Context-Aware Mashups for Mobile Devices

Andreas Brodt[1], Daniela Nicklas[1], Sailesh Sathish[2], and Bernhard Mitschang[1]

[1] Universität Stuttgart, Institute of Parallel and Distributed Systems,
Universitätsstraße 38, 70569 Stuttgart, Germany
[brodt,nicklas,mitsch]@ipvs.uni-stuttgart.de
[2] Nokia Research Center, Visiokatu 1, 33720 Tampere, Finland
sailesh.sathish@nokia.com

Abstract. With the Web 2.0 trend and its participation of end-users more and more data and information services are online accessible, such as web sites, Wikis, or web services. So-called mashups—web applications that integrate data from more than one source into an integrated service—can be easily realized using scripting languages. Also, mobile devices are increasingly powerful, have ubiquitous access to the Web and feature local sensors, such as GPS. Thus, mobile applications can adapt to the mobile user's current situation.
We examine how context-aware mashups can be created. One challenge is the provisioning of context data to the mobile application. For this, we discuss different ways to integrate context data, such as the user's position, into web applications. Moreover, we assess different data formats and the overall performance. Finally, we present the Telar mashup platform, a client-server solution for location-based mashups for mobile devices such as the Nokia N810 Internet Tablet.

1 Introduction

The proliferation of public web services and other online data sources enables new services and applications that combine existing information in a new manner. So-called mashups integrate data and services from multiple sources to provide innovative services, like the visualization of crime statistics from the Chicago Police Department on a street map[3] (one of the first mashups). Mashups are mostly realized by web pages that leverage script languages such as JavaScript, which enables better user interactivity by locally executed functions and the dynamic loading of data from web services. These techniques are often associated with the Web 2.0 trend. Typically, mashups are created dynamically from existing data sources that have no knowledge about their participation. Thus, they can change their interfaces and data formats at any time, which adds a new flavor to the general data integration problem.

Another trend are mobile systems that have become more and more powerful in the last years. Soon, users expect their handheld devices to run the same applications as their desktop systems do. In addition to that, mobile devices

[3] http://chicagocrime.org

feature an increasing number of built-in or easily connectable sensors, such as cameras, GPS receivers, or acceleration sensors. One example is the Nokia N810 Internet Tablet that possesses 128 MB of RAM, 2 GB of flash memory, an OMAP2420 microprocessor at 400 MHz as well as a small camera and a GPS receiver. This is sufficient to run a Linux-based operating system and thus numerous Linux applications, including context-aware software, such as car navigation.

However, mobile systems still have limitations in many resources, such as display size, communication bandwidth (due to their wireless connections), energy, or the user's focus. The latter imposes one of the major design issues: mobile applications should be able to adapt to the user's situation. Every piece of information that helps the application to detect that situation is referred to as context. Such adaptive applications are called context-aware applications [1].

The combination of these two trends—mashups and context-aware applications on mobile devices—offers great additional value to the user. By integrating multiple data sources into one experience, new services can be created that are tailored to the user's personal needs. And by using local sensor data on a mobile device, this experience can be adapted to the user's current situation. In this paper, we examine the creation of context-aware mashups according to the following requirements:

- Adaptation should be based on sensors which are built into the mobile device or locally connected. Although our scenario focuses on location data gained from a GPS receiver, the solution should allow arbitrary local sensors.
- The mashups should be user-centric, i. e., the user of a mobile device should benefit from the mashups, rather than a remote person or service provider.
- The mashups should be viewable with the web browser of the mobile device. This prevents a native solution on the mobile device and has potential influence on performance.
- There should be a non-adaptive version of mashups in case no context information is available. This allows viewing the mashups on sensor-equipped mobile devices as well as on desktop computers and thus increases the usefulness of the mashups.
- Multiple data sources using arbitrary data formats and interfaces should be integrated into the mashups. The user should be able add and remove data sources at runtime, according to her current interest.

The example scenario on which we focus in this paper is a context-aware mashup which integrates the user's position obtained from a GPS device, a map from an online map service, and nearby points of interest (POIs) from different online data providers, as depicted in Figure 2. The POI metaphor is used by a large portion of geo-mashups, so scenario covers the main use case of these mashups. We put this scenario into practice using a three-tier system architecture with a Nokia N810 Internet Tablet as the client device. We demonstrated this sample scenario at the EDBT'08 conference [2].

The main contribution of this paper is a system architecture featuring a context provisioning framework for utilizing local sensors in context-aware mashups. We

give an overview on related work in Section 2. In Section 3 we identify the requirements of a context provisioning framework for context-aware mashups, address data integration, and describe our proposed system architecture. We shortly give a few implementation details of the TELAR mashup platform in Section 4, discuss the performance of AJAX on mobile devices and illustrate our optimizations. Finally we conclude the paper in Section 5.

2 Related Work

2.1 Context Provisioning

Context is any information that can be used to characterize the situation of an entity [1], which is used to adapt the behavior of a context aware application. In our example scenario, the entity would be the user, and context are his or her location and interest. While the interest—i.e., which data providers should be included in the mashup—is configured by the user, the location should be provided automatically to the mashup system, i.e., by a sensor.

In related research, several context provisioning systems were proposed, e. g., the Context Toolkit [3], the context manager of Henriksen et al. [4], or the Nexus platform [5], to name a few. Unfortunately, none of these approaches are mature enough to build them into a running product, not to talk about standardization. Also, they are meant to manage a complex infrastructure of installed sensors and context services, which is not necessary for context-aware mashups.

On the web, context-aware web pages currently integrate context information obtained from third-party web services. Today, web services locating the user's IP address, such as hostip.info, are able to determine the user's country and occasionally the town. This can be sufficient to embed advertisements into a web page. Services like plazes.com or jaiku.com support uploading context information, such as the current location and activity. Some people embed this information into their web page or blog in order to share it with friends. In principle, it is possible to upload context data to a third-party service. However, this approach cannot satisfy the time and accuracy requirements of systems such as an electronic tour guide.

Heading towards device independent web applications several standards have been defined for supplying context information of a kind. HTTP headers may provide information about the client's web browser and could be extended to convey other context data [6]. However, such extended HTTP headers lack asynchronous notifications, search, control and standardization. Approaches based on profiles, such as CC/PP [7] and UAProf [8], provide only static context attributes, thus lacking asynchronous notifications and mutability. The Device Profile Evolution (DPE) [9] solves this problem, but is limited to mobile phones and follows a server-based approach. However, to reduce latency and increase accuracy, client-based provisioning is desirable.

The context provisioning framework that matches our requirements best is the W3C Delivery Context Client Interfaces [10]. DCCI is a client-based framework

to which both local sensors and remote services can be bound. DCCI represents context as a number of *properties* which are organized hierarchically, using the W3C Document Object Model (DOM). Web pages can access the resulting property tree via JavaScript. DCCI properties may be static (e. g., screen size) or dynamic (e. g., the user's position). The main advantage of DCCI is that it is undergoing standardization within W3C and is (at the time of writing) expected to reach proposed standard status soon.

2.2 Mashups

Mashups are web application hybrids that integrate data from different sources to provide a value added service. They leverage the availability of open web services, RSS feeds, or extract information out of regular web pages using screen scraping. A popular combination is to display information from different sources on a map (geo-mashups), or to enrich search results from one source (e. g., a hotel finder) with information from others (e. g., recommendations and pictures).

The mashup portal programmableWeb.com lists 3046 mashups on May 20, 2008, with an average of 3.14 new mashups per day. Over one third are geo-mashups, 17% are multimedia mashups (video and photo). Floyd et al. [11] show how mashup techniques can be used for rapid prototyping in user-centered software development processes. A study at the Human-Computer Interaction Institute of the Carnegie Mellon University showed that mashups can be even used for end-user programming [12]. IBM emphasizes the great benefits of so-called *Enterprise Mashups* [13], information heavy applications that integrate distributed information within an enterprise in a quick and dynamic way. Erik Wilde [14] applied the mashup idea to the management of large knowledge bases.

Most of the existing mashups are programmed manually. However, a number of mashup platforms exist that facilitate the development: *Mash-o-matic* [15] can be used to generate geo-mashups based on so-called superimposed information. The *Openkapow* platform[4] realizes mashups as a combination of so-called robots, which extract information from RSS streams, web services, or via screen scraping. With online tools like *Yahoo! pipes*[5] or Microsoft's *Popfly*[6], mashups can be built out of predefined components and combined using interactive drag-and-drop interfaces. IBM's *QEDWiki*[7] is an AJAX interface to combine user interface components that are connected to external data providers. IBM is also working on a data mashup service for web and enterprise information called *DAMIA*[8]. Intel's *MashMaker* [16] allows the creation of complex mashups by browsing, rather than writing code, applying the principles of functional programming.

While these platforms are good at integrating various data sources into a new presentation, none of them is capable of utilizing local sensors. Also, our

[4] `http://openkapow.com`
[5] `http://pipes.yahoo.com`
[6] `http://www.popfly.ms`
[7] `http://services.alphaworks.ibm.com/qedwiki`
[8] `http://services.alphaworks.ibm.com/damia`

scenario focuses on independent points of interest (POIs), thus complex interaction between data providers, e. g., as supported by Yahoo! pipes or MashMaker, is not necessary. On the other hand, our scenario may require a lot of flexibility with respect to accessing the data providers.

3 Context-Aware Mashups

3.1 Context Provisioning

Context-aware mashups (and context-aware web-applications in general) based on local sensors require a means of context provisioning with the following characteristics:

Asynchronous Notifications. The mashup needs to be notified of changes in the user's context in order to react accordingly.

Mutability. As sensors may become inactive or new sensors may be connected to the mobile device, the context model needs to be capable of reflecting these changes.

Search. The mashup needs a means to find out, which kinds of context data are available. This in turns requires **metadata** describing the context data.

Control. In order to utilize a local sensor, the mashup needs some degree of control over the sensor, e. g., to activate it or to trigger a measurement.

Standardization. It is of utter importance to have standardized interfaces when exposing an API to a huge multi-platform and multi-vendor system such as the web.

Privacy. The user must be in control of what information is disclosed about her current situation. As mashups are third-party applications, the context provisioning system must ensure privacy before the context data reaches the mashups. The privacy aspect of context provisioning, however, is not in the focus of this paper.

As discussed in Section 2.1, not every context provisioning framework fulfills these requirements. We chose the W3C Delivery Context Client Interfaces (DCCI) [10] as a standardized means for context provisioning, since it possesses most of these characteristics. DCCI uses the DOM event model to provide asynchronous notifications. A web page can register for events, such as the change of a property value or the removal of a property, and is notified via the provided JavaScript event listener function. In addition to that, it is possible to add and remove properties dynamically, thus achieving mutability. Moreover, DCCI properties can have a metadata interface and the DCCI tree is searchable. DCCI does not directly support control over local sensors. However, for simple notions of control, such as activating or triggering a sensor, the event model can be used. Assuming that the value of sensor is only needed when at least one event listener is registered to the respective DCCI property, the property can (de)activate or trigger the sensor accordingly.

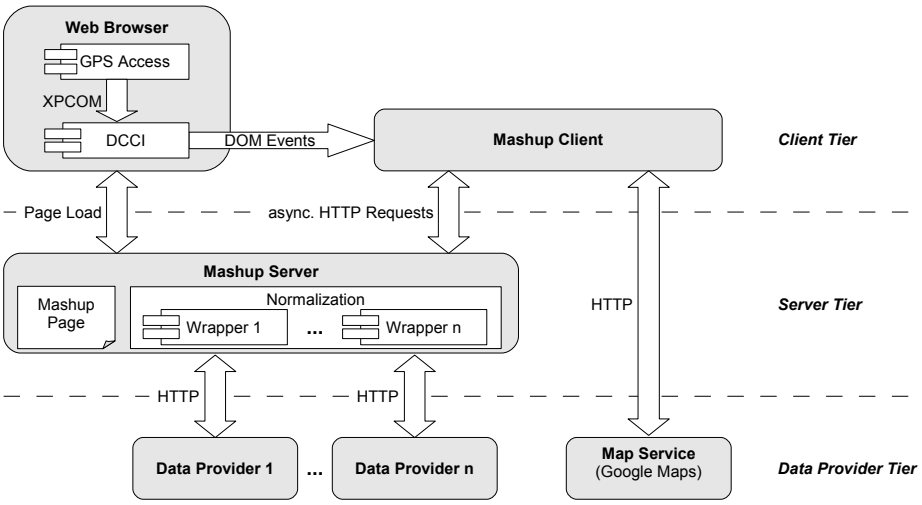

Fig. 1. Proposed system architecture for context-aware mashups

The drawback with DCCI is that it is meant to be a consumer interface and therefore lacks a standardized provider interface. Moreover, DCCI is meant to be used within a wider framework with security controls and object management that is not addressed within the current specification.

3.2 Data Integration

Our solution to integrate the data from various different data providers is a simple wrapper approach. Small and independent wrapper scripts impose an abstraction layer on the data providers creating a consistent RESTful interface to retrieve the data. The wrappers query the data providers and convert the data into a single well-known format. As our scenario focuses on POIs, a common data model is easy to find. The fact that the wrappers need to be programmed makes user-programming difficult, on the other hand virtually any data source can be accessed. However, given a sufficient amount of wrappers, one can choose which data providers to include in a mashup. In addition, wrappers can be parametrized giving the user or the mashup creator some control.

3.3 Architectural Overview

A graphical overview on our proposed system architecture is given in Figure 1. As with a typical AJAX-based mashup, there are three tiers: A mashup is viewed in the client tier. The web browser loads the mashup page and starts the JavaScript code of the mashup client AJAX application. The mashup page is loaded from the mashup server, which resides on the Internet and constitutes the server tier. Data

offered by third-party data providers is used, which are distributed throughout the Internet. The map is loaded from a map service, which, together with the data providers, makes up the data provider tier. Note that the data provider tier is outside of the organizational boundaries of the mashup.

A mashup consists of an HTML page importing the JavaScript files of the mashup client. The mashup client is only deployed to the mashup server and used as-is. It needs to be configured (most notably, the data providers to use and the initial position of the map), but not programmed. The mashup client asynchronously reads the configuration when the mashup page is loaded. The mashup client then constructs the user interface. It displays a map and visualizes POI data from the data providers. In order to cope with the heterogeneity of data formats and interfaces used by the different data providers, a normalization layer is required. As mentioned in Section 3.2, we used programmed wrappers to achieve a consistent interface for data retrieval, as this approach offers the highest degree of flexibility. For other scenarios with less diversity, other, possibly automatic normalization approaches are applicable (e. g., data providers solely providing RSS feeds, as used by Yahoo! pipes).

Context information, such as the user's location, is integrated into the mashup by extending the web browser. There are two additional components: the DCCI module and the GPS access module. The DCCI module implements the DCCI specification [10] and constitutes the interface for providing context data to web pages. The mashup client registers itself as an event listener to the DCCI module and is notified whenever the user's location changes. The GPS access module connects to the GPS device and ships the location information to the DCCI module. Dividing the context provisioning framework into a client interface and a provisioning module allows further context provisioning modules to be added later on.

The data flow works as follows: Whenever the GPS access module obtains a new location from the GPS device, the location information is updated in the DCCI module. The mashup client, which is registered as an event listener to the DCCI module, is notified about the change via DOM events. Subsequently, the mashup client updates the user's location on the map and centers the map to the new location. If the area shown on the map has significantly changed, the mashup client sends asynchronous HTTP requests to the wrappers, in order to obtain POI data for the new map area. The wrappers translate these requests into calls to the particular APIs of the data providers and convert the resulting data into a unique data format understood by the mashup client. Finally, the mashup client reads the reply sent by the wrappers and visualizes the POI data on the map.

3.4 Mashup Application Development

In order to create a mashup using our proposed architecture, the following steps need to be taken:

1. Select the data providers. Create the respective wrappers, if not yet available.

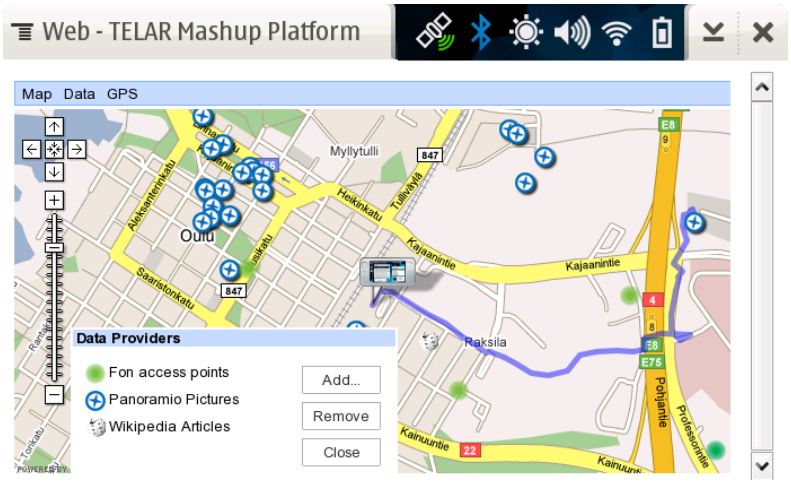

Fig. 2. Screenshot of a TELAR mashup on a Nokia N810

2. Write an HTML page, into which the map presentation should be embedded.
3. Deploy the mashup client, the wrappers, and the HTML page to a web server.
4. Configure the mashup client defining the initial map area (center point and zoom level) and the data provider wrappers to use.

No AJAX programming is required, as the mashup client handles all map interaction, displays the POIs and integrates the user's location.

4 The Telar Prototype

4.1 Implementation

We put our proposed architecture into practice on a Nokia N810 Internet Tablet, as depicted in the screenshot in Figure 2. The Mozilla-based web browser of the Nokia N810 provides a powerful extension mechanism, using which the DCCI module and the GPS access module were implemented in C++. The two modules communicate directly via XPCOM, the component framework of the Mozilla browser. As DCCI only defines a client interface, the DCCI module had to define its own provider interface as the back end. Although we did not aim at implementing a full-fledged provider framework, as proposed in [17], this provider interface is generic and can be used for different kinds of context data providers. Based on the provider interface, further browser extensions can be written, which expose context data via one or more DCCI properties.

To implement the mashup client we used the Google Web Toolkit[9]. This gave us the possibility to write the non-trivial mashup client in Java utilizing

[9] `http://code.google.com/webtoolkit`

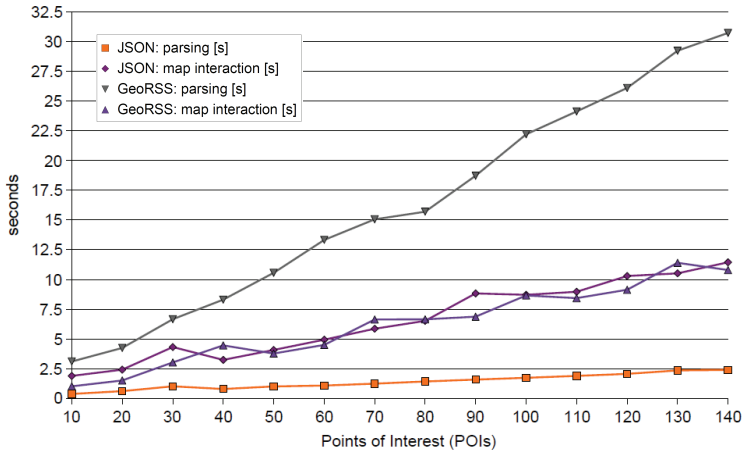

Fig. 3. Time for processing POIs in GeoRSS and JSON format

professional development tools. Moreover, the menus and dialogs making up the user interface of the mashup client could be easily created with the widget library of the Google Web Toolkit.

The wrappers were implemented in PHP 5 using XSL stylesheets to convert from XML-based data formats. The wrappers are significantly shorter than 100 lines of code and their implementation was very straight forward. Thus, additional data providers can be added to the Telar mashup platform without problems and changes in the interface of data providers can be easily resolved.

4.2 Performance Optimizations

When the mashup client and some wrappers were implemented, first tests showed that performance was insufficient. Working fine on a state-of-the-art desktop PC, it could take minutes until a mashup was completely constructed on a Nokia Internet Tablet. A similar experience was made on a Pentium II PC. A first analysis revealed that most of the time was spent for parsing the POI data which the mashup client retrieved from the wrappers. It took additional time to draw the POIs on the map. Both steps are done by JavaScript code interpreted in the browser. In contrast to a desktop PC, the processor of the Nokia Internet Tablet was simply not powerful enough to do this job quickly.

Our first version used an extended version of GeoRSS [18] as serialization format for the POIs. GeoRSS is a standardized, simple, and popular format based on Atom or RSS for describing geographically annotated objects. By using GeoRSS, existing tools and web pages supporting GeoRSS could be used for testing the wrappers and the wrappers could even be reused for other applications.

In order to improve performance, the standardized GeoRSS format was replaced by a proprietary JSON format [19]. As JSON is a subset of JavaScript,

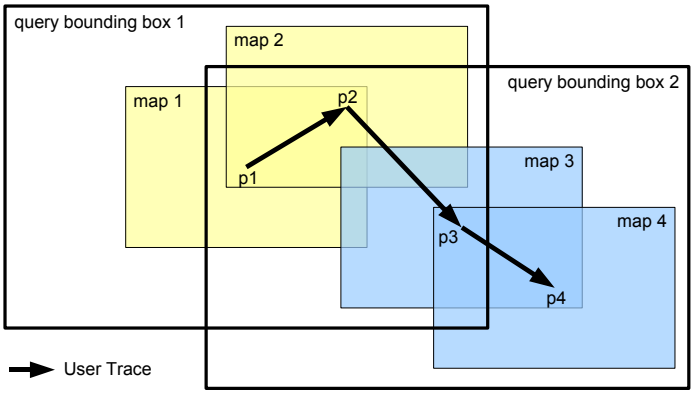

Fig. 4. Query bounding boxes for reducing the number of queries

it can be parsed very efficiently using the `eval()` JavaScript function, which the web browser implements in native code. Figure 3 shows the performance measurements done with GeoRSS and JSON. For both formats two values were measured on a Nokia Internet Tablet with various amounts of data: the time for unmarshalling the data into objects and the time for drawing these objects on the map. The readings show that for GeoRSS parsing takes much longer than drawing the POIs. Parsing 100 POIs of GeoRSS data takes more than 20 seconds, whereas parsing 100 POIs from JSON takes less than three seconds. As the POIs are in any case unmarshalled to objects first, the time to draw the POIs on the map is independent of the serialization format. Drawing 100 POIs takes between eight and ten seconds. Thus, despite of using JSON to serialize the POIs, adding the POIs to the map still takes too long.

As adding POIs to the map is expensive for devices with limited resources, unnecessary calls to the map API should be avoided. In the scenario of a walking user, the map changes every time the GPS access module updates the location data in the DCCI tree, e. g., in intervals of a few seconds, but only to a small degree. It would be sufficient to query POIs for the area which was added to the map with the last location update. However, this is not easily feasible, as the new area is generally not rectangular. As most data providers only support rectangular query areas, each query would have to be split in two separate queries, doubling latency.

Instead, we introduced the approach of an *extended query bounding box*. Whenever the mashup client needs new POI data, 25 % of padding is added to the map bounding box. Then, POI data is queried for the resulting query bounding box. Whenever the map bounding box changes, the mashup client first tests, whether the new map bounding box is still within the last query bounding box. Only if this is not the case, a new query bounding box is calculated and new data is queried.

Figure 4 illustrates the query bounding box. To display POIs for map 1, query bounding box 1 is calculated and POIs are retrieved accordingly. As the user

moves from position p1 to p2, map 2 is displayed. But as the map bounding box of map 2 is within query bounding box 1, no new data is required. Only when the user walks to position p3, the map bounding box of map 3 is not entirely within query bounding box 1 and new data must be queried. For map 4, the POI data fetched for query bounding box 2 is available again.

As the area for which the data providers are queried is a lot larger in the approach of the query bounding box, a larger number of POIs is retrieved and the initial loading time of the mashup even increases. But once the mashup is loaded, significantly fewer queries are needed. In addition to that, the approach of the query bounding box can cope with changes of the user's walking direction very well. The query bounding box is extended in all directions, so if the user changes his direction, the POIs will be still available for a while.

5 Conclusions

In this paper, we examined user-centric context-aware mashups for mobile devices on a sample scenario of location-based mashup integrating points of interest (POIs) from arbitrary data providers. We formulated a list of requirements for a context provisioning framework for context-aware mashups and found the W3C Delivery Context Client Interfaces (DCCI) to fulfill these requirements best. We addressed the challenge of integrating POI data from arbitrary heterogeneous data providers. For our scenario, we chose an approach based on manually programmed wrappers to abstract from the various data formats and interfaces, trading flexibility with the possibility of user programming. Proceeding from these fundamental design decisions, we presented a system architecture for context-based mashups making wide use of AJAX. Based on this architecture, we implemented the Telar mashup platform, which puts our scenario into practice on a Nokia N810 Internet Tablet.

We shared our experience with rich web applications making intensive use of AJAX on a mobile device with limited resources: the performance can be sufficient for small applications but is not yet fully satisfying. By switching from GeoRSS to JSON as the serialization format for the POIs we could improve performance significantly. The same holds for a caching strategy based on extended query bounding boxes. Despite these measures, the mashup is still far from comparable to a native application or from viewing the mashup in a PC browser. This is mainly due to the fact that AJAX applications, such as the mashup client, use the web browser in a way for which it was not originally designed, which leads to worse performance.

A main future challenge is implementing privacy in context provisioning frameworks like DCCI. For the purposes of the Telar mashup platform, a simple mechanism similar to popup blockers would be sufficient. However in order to widely equip web browsers with context frameworks, a general solution is required. Further potential for future work lies in the sustainability of mashups, so that even in the case of data providers failing or changing their interfaces mashups can continue to provide user value.

References

1. Dey, A.K.: Understanding and using context. Personal and Ubiquitous Computing **5**(1) (2001) 4–7
2. Brodt, A., Nicklas, D.: The TELAR mobile mashup platform for Nokia internet tablets. In: Advances in Database Technology - EDBT 2008, 11th International Conference on Extending Database Technology, Munich, Germany, Proceedings. (2008 (to appear))
3. Salber, D., Dey, A.K., Abowd, G.D.: The context toolkit: Aiding the development of context-enabled applications. In: CHI. (1999) 434–441
4. Henricksen, K., Indulska, J.: A software engineering framework for context-aware pervasive computing. In: PerCom, IEEE Computer Society (2004) 77–86
5. Mitschang, B., Nicklas, D., Großmann, M., Schwarz, T., Hönle, N.: Federating location-based data services. In: Data Management in a Connected World. (2005)
6. Daviel, A., Kaegi, F.A., Kofahl, M.: Geographic extensions for http transactions. Internet draft, The Internet Society (September 2007)
7. Klyne, G., Reynolds, F., Woodrow, C., Ohto, H., Hjelm, J., Butler, M.H., Tran, L.: Composite capability/preference profiles (cc/pp): Structure and vocabularies 1.0. Recommendation, W3C (January 2004)
8. OMA Device Capability working group: Uaprof. Specification, Open Mobile Alliance (October 2001)
9. OMA Device Capability working group: Device profile evolution architecture. Draft, Open Mobile Alliance (September 2007)
10. Waters, K., Hosn, R.A., Raggett, D., Sathish, S., Womer, M., Froumentin, M., Lewis, R.: Delivery context: Client interfaces (dcci) 1.0. Candidate recommendation, W3C (December 2007)
11. Floyd, I.R., Jones, M.C., Rathi, D., Twidale, M.B.: Web mash-ups and patchwork prototyping: User-driven technological innovation with Web 2.0 and Open Source software. In: HICSS, IEEE Computer Society (2007)
12. Wong, J., Hong, J.I.: Making mashups with marmite: towards end-user programming for the web. In Rosson, M.B., Gilmore, D.J., eds.: CHI, ACM (2007)
13. Jhingran, A.: Enterprise information mashups: Integrating information, simply. In Dayal, U., Whang, K.Y., Lomet, D.B., Alonso, G., Lohman, G.M., Kersten, M.L., Cha, S.K., Kim, Y.K., eds.: VLDB, ACM (2006)
14. Wilde, E.: Knowledge organization mashups. TIK Report 245, ETH Zürich (Swiss Federal Institute of Technology) (March 2006) available at `http://dret.net/netdret/publications#wil06f`.
15. Murthy, S., Maier, D., Delcambre, L.M.L.: Mash-o-matic. In Bulterman, D.C.A., Brailsford, D.F., eds.: ACM Symposium on Document Engineering, ACM (2006)
16. Ennals, R., Gay, D.: User-friendly functional programming for web mashups. In: ICFP '07: Proceedings of the 2007 ACM SIGPLAN international conference on Functional programming, New York, NY, USA, ACM (2007) 223–234
17. Sathish, S., Pettay, O.: Delivery context access for mobile browsing. In: ICCGI '06: Proceedings of the International Multi-Conference on Computing in the Global Information Technology, Washington, DC, USA, IEEE Computer Society (2006) 18
18. Reed, C., Singh, R., Lake, R., Lieberman, J., Maron, M.: An introduction to georss: A standards based approach for geo-enabling rss feeds. White Paper OGC 06-050r3, Open Geospatial Consortium Inc. (July 2006)
19. Crockford, D.: The application/json media type for javascript object notation (json). Request for Comments 4627, The Internet Society (July 2006)