A Browser-based UI Framework for Smart Interaction with Ambient Services

Heinz-Josef Eikerling, Matthias Benesch, Frank Berger

Distributed Interactive Systems
Siemens AG IT Solutions & Services C-LAB
Fürstenallee 11
D-33 102 Paderborn
<forename>.<surname>@c-lab.de

Abstract: The support of smart interaction in ubiquitous computing is among the core criteria to achieve user acceptance and thus business success. We describe an architectural framework usable in various application domains to ease the interaction of the user with functions of devices in his ambience. We assume that the interaction is formalized through a service-oriented wrapping of the device functions, i.e. as services coming along with the device. The framework is constituted by a middleware that can be plugged into the communication between UI and the services for supporting push and pull mode interactions. Through applying the framework situational awareness as a means to improve the user processes according to key process metrics (i.e., latency, quality, and efficiency) can be leveraged.

1 Introduction

We envision a world, in which gadgets of daily life increasingly ship with computing and communication capabilities. The according device functions (e.g., for sensing, tuning) can be dynamically incorporated in processes controlled by a user interface (UI) running on a mobile or stationary terminal (cell phone, laptop, desktop PC, paddle, …). The implementation of the user / device interaction usually follows a request / response scheme. A major demand here is to keep the interaction *smart*, i.e. the exchange of control information between the UI and the target device should be limited to the bare essentials. Thus some level of customization / personalization needs to be featured in the controlling system the UI is part of.

Native UIs targeting a specific terminal device platform have the major drawback of not supporting genericity. This might result in costly deployments especially for handling updates. In contrast to this, a browser is a rather generic UI concept for tangible user / device interaction which is supported for an extensive lineup of end user devices. Devices range from desktop computers over PDAs / cell phones etc. to more advanced technologies like for instance Microsoft Surface which in essence delivers a browser-alike UI concept to access and manipulate data.

In this paper we describe an extension of the browser paradigm. Whereas the traditional request / response scheme devises a data *pull*, we enhance the scheme by also permitting to *push* data to the UI. This is achieved by introducing extensions being plugged into the browser. As will be discussed later on, the browser and the extensions yield a *framework*, which is highly configurable and can be used in various settings. The extension accesses a *service* which can be configured to adapt the interaction depending on *context* data[1] being supplied by an external device. Such context can be given by the status of the surrounding devices which are going to be accessed in order to retrieve their status or tune their settings.

We start with a description of this service, then explain how it is used in our UI framework for service push and afterwards describe how pull like interactions can benefit from the concepts. Finally we will briefly describe some application areas.

2 Context Management and Situational Awareness

The context- / situational awareness service features concepts for integrating contextual information into user processes. Aligning to a widely adopted definition [ADB01], we assume context to be any information that can be used to characterize the situation of a person, a computing device, or a software agent. Instances of information covered by this definition are distinct locations, capabilities and services offered or sought, or activities and tasks in which services are utilized.

As opposed to other architectural approaches to context management (widget-based or blackboard, see [Wi01]), the core component, the so called *Context Engine*, is implemented as a service offering a dedicated interface to process the acquired raw data and making it accessible to other services and applications through various interfaces. Such service-oriented wrapping results in the key advantage that the underlying context-sensing (i.e., tracking) technology can be easily exchanged without affecting the rest of the system (e.g., the UI) or – even worse – the process.

The Context Engine contains means to dynamically register any networked legacy web service as context sensing systems (context producers); through complying with a prescribed *context provider adapter* (CPA) these sensing systems can be easily changed without affecting the business logic the engine is part of. At the other side, *context consumers* register for contextual changes.

A high-level of customization is achieved by implementing the processing (transformations, adaptations, interpretation, reasoning) of context data through a rule engine which works on an *editable hierarchical rule set*. The rules can be defined in a domain-specific language and the data base managing the rule sets follows a specific schema. In principal, the following types of rules have to be distinguished [EBB07]:

[1] The implementation of the device function delivering context can be quite complex going beyond the scope of a simple store and forward paradigm as featured by typical sensors. Actually the context may be constituted by instantaneous configuration of a finite state machine. Due to space constraints and the focus of this paper we not further elaborate this aspect.

1. *System rules*: these are kind of default rules applied to context data entering or leaving the engine. These rules can be only changed through the administrator and refer e.g. to the management of the context cache (management of cache size, data extrusion strategies etc.)
2. *Producer rules*: these rules refer to the production of context data. Informally rules of this type can look like this: *if an object moves from one location to another, remove location data and transform & store new data set*.
3. *Consumer rules*: potential consumers of context information provide the format in which they expect to receive the context data (Required Context Syntax) and subscribe for context through the according rules.

These rules are persistently stored in a database which is administrated through a thin object-relational mapping layer. Due to space constraints, we cannot extensively to other approaches to context management. However, the table below compares the Context Engine with other systems for context management (AURA [GSS02], CARMEN [BCM03], COOLTOWN [DGV03], GAIA [RHC02], MOBIPADS [CC03], CORTEX [SMT04]) in terms of key criteria for devising context-enhanced UIs (i.e., supporting reflection, composition of contextual information, migration and exchange of context data among different instances of the middleware in a networked environment and the support of configuration of the system through rules).

	Environment	Reflection	Composition / Linking	Migration	Rules
Aura	Infra-structure		×	×	
CARMEN	Infra-structure		×	×	
Cooltown	Infra-structure		×		
Gaia	Infra-structure		×	×	
MobiPADS	Self-contained	×	×	×	
CORTEX	Service-oriented	×	×		
Context Engine	Service-oriented	×	×		×

Table 1: Comparison of Context Engine with different approaches to context-management.

3 Push Interactions for Smart Browsing

By means of the Context Engine and the browser running on the user terminal we can now devise a concept for smart interaction with the devices in the user's ambience via their comprised services. We distinguish two different browser modes:

1. Service *pull*: this is the normal mode operation. The browser explicitly requests device services.
2. Service *push*: in this (non-standard) browser mode data is pushed to the browser. Subscriptions are handled by tuning the context-delivering middleware described above.

We will later on describe how to enhance ordinary (pull) browser mode. For the push mode, the Context Engine needs to be configured by registering the target device as a context producer. For using the context data produced by the target device at the terminal side, suitable subscriptions and actions in terms of rules have to be configured for the context consumer, i.e., the browser UI. In the actual prototype implementation we have foreseen a web interface to access the configurator of the Context Engine.

A push interaction is emulated by collecting the context data produced by the target device and letting the Context Engine react upon the data and the previously configured settings in the rule manager. The pushed data is delivered to the browser via a plugin as indicated in Figure 1. In the concrete implementation we have extended a browser control by a listener which then triggers the rendering engine for loading and displaying data (e.g., html pages). It has to be mentioned that pull mode is not necessarily disabled, but could be e.g. for the duration of an activity or transaction through according tuning the Context Engine accordingly. The logical communication (dashed line) is thus deflected to the Context Engine where the subscriptions of the consumers and the transformations to match the requirements of the consumers are handled.

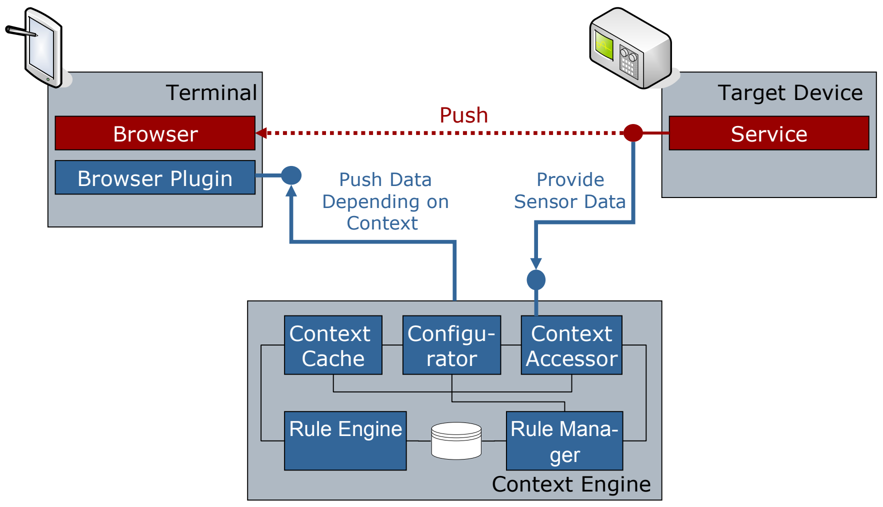

Figure 2: Realization of Service Push via browser plug-in and Context Engine.

Concerning the deployment of the Context Engine, different scenarios can be envisaged. For instance, the Context Engine can be run on the user terminal or on the target device in case the referring device is capable enough to run it. Alternatively a dedicated node running the Context Engine can be allocated. The producers and consumers will then have to use the service interfaces offered.

4 Request / Response Adaptations in Pull Interactions

Through the mechanism described in section 3 the browser and hence the application is relieved from processing data (i.e., context) constraining the processes the application is intended to implement. However, even for *pull* interactions there is a need for improved flexibility and adaptability with respect to the issued service requests and responses. For doing so two exigent problems need to be resolved:

1. There are tremendous *interoperability problems* when trying to couple a browser UI with services running on ambient devices.
2. Due to technical limitations pull interactions are rather *static* and thus close to *unchangeable*.

Contextual data may help in this regard, since it expands the conversational bandwidth and enables humans and technical systems to use implicit, additional information to comprehend and react upon transmitted data in a better way. Contextual information can also enhance the dynamicity of current pull-style interactions in browsers and thus help to overcome the above mentioned interoperability problems.

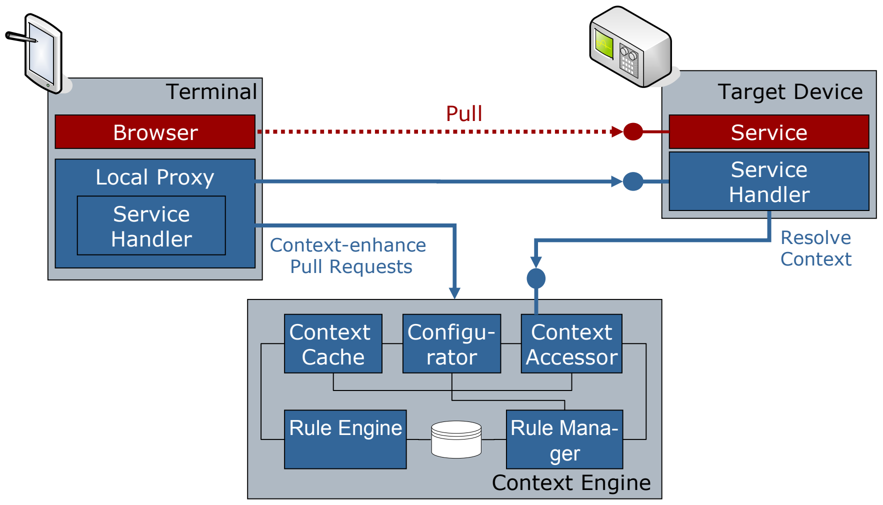

Figure 2: Adaption of Service Pull via Context Engine and handlers.

In this regard we offer a mechanism that transparently enhances service calls by context information. I.e., the context can be delivered and extracted in a way which is transparent to the involved peers (i.e., UI and back-end services), adaptable at runtime and permits the transparent provisioning and consumption of context.

The general principle of the envisaged solution as depicted in Figure 2 follows the practice to *separate the different concerns* (acquisition / storage on one side, querying context on the other). This is captured by the following two parts.

1. The first part is given by an *instantiation of the Context Engine*. Through this concurrent and remote access to contextual data is granted. The Context Engine acts as the central point for context retrieval requests, context processing and distribution. Since the Context Engine is largely configurable, such approach ensures the maximum flexibility concerning context handling.
2. The advancement of this work over the state of the art is given by the second part, a *handler mechanism* deployed to the browser that post- (respectively pre-) processes requests and responses to acquire and adjust context from the Context Engine during the service invocation.

In an implementation of the above concept using the Axis2 service container we have devised an XML format to be embedded into the request / response messages to tag the intended context-dependent adjustments to the request / response. This yields significant flexibility when compared to other approaches [BDR07] or [KSJ04] which focus on a fixed set of context elements (location , presence,…) and thus lack extensibility aside from flexible use. Even the context plug-in solution for context-enhanced service calls described in [KK04] cannot compare to the solution here, since it misses runtime adaptability and efficiency (several plug-ins will have to be activated simultaneously).

5 Application Areas

5.1 Manufacturing Industry

In the domain of industrial applications we have featured the framework in a Field Service Management (FSM) solution [EM08]. FSM is a means to optimize processes and information defined by large manufacturing companies who send technicians or staff out of the office / into the field in order to fix problems regarding products on behalf of the respective company. In the concrete example, FSM allows to dispatch requests for resolving vehicle issues to field workers issued by the service division of a car manufacturer at a dealership location. The field engineer uses a mobile terminal running an application to search and download all the relevant information concerning the site where the alert was raised (e.g., relevant service bulletins, parts information, and dealer profile information). The use of the application framework in this scenario is such, that both interaction modes are supported handled: issues can be *pushed* to browser UI of the field engineer; in *pull* mode, requests for receiving issues can be adjusted to the field engineer's actual context (location, preferences).

5.2 Health Care

In a similar way, applications in the health care domain can be supported. In a concrete scenario the care taking for outpatients by health professionals (phyisicians, nurses,…) was studied. Very much alike the FSM solution, the health professionals are scattered over a certain area. They are equipped with mobile terminals supporting the execution of the patient-related processes. Once an emergency situation occurs, the according information pops up on the health professional's terminal. Information is *pushed* depending on the context of the health professional's working status (busy, on the move, waiting), actual schedule and the emergency level of the request.

5.3 Transportation

The ever-increasing volume of air passengers and complexity of airports have reached a point where it affects all passengers, irrespective of travel class. We consider a scenario supporting *people moving*, covering the seamless fulfillment of passenger tracing, and provisioning of guidance services and other useful information via stationary and mobile solutions. Here active tokens (implemented as extensions to frequent flyer cards) and passive tokens (ordinary paper-based tickets containing UHF RFID tags) are used for tracking the passenger. Information *push* to the passenger can be implemented in two ways: either by delivering information to the mobile terminal under disposal of the passenger or to a stationary terminal being in the user's vicinity. *Pull* requests for supporting guidance can be enhanced to deliver information on the user's location without requiring the user to explicitly enter this data.

6 Conclusions

We have described a browser-based application framework to ease the development of applications accessing device services in the user's ambience. The idea is to enhance the well-known browser paradigm to incorporate situational awareness features in order to reduce user intervention to the bare limits. We propose a middleware that can be plugged into the communication between UI and the services in order to tune ordinary *pull* and additionally permit *push* interactions. Such requirement will certainly gain importance in the Internet of Things where the user will be confronted with enormous amount of smart gadgets.

Due to space constraints we could not fully dig into the versatile usage of the framework. Some samples explain the usage of the framework for real-world and present-day applications in industry, health care and transportation. In future developments it will have to be studied, how context information can be used to secure interactions between communicating peers particularly in push mode and how user defined policies can be used to avoid annoying push data delivery in pull mode.

References

[ADB01] Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., & Steggles, P. (2001). Towards a Better Understanding of Context and Context-Awareness. Lecture notes in computer science: Vol. 1707. Handheld and ubiquitous computing. Proceedings (pp. 304–307). Berlin: Springer.

[BCM03] P. Bellavista, A. Corradi, R. Montanari, C. Stefanelli. Context-aware Middleware for Resource Management in the Wireless Internet. IEEE Transactions on Software Engineering, Vol. 29, No. 13, 2003.

[BDR07] Baldauf, M., Dustdar, S., & Rosenberg, F. (2007). A survey on context aware systems. IJAHUC, 2(4), 263–277.

[CC03] A. T. S. Chan and S.-N. Chuang. MobiPADS: a reflective middleware for context-aware mobile computing. IEEE 2003.

[DGV03] Philippe Debaty, Patrick Goddi, Alex Vorbau. Integrating the Physical World with the Web to Enable Context-Enhanced Services. Hewlett-Packard Technical Report, HPL-2003-192 20030912.

[EBB07] Eikerling, H.-J., Benesch, M., and Berger, F. (2007). Using Proximity Relations for the Adaptation of Mobile Field Services. In International Workshop on the Engineering of Software Services for Pervasive Environments (ESSPE '07) at ESEC/FSE 2007, September 4, 2007, Dubrovnik, Croatia.

[EM08] Eikerling, H.-J., & Mazzoleni, P. A methodology for the Design, Development and Validation of Adaptive and Context-aware Mobile Services. In: *Context-Aware Mobile and Ubiquitous Computing for Enhanced Usability* (Ed.: Stojanovic), Idea Group Inc (IGI), 2009.

[GSS02] Garlan, D., Siewiorek, D., Smailagic, A., Steenkiste, P: Project AURA: Toward distraction-free pervasive computing. IEEE Pervasive computing (2002) 22-31 4.

[KK04] Keidl, M., & Kemper, A. (2004). Towards context-aware adaptable web services. The thirteenth International World Wide Web Conference. Alternate track papers & posters (pp. 55–65). New York: Association for Computing Machinery.

[KSJ04] Kerer, C., Schahram, D., Jazayeri, M., Szego, A., Gomes, D., & Caja, J. A. B. (2004). Presence-Aware Infrastructure using Web services and RFID technologies. Proceedings of the 2nd European Workshop on Object Orientation and Web Services. Oslo, Norway.

[SMT04] Carl-Fredrik Sørensen, Maomao Wu, Thirunavukkarasu Sivaharan, Gordon S. Blair, Paul Okanda, Adrian Friday, Hector A. Duran-Limon: A context-aware middleware for applications in mobile Ad Hoc environments. Middleware for Pervasive and Ad-hoc Computing 2004: 107-110.

[RHC02] Manuel Roman, Christopher K. Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. Gaia: A middleware infrastructure to enable active spaces. IEEE Pervasive Computing, 1(4):74--83, October-December 2002.

[Wi01] Winograd, T. (2001). Architectures for Context. Human-Computer-Interaction, Special Issue on Context-Aware Computing, vol. 16(2-4).