WIRELESS WORLD
RESEARCH FORUM

The Simplicity Architectural Concept

C. Noda[(1)], W. Kellerer[(1)], E. Rukzio[(2)], H. Hussmann[(2)], N. Blefari Melazzi[(3)], S. Salsano[(3)], R. Seidl[(4)]

(1) DoCoMo Communications Laboratories Europe, (2) Media Informatics Group, University of Munich, (3) DIE, University of Rome, (4) Siemens AG

***Abstract*—Beyond 3G mobile systems are characterized by the usage of a multitude of different terminals, from high-end devices to miniaturized ubiquitous devices, as well as heterogeneous networks. Complexity for configuration and service provisioning is increased on both terminal and network sides. To ease of the users' burden, flexible adaptation and personalization support are fundamental requirements for service architecture. In this paper, we present the Simplicity architecture based on the concepts of a Simplicity Device for storing a user's profile, and a policy-driven brokerage framework. We further discuss a user's personal assistant (i.e., a software agent), which acts proactively to simplify user interactions with the system.**

***Index Terms*—agent, brokerage framework, personalization, service architecture,**

INTRODUCTION

THE Simplicity project is a European Union research project, performed for two years (January 2004 – December 2005). The consortium consists of 11 partners from both academia and industry. Simplicity stands for Secure, Internet-able, Mobile Platforms Leading Citizens Towards simplicitY [1].

Our main goal is to simplify the process of services usage by autonomic service configuration based on user preferences and user-friendly interfaces. More specifically, we design architecture based on a brokerage framework, which enables:

- personalization of services according to user preferences and the current status,
- adaptation of services to available network/terminal resources and service support technologies, and
- seamless portability of services, applications, and sessions across heterogeneous terminals.

We introduce a 'Simplicity Device (SD)' as a user's personal device, which stores user profile information. The user profile stored on the SD shall be available and accessible globally. The user profile is for personalization and automatic adaptation, e.g. by specifying QoS and price preferences for automatic network configuration, or by defining a preferable user interface for transparent usage on different terminals. The profile is managed proactively by a user's personal assistant (i.e., a software agent), a Simplicity Personal Assistant (SPA), on behalf of the user. The SD can be implemented on a physical device such as a smart card, a USB memory, or a Java ring. It is plugged into the chosen terminal or connected to via short-range wireless links (e.g., Bluetooth, WLAN, or NFC). Alternatively, the SD can be virtual, i.e., based on a pure software solution implemented on a network infrastructure. In this case, a physical SD may still be required storing a pointer to the network location of the profile.

The SD provides necessary information to personalize and adapt services, and support seamless services across terminals. The SD interacts with terminals directly via physical interfaces, and with networks virtually through the terminals, for configuration of network access and automatic service selection. To achieve flexible interactions supporting reconfigurability, we rely on a brokerage framework driven by policies. Our brokerage framework enables orchestrating functionality and hiding complexity of systems by functionality encapsulation.

Functionality is allocated to different entities according to physical capabilities, accessibility, or security (including privacy) requirements. It also allows network/service providers to deploy new services in a short term by adding new functionality or updating existing. There are two types of policies for autonomic decisions toward adaptations. One specifies a set of rules for intra- and inter-communication of brokers. Another is applied for higher service/application level management (e.g., QoS, security).

In this paper, we introduce one scenario for a mobile worker, and further discuss the architecture based on a brokerage framework in more details. Then, two key enablers for adaptation and personalization, the policy framework and the SPA, are described.

Mobile worker scenario

When Anthony arrives at his office, where the 'moving work place project' is introduced, his SPA on the PDA discovers available services such as a work place assistant service and a navigation service. It collaborates with these services to offer a workstation according to his preferences (e.g., large display, high speed network access). When Anthony sits at the workstation desk, the SPA automatically configures the workstation (e.g., high speed network access connection, data storage, printer, personalized user interface) with requesting only the user authentication. After a few hours of work, he has to leave for a business trip. The active session and the status of active applications are stored on the SD. One hour later, he starts to use his laptop in a train. Again the SPA automatically configures network access, with middle speed and low price. The suspended session and the applications are resumed.

This application scenario was used to identify functional requirements, and has been selected for one of our prototypes. More detailed application scenarios can be found in [2], [3]. The following cases are identified.

- Different users share the same laptop, but see their own personalized working environments.
- One user who uses different terminals sees his personalized working environment.
- Users suspend and resume running sessions and applications by plugging and unplugging the SD.

Architecture

Broker architecture

The Simplicity architecture foresees a number of software and hardware entities that are part of the "Simplicity system" and collectively provide Simplicity services to users. The Simplicity system interacts with other "external" elements, such as user terminals, applications running on user terminals, network elements, servers, network services and so on. An overall picture of the Simplicity system is represented in Figure 1. The main components of the Simplicity system are the SD, the Terminal Brokers, the SPA, and the Network Brokers. The interaction of the Simplicity system with existing ("legacy") application and services is depicted, as well as the interaction of the system with external applications that are designed to exploit the capability of the system (denoted as "Simplicity enabled 3rd party applications").

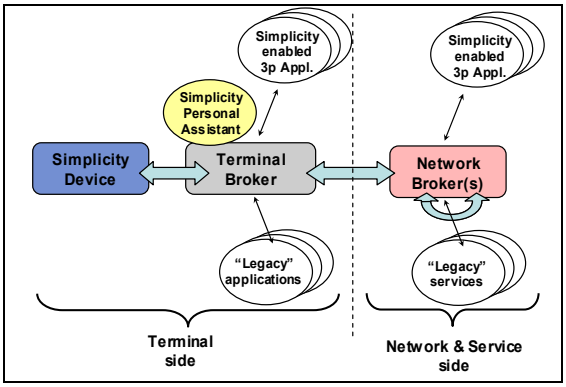

Figure 1: The Simplicity system components

The role of the SD is to store user's profiles, preferences and policies. It also stores and

allows the enforcement of user-personalized mechanisms to exploit service fruition, to drive automatic adaptation to terminal capabilities, and to facilitate service adaptation to various network technologies and related capabilities.

The Terminal Brokers manage the interaction between the information stored in the SD and the terminal in which the SD is plugged in. The Terminal Brokers enable the SD to perform actions like terminal capability discovery, adaptation to networking capabilities and to the environment, service discovery and usage, adaptation of services to terminal features and capabilities. The Terminal Brokers cater also for the user interaction with the overall Simplicity system (including network technologies and capabilities).

The SPA represents the interface of the Simplicity systems towards the end-user. The SPA interacts with users via a convenient user interface, assisting users towards completing their tasks. It acts autonomously whenever it can, requiring only minimal input from the user. The SPA is involved in many tasks, which include user authentication, management of user's preferences and also application related functionalities like session management, service subscription, adaptation (personalization) and invocation. We discuss further details on design of the SPA later.

The Network Brokers have the goal to provide support for service advertisement, discovery and adaptation. Moreover, they orchestrate service operation among distributed networked objects, taking into account issues related to the simultaneous access of several users to the same resources, services, and locations. They also share/allocate available resources, and manage value-added networking functionality, such as service level differentiation and QoS, location awareness, and mobility support.

3rd Party Applications run on the user terminal and on other network-side entities. 3rd Party Applications use features provided by the Simplicity system through a specific interface, called Simplicity Applications Interface (SAI).

The interfaces between the identified entities (see Figure 1) must be clearly defined. In particular, three fundamental interfaces have been addressed: 1) the interface among the brokers, which will be called Simplicity Broker Communication (SBC); 2) the interface between the brokers and the external applications, called SAI; 3) the interfaces between the Terminal Broker and the Simplicity Device, called SD Access Interface (SDAI). Instead of presenting the detailed specification of these interfaces, we rather discuss the decomposition of the architecture in "subsystems" showing which subsystem takes care of the identified interfaces. In addition to the interfaces, we need to specify the representation of the user profile information, which provides a common underlying information model for all the elements of the Simplicity architecture. This representation has been called "Simplicity User Profile" (SUP), extending the "Generic User Profile" by 3GPP [4]. The full XML definition of the SUP is included in [5].

In order to achieve a flexible and modular specification, Terminal Brokers and Network Brokers have been de-composed to a set of separate logical components, i.e., subsystems that implement the required functions. Subsystems are attached to the mediator, which handles intra-communication on the broker. Reusable subsystems implement common Simplicity functions, while specific subsystems may be defined to implement specific applications in the Simplicity system. The interaction between subsystems is defined in terms of asynchronous events exchange (specified using UML class and sequence diagrams).

The inter-communications between subsystems that are physically located in different brokers constitute the SBC. The SBC specifies how the brokers talk each other in the Simplicity system. Each broker includes a dedicated subsystem that implements the SBC.

The defined subsystems are:

- SBC – Simplicity Broker Communication
- SAIM – Simplicity Applications Interface Manager

- SDAM – Simplicity Device Access Manager
- Profile Management
- Capability Management
- Policy Management
- Policy Decision Point
- Service Management
- Presence
- User contracts & pricing
- Access Network
- SDS-c - Secure Distributed Storage client
- Application specific subsystems

Figure 2 provides a graphical representation of the Simplicity detailed architecture, which shows the subsystems and their relation to the other defined entities.

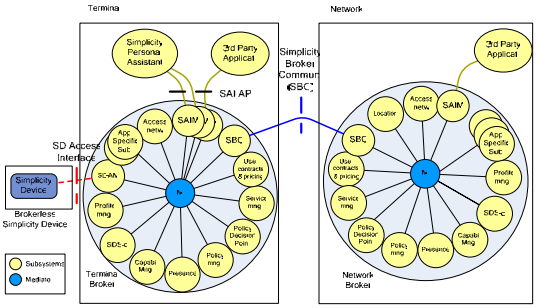

Figure 2: Overall picture of Simplicity detailed architecture

Adaptation and Personalization

As already mentioned, adaptation and personalization of services, applications and devices are core concepts of the Simplicity architecture. We discuss two key enablers, the policy framework and the SPA, and explain how they work in the broker architecture. One concrete example for adaptation done by the SPA is further presented.

Policy framework

Context information (e.g. user preferences, device capabilities, available services and sensor data) is the basis for adaptation and personalization. Furthermore there are different parties like the user, the service provider or the network provider involved in specifying their preferences and rules regarding the different adaptations.

To address this issue we have integrated a policy-based architecture in the Simplicity architecture. Policies can be seen as conditional rules, which can start an adaptation process when a specific situation represented by context information arises. The advantage of this approach is that the policies allow creating a custom program for each situation and allow handling adaptations that the programmer might not have considered. Such a program is realized as a subsystem in the Simplicity architecture, and is called policy decision point. It can be comparable with a rule engine in the field of artificial intelligence.

The policy management subsystem is responsible for storing, updating and accessing policies. Both policy-related subsystems could be part of the terminal as well as the network. Figure 3 illustrates the relation between the different subsystems and their connection to the mediator.

If an arbitrary subsystem needs information for an adaptation process it sends a corresponding policy event *A,* which includes mostly also some context information, to the policy management subsystem. This subsystem looks up which policies are needed, and adds them to event *A* to be sent to the PDP. The PDP requests additional context information from other subsystems (e.g., context management subsystem, capability management subsystem, profile management subsystem) that are needed for the decision process. After the execution of the policies, the PDP sends the results back to the subsystem, which asked for the information needed for an adaptation process.

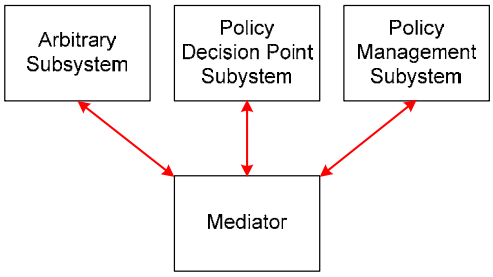

Figure 3: Policy framework

Because of the permanent changes of the context information, the involved parties and the desired adaptations the described policy-based framework has to be very flexible. We achieve flexibility through the separation of context information, the policies, and the adapted services, which makes it easier to modify policies and to integrate new adaptations. Furthermore the policies are not designed for a specific kind of adaptation, which allows the integration of arbitrary adaptations or decisions requested by different subsystems.

Some of important problems in the field of policy-based adaptive systems are distribution, the definition of the policies by the developer, conflict detection, complexity, and performance. We address these problems through the usage of a defined process for policies and through a module-based policy pipeline.

Design of the SPA

The SPA acts proactively on behalf of the user to achieve full personalization of user interfaces, services, and applications running on the TB. It is a virtual representation of the user in the Simplicity system and resides on the TB. The SPA provides homogeneous personalized access to the system and services through user-friendly interfaces. The integration of the SPA on the TB is depicted in Figure 4.

Core functionalities of the SPA are:

- login for user authentication
- service subscription and invocation
- management of profile information
- adaptation and personalization, and
- session management.

When the SD is plugged into a TB, the SPA provides a uniformed login display for the user. After successful login, the SPA starts to interact with other subsystems through the Simplicity Application Interface Manager (SAIM). It retrieves user context information form different subsystems, e.g., profile information from the profile management subsystem, user status information from the context management subsystem, and policies from the policy management subsystem. The SPA sends a request for a list of available services to the SAIM with setting a set of service attributes, i.e., a subset of retrieved contextual information. After getting the list, it sends back the selected services list, for subscription to the SAIM. This happens whether i) the SPA has selected, according to locally available information on the TB or the SD, such as user profiles and contexts information; or ii) the user has selected from the pre-selected services on the SPA, through the user interface. The SPA also provides the information required for subscription, such as a user identity and a user credential. Finally the SPA stores contract and access information as a part of the user profile for subsequent use.

The service invocation function of the SPA is in charge of invoking services, by using access information for configuration. The SPA activates a service, using access information stored in the profile management subsystem on the TB or the SD. It acts autonomously, when it receives a trigger, e.g., through the user interface, or a notification from the NB.

The SPA can adapt 3rd party applications, such as a web browser or a media player, directly, by calling their adaptation methods. The SAIM acts as a proxy, and provides retrieved data to the SPA. The SPA can adapt and personalize the context information, which goes through the SAIM proxy. It enables, for example, automatic form filling, which we discuss in the following section, or the automatic login process. The latter can be achieved by modifying the HTML-code that goes through the SAIM.

The SPA gets the decisions that are needed for the different adaptations from the

policy framework. Therefore the SPA sends a corresponding event to the policy management subsystem. This subsystem collects the different policies and sends them together with the first event sent from the SPA to the PDP. The PDP processes the event, context information and policies. Afterwards the PDP sends the result back to the SPA which uses it for the concrete adaptation.

The SPA stores the session status information at an applicable timing, a certain interval or when an event has occurred (e.g. session suspending). The information is stored in the profile management subsystem on the SD. The SPA can resume the suspended session, using the latest session status information. When the user logs into the terminal, the SPA checks for suspended service sessions, by searching session status information, and resumes if required.

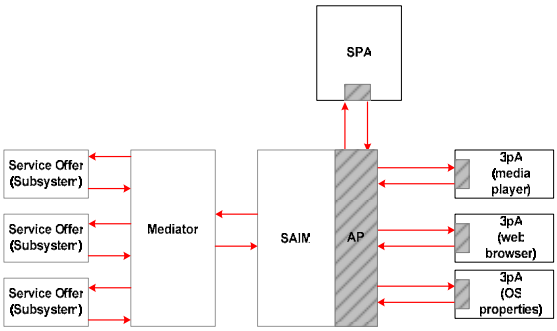

Figure 4: Integration of the SPA into the terminal broker.

Automatic form filling by the SPA

Here we focus on a concrete example for an adaptation that is done by the SPA on a mobile phone. In a study [6] we found out that mobile services (i-mode, WAP, etc.) often do not provide forms but only very basic login input forms. For instance we found a mobile service of a German bookshop where after the user selected a book for buying, this company called the user instead of having the user type in his name, address, phone number, email address and bank account. It is because typing in texts on a mobile phone is a very complicated and often also a time consuming task for a lot of people. We see this as an obstacle for the usage of a lot of mobile commerce services.

Therefore the idea of automatic form filling on mobile devices seems to be on solution for this kind of a problem. The SD may include all the user data already mentioned for the forms of most buying or booking processes.

Regarding the implementation we considered three different cases, an XForms solution [7], a modification of existing mobile services, and the support of the already existing mobile services. We chose the last one which is the most complicated on the one hand, but on the other hand compatible with all existing mobile services, which may improve the acceptance of such an intelligent personal assistant, the SPA.

Our form filling algorithm is based on a concept developed by Chusho et al [8]. The basic idea is to analyze a huge set of existing web pages and to figure out rules like “If ‘name’ is written left of a field then a probability of X percent exists that the field should filled out with the name of the person”. This algorithm takes different aspects of web pages into account, such as the attribute name of the input field, the label of the field before or after, and the data type of the field.

We adapted this concept and integrated it in the Simplicity architecture as shown in Figure 5.

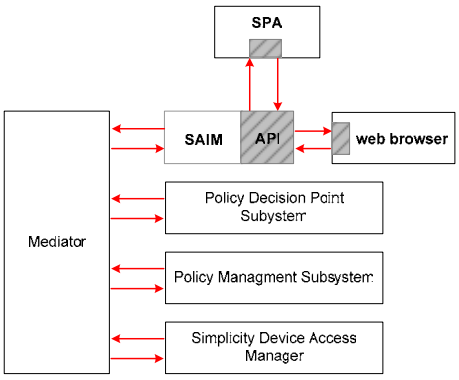

Figure 5: Components needed for automatic form filling

The SPA is the component that controls the whole process. The SAIM acts as a proxy that transfers data between the mediator and the web browser on the mobile phone. The SPA can access the transmitted data through an integrated parser for web pages (e.g., WAP or i-mode). After the SPA has recognized a transmitted form, it sends this information as an event to the policy management subsystem that selects the needed policies, and sends afterwards all information to the policy decision point. This subsystem requests the needed user data from the Simplicity device access manager and processes all polices. The corresponding result is sent to the SPA which changes the transmitted webpage. Through this, the users sees in his web browser an already filled form.

Like most context-aware adaptive systems we also have to take the uncertainties into account that exist when reasoning about context and the usage of probabilities. We address this issue through the visualization of uncertainty to the user. If a field is filled out with a probability of 90%, this field has a green background and a worse probability leads to a red background. Through this the user is aware of the uncertainty and checks red fields better than green fields.

Furthermore the policies get updated, if for instance a field of a specific web page cannot be filled out and several people had to fill this field by themselves. Such a behavior is recognized by the SPA because also the data which is sent from the browser of the mobile phone to the server goes through the SAIM. This information is used to update the policies in the policy management subsystem.

Conclusion

In our prototype, the SPA is implemented on each TB. However the SPA can be a mobile agent, which migrates to other entities for service execution on demand and returns with an achieved result. We assume two interesting cases:

- the SPA is a mobile agent resides on the SD, and be carried by the user, or
- the SPA is a mobile agent residing on one of the NBs that are always accessible form TBs, and migrates to the TB when the corresponding SD is plugged in.

Both of the cases have advantages of security. Instead of relying on SPAs provided by different TBs, which the user may share with other users, the user can own a trusted SPA on her/his own TB, or download from a trusted NB. One solution for the latter case is discussed in [9]. So far in our implementation, only user authentication aspect is taken into account, but other security and privacy issues such as trustworthiness of TBs (e.g., how we make it sure that all data retrieved by the SPA is deleted when the SD is plugged off) are still open.

The Simplicity architecture is designed to ease user's burden in the heterogeneous environment. To achieve a configurable system, broker architecture and a user profile stored on the SD are as basis. For autonomous adaptation and personalization, two key enablers were presented: policies to solve inconsistent problems between different parties, and the SPA to act on behalf of users with providing personalized user interfaces and managing the profile on the SD.

REFERENCES

[1] IST Simplicity project: http://www.ist-simplicity.org
[2] N. Blefari Melazzi et al, “The Simplicity Project: Managing Complexity in a Diverse ICT World”, in Ambient Intelligence, IOS Press.
[3] R. Seidl et al, “User scenarios for simplified communication spaces”, IST Mobile & Wireless Communications Summit 2004, June 2004, Lyon, France.
[4] 3rd Generation Partnership Project. Data Description Method (DDM) - 3GPP Generic User Profile (GUP). Technical specification of Technical Specification Group Terminals, Version 6.1.0. 2004. Reference example
[5] Simplicity Deliverable D2202: “Final system architecture specification”; http://server.ist-simplicity.org/deliverables.php
[6] E. Rukzio et al, “Privacy-enhanced Intelligent Automatic Form Filling for Context-aware Services on Mobile Devices”, Workshop Artificial Intelligence in Mobile Systems 2004 (AIMS 2004), Nottingham, UK, September 7 2004.
[7] XForms, http://www.w3.org/MarkUp/Forms/
[8] T. Chusho et al, “Automatic Filling in a Form by an Agent for Web Applications”, Asia-Pacific Software Engineering Conference 2002, IEEE Computer Society, pp.239-247, 2002.
[9] C. Noda, T. Walter, “Smart Devices for Next Generation Mobile Services”, Construction and Analysis of Safe, Secure, and Interoperable Smart Devices workshop (CASSIS), March 2004, Marseille, France