexity was related to the following findings:

- Working with a PDA-type of mobile phone, such as the P800, has a steep learning curve, especially working with a stylus on a display of about 300 by 200

pixels. Therefore, interacting with the phone was difficult, which in turn influenced the user experience of interacting with the COMPASS application.

- The small screen size forced us to work with multiple tabs (see Figure 2). For that reason, to obtain different pieces of information, the users had to browse through different tabs.
- The context-aware functionality of the COMPASS application caused unpredictable behavior, which was not always appreciated by the users. We see this as an important research issue, since sometimes the predictability is more important the adaptive behavior imposed by the application. For example, nobody is surprised that if he moves, the map changes, but he can become vary confused if his favorite restaurant is missing on the map because he has visited it recently.
- Having the GPS as a separate device causes some undesired hassle.
- The thin client and the limited bandwidth of the GPRS connection caused delays in the interaction. Sometimes we were able to predict the need for new data and preload it (e.g. a new map when the user is moving in a certain direction) but that is not possible, for example, for explicit user requests (e.g. for an image of a monument).

It is clear that these observations have more to do with mobile application development in general than with the web services or context-aware aspects of those applications. In general, the people liked the surprise effect very much, for instance, that the system directs them to unexpected places or provides new information about already known places. They also appreciated that it was possible to request information about nearby static points of interest (e.g. monuments, shops or nearby toilets) as well as dynamic point of interests (e.g. the location of their friends in the city) at anytime. The local inhabitants enjoyed the ability to browse through the different periods in the history of their current locations very much.

7. Related work

The goal of Cyberguide[9] project was to investigate future computing environments. It focuses on location and orientation as contextual information and provides a position-aware tour guide for tourists.

The CoolTown project[7] ties web resources to physical objects and places, and allows users to interact with these resources using their mobile devices.

The goal of GUIDE[3] project was to develop electronic touristic guides for visitors of Lancaster. GUIDE utilizes a cell-based wireless communications infrastructure (e.g. GSM) in order to broadcast dynamic information and positioning information to portable GUIDE units that run a web browser.

Smart Sight[18] provides an intelligent tourist assistants system that tries to remove the language barriers, provides navigation assistance, and handles queries posed and answered in natural language. The system does that by correlatating information from different sensors (e.g. GPS, video camera, sound recorder).

AccesSights[8] focuses on providing blind and visually impaired people with touristic information, based on their location. The proposed platform acts as intermediary between the tourist application and the application specific services such as map sources and tourist information.

There are several aspects that make our distinct that the previous work in the field. First, the platform provides *dynamic* way to develop, deploy and integrate mobile, context-aware services. Second, it handles *many, different types* of context. Finally, the platform offers *sophisticated personalization* mechanisms to tailor the output of the different services to the user need.

8. Conclusions

We have demonstrated a specific context-aware application targeted at tourists, built upon a generic platform that facilitates the development and deployment of context-aware applications. We have demonstrated the applicability of the web service paradigm to improve developer convenience, and to encourage re-use of components, and transparency of interfaces and functionality. We also scratched the surface of what is possible with context ontologies. Once the ontologies are in place, they offer substantial semantic expressive power, which allows the definition of semantically rich queries, smart processing of those queries and intelligent interpretation of the query results. However, we experienced that creation, maintenance and combination of ontologies can be time-consuming. Substantial tooling effort is needed on that part to make ontology management more convenient.

Our work also shows how valuable data, buried deeply down in different archives (e.g. cadastral maps, old aerial pictures, museum databases, etc) can become “alive” for the normal people, enabling them to learn more about their current environment. Finally, the knowledge we acquire generated a lot of awareness about context-aware service development in the Dutch circle of application developers.

9. Acknowledgements

The first version of the platform was developed in Freeband WASP[5] project. An improved version is being developed in Freeband AWARENESS[4] project.

Freeband[6] is sponsored by the Dutch government under contract BSIK 03025.

We would like to thank Rogier Brussee and Maarten Steen from Telematica Instituut for providing us with useful comments and suggestions to improve this paper.

References

[1] Apache Axis, http://ws.apache.org/axis/

[2] Broens, T., Pokraev, S., van Sinderen, M., Koolwaaij, J. & Dockhorn Costa, P. Context-aware, ontology-based, service discovery. In: Markopoulos, p. [et al.] (eds): Proceedings of Second European Symposium on Ambient Intelligence, Eindhoven, Netherlands, LNCS 3295, Springer-verlag, pages 363-367, 2004

[3] Cheverst, K., Davies, N., Mitchell, K., Friday, A.,Efstratiou, C., Developing a Context-aware Electronic Tourist Guide: Some Issues and Experiences, Proceedings of Human Factors in Computing Systems (CHI'00), the Hague, the Netherlands, 2000

[4] Freeband AWARENESS, AWARENESS - context AWARE mobile NEtworks and ServiceS, http://awareness.freeband.nl/

[5] Freeband WASP, Web Architectures for Services Platforms, http://wasp.freeband.nl/

[6] Freeband, http://www.freeband.nl/

[7] Kindberg, T., Barton, J. A Web-Based Nomadic Computing System. Computer Networks, Amsterdam, The Netherlands 1999.

[8] Klante, P., Krösche, J., Boll, S., AccesSights - A Multimodal Location-Aware Mobile Tourist Information System, Proceedings of Conference on Computers Helping People with Special Needs (ICCHP'04), Paris, France, 2004

[9] Long, S., Aust, D., Abowd, D., Atkenson, C., Cyberguide: Prototyping Context-Aware Mobile Applications, Proceedings of Human Factors in Computing Systems (CHI'96), Vancouver, Canada, 1996

[10] Micosoft Mappoint, http://www.microsoft.com/mappoint

[11] Mictosoft .Net, http://www.microsoft.com/net/

[12] OWL, Ontology Web Language, http://www.w3.org/2001/sw/WebOnt

[13] Riesbeck, C. K., Schank, R.: Inside CBR. Lawrence Erlbaum Associates, Northvale, NJ, USA (1989)

[14] Shardanand, U., Maes, P.: Social information filtering: algorithms for automated "Word of Mouth". In: Proceedings of Human factors in computing systems 1995 (New York, USA). ACM (1995) 210-217

[15] van Setten, M., Pokraev, S., & Koolwaaij, J. Context-Aware Recommendations in the Mobile Tourist Application COMPASS, In Nejdl, W. & De Bra, P. (Eds.). Adaptive Hypermedia 2004, Eindhoven, The Netherlands, LNCS 3137, Springer-Verlag, pages 235-244, 2004

[16] van Setten, M., Veenstra, M., Nijholt, A., van Dijk, B.: Case-Based Reasoning as a Prediction Strategy for Hybrid Recommender Systems. In: Proceedings of the Atlantic Web Intelligence Conference, Cancun, Mexico, Springer-Verlag, LNAI 3034 (2004) 13-22

[17] van Setten, M.: Experiments with a recommendation technique that learns category interests. In: Proceedings of IADIS WWW/Internet, Lisabon, Portugal (2002) 722-725

[18] Yang J., Yang, W., Denecke, M., Waibel, A., Smart Sight: A Tourist Assistant System, Proceedings of IEEE International Symposium on Wearable Computers (ISWC'99), Victory, Canada, 1999