Service Creation for End-Users

Olaf Droegehorn, Sian Lun Lau, Niklas Klein, Immanuel Koenig
University of Kassel, Germany
Chair for Communication Technology
Email:droegehorn@uni-kassel.de

Abstract—The current computing and communication services are driven by various technologies. The creation of such services is however not an easy task. The different platforms lack openness that simplifies interoperability. Service development is still mainly technical oriented, where service creation tools are meant for serving and assisting the professional developers. Service creation is not seen as a task for end users right now.

In this paper we will discuss how end users could be enabled to create new services, shown on the example within a mobile context-aware service platform. This is achieved by introducing the concept of the technology agnostic approach into the process. We present the conceptual architecture and explain the idea of end-user driven service creation that can provide a solution. Several tools were designed and implemented to prove that the concepts work. With the help of semantic service description, the end users can express their service wishes and create services using a graphical interface technology agnostically. The idea was implemented in two European projects and we have identified the next steps to improve the prototypes as well as the idea.

Based on several tools and an underlying platform the ideas have been tested in a development environment for mobile context-aware services. Here end-users are able to design their own services or combine existing services, which can be shared and used by other users later on.

Index Terms—Service Creation, Technology Agnostic, Semantic Service Description, Service Execution Environment, Rule-based approach, MUPE platform, Mobile context aware services.

I. INTRODUCTION

The existing computing and communication technologies brings us a wide variety of services. These services are typically created to bring convenience and simplicity to end-users. However, the diversity of the technologies makes the service creation process a complicated task. The plurality of services demands more decisions and more know-how from anyone who wishes to contribute in this process. The current development of services brings up several issues to our attention. There are various device platforms, both software and hardware, that are unique to each other. The devices and operating systems are designed in such a way, that services can only be deployed on a specific group of devices. This makes service creation and deployment a resource demanding and complex process, since a service needs to be created for each group of devices that has specific requirements and specifications. The development of the same service for every potential platform requires at least recompilation of the service for each single designated platform. In some cases, a service might even need to be rewritten.

Furthermore, the current inter-service communication shares also the same closed nature. For example, to exchange information between a messaging service and a calendar service, specific Application Programming Interfaces (APIs) need to be predefined and used. Each service is a single self-containing entity. Different services from respective providers will only work within their own defined areas and limits, unless the needed interfaces are defined and provided during the design time.

An end-user has no influence in the service development phase. His role is merely a consumer of a service – in other words, he should use what is given to him. End-users are seen as non-developers. A professional developer is defined as a programmer that has the necessary technical know-how to create services, for instance, by utilizing source code and software libraries. The complexity and technical know-how requirements hinder the end-users from having a role in service creation. In our vision End-users can make own decisions in creating, obtaining and using their desired services. The complexity and technicality should be hidden, if not removed, so that the end-users can get involved in the service creation process.

This vision demands new ideas and approaches that will bring us beyond current environments. One can create services provided by service providers using intuitive and easy tools. The complexity should be hidden without compromising the quality of the provided functionalities. Issues such as the heterogeneity of networks and the support for different device platforms should be taken into account, so that users do not need to be bothered about most technical-related problems.

In the next section we will first present some current service creation approaches. Section II will explain the conceptual architecture of our approach. This is followed by section III, where an implementation of the architecture will be elaborated. An end user focused experiment and its results are presented in Section IV. Section V will conclude this paper.

II. CONCEPTUAL ARCHITECTURE

The main challenge of the idea is to solve the issues of
technicality and complexity in service creation. If one takes a
detailed look into the process, services are created with a goal
in mind – to provide functionalities to their users. Normally
one finds a composition of functionalities in a given service.
Therefore, we make this assumption - when a user can
compose needed functionalities without the technical know-
how, he can “create” a service, too. If functionalities can be
extracted from a service and be made into independent
entities, service creation can be achieved by connecting them
meaningfully.

With this idea, few definitions have been proposed.
Firstly, in order to differentiate services created by
professional developers from the services created by normal
users, the former will be defined as Service Building Blocks
(SBBs). An SBB can either be referred to entities commonly
known as services, or certain functionalities offered by a
conventional service. For example, a conventional service that
offers location based services can provide GPS coordinates of
locations and calculate distances between locations. Each of
these functionalities will be a single SBB in the designated
platform. This definition has similarity to the use of enablers
specified by the Open Mobile Alliance (OMA) [3].

The term “Service” will be used for the composition of
several SBBs. As mentioned earlier, from a user's perspective,
services are commonly goal-oriented and are designed to fulfil
the potential users' need. When he identifies a certain goal, he
can put together the needed SBBs to form a service. A service
can be seen as a collection of different functionalities that
aims to provide solutions to the end-users' needs.

Using the above definitions, the services created by this
are not necessarily technically demanding. A user's need is
goal-oriented. He wishes to accomplish a certain task with a
given service. As an example, a user wishes to inform his wife
that he is coming home soon after work. In this example,
communication – the delivery of the message - is the primary
task. Other decisions such as the choice of communication
channel, type of devices or type of message medium are
comparatively secondary. Whether the service delivers the
message as an SMS, or converts it into a voice message, the
goal is considered to be achieved when the recipient received
the message.

Therefore we introduce the concept of technology
agnostic approach into service creation. Users should not be
overwhelmed by the technicality of the process. Instead, the
provided environment should assist users to create services
without the need of technical know-how. This requirement
brings us a challenge – the system should provide the
abstraction capability in hiding and translating the technical
information into technology agnostic descriptions, which will
be used by the user to create services. With this, end users can
start creating services that answer to their needs while the
underlying service execution environment (SEE) will take
care of the actual execution and invocation.

The following conceptual architecture illustrates our idea of
the solution. The three layer architecture describes the
separation between different roles in the service creation
process, and how selected technologies and tools can be
applied to realise it.

Layer 1 is where the SBBs are developed and deployed.
As mentioned earlier, a SBB has clear definitions of its
functionalities and features. Ready-to-use SBBs will be
executed on an SEE. The integration of SBBs for a given
SEE should be simplified, so that professional developers do not
need to deliberately learn to adapt to a specific SEE.
Therefore development tools can be provided to assist easy
SBB creation. For example, development plug-ins can be
provided for an integrated development environment (IDE) to
simplify the SBB development and deployment process. In
this way, a developer can focus on important issues related to
functionalities and features of a SBB.

Layer 2 provides the support of the semantic service
description and semantic service discovery. The usage of
semantic service description enables the system to
“understand” the meaning of SBB functionalities. It describes
SBBs with semantic relations. It is able to understand the
queries made for the required functionalities and hence will
look for the corresponding SBBs. With this layer, even though
the created services are defined without a direct link to or
knowledge of any of the offered SBBs, the SEE is able to
understand the services created in Layer 3 and will invoke the
needed SBBs as desired by the users.

The creation of services takes place in Layer 3. Suitable
service creation tools should be developed to assist end-users.
An important characteristic in this layer is the technology
agnostic service creation approach. In this approach, end-users
are expected to express their needs according to their own
understanding, without the technicalities. In other words,
users do not need to search and understand any technical
information related to the SEE or SBBs. The actual execution
of the service will take place in the SEE.

With the layered architecture, different parties can also
create tools for a specific layer, and expect possible
interoperability with tools from other layers. This promises
openness and flexibility for both users and developers

III. AN IMPLEMENTATION OF THE ARCHITECTURE

Based on the architecture mentioned above, several tools
were created to evaluate the viability and practicality of the
ideas. Our Java-based SEE [4] was used in our
implementation. The SEE supports service deployment and
life cycle management. To allow easy deployment for
developers, an SBB development plug-in for the Eclipse IDE
[5] was created. With this plug-in one can create a new SBB
or port an existing SBB for the chosen SEE. The developer can also define the semantic service descriptions for the developed SBB using the provided wizard dialogues. In this way, the SEE-specific requirements and routines can be assisted, simplified and automated with the plug-in. This will minimize the effort required for the SBB development for a given SEE.

For easy service creation (Layer 3), a Visual Service Editor (VSE) was developed to enable service creation using logic rules. Users express their service needs via “if-then” relationships. We call this technique “Draw a service” - the users basically draw services that answer their needs. The drawn logic expressions are graphical representations of high level abstraction of service behaviours. These abstractions do not necessarily have direct relations to any of the SBBs found in the SEE. The users can search for available SBBs in the SEE, described with high level abstraction descriptions, and compose services using these SBBs. They can also start composition free-hand – by drawing rule expressions with desired keywords without searching for available SBBs beforehand.

Imagine a scenario in which it is common that parents set rules for their children, for instance, the television can only be watched everyday in the evening and in the living room. Other restrictions, which might be important in the reality like, 2 times per week only and only for this or that type of film and only till 21:00 can easily be added in the same way. But the service should not give the same restriction to the parents. As shown in Figure 2, an example of a television control service is created using the VSE.

Since the service wishes are expressed in logic rule policies, we have chosen RuleML [6] as the representation language for the composed services. In this manner, any VSE that produces services expressed in RuleML (with supported language versions) will promise interoperability. This has been implemented within the S4ALL and the IST project SPICE [7].

The SEE will process the “drawn” services, produced by a user, and executes them. It bridges between end-users and the SBBs, allowing the desired functionalities to be invoked when desired. In our implementation, the inference of the produced services will be carried out by a component called Business Rule Evaluator (BRE). The BRE enables the collection of needed information that may trigger corresponding SBBs. A rule engine is used in the BRE to perform rule inference. If needed, rule language translations can be used to support other rule languages.

The semantic service discovery is a crucial component in our implementation. Since services are no longer created using technical and exact definitions, the discovery component must be able to process the given query to find the designated SBB using the defined semantic description. As seen in the example above, the parents have only stated non-technical terms such as time, location and television in the rules. The BRE finds and triggers the SBBs that offer corresponding information, and then evaluates the rules. As long as the SBBs are semantically well-defined, the best matched SBBs will be invoked. A combination of keyword matching and ontology inference will be a possible solution for such scenarios.

### IV. USER CREATED SERVICES

Another, slightly different approach is presented by Multi-User Publishing Environment (MUPE) [8, 9] by Nokia.

MUPE is an Open Source application platform for creating mobile multi-user context-aware services. This platform is based on client-server architecture and it can be used to create e.g. mobile games, virtual worlds, collaborating applications etc. MUPE is built entirely on top of JAVA (J2ME on client-side and J2SE on server-side). The use of JAVA as well as standard communication protocols within MUPE platform enables portability of the platform. Figure 3 presents the overview of the MUPE platform. End-users use MUPE client to connect their mobile devices to the MUPE server. MUPE server contains the core components, i.e. client manager, world manager and context manager. The actual service is implemented into the server as service logic. Different context
information can be added to the MUPE server through context manager.

The MUPE platform follows closely the concept of Technology Agnostic Service Creation. MUPE client, MUPE server, i.e. core elements, and different context producers are developed by professional developers (Service Building Blocks Development layer). MUPE uses a general client (installed once) for all services. As soon as the MUPE client has been installed it can be used for browsing new services provided by the MUPE server (Semantic Service Discovery and Service Description layer). The data transfer between MUPE client and server is performed as structured Extended Markup Language (XML) messages. Client acts much like a web browser providing user interface based on the XML descriptions received from the server. With this kind of an approach MUPE combines the best parts of stand alone and browser based applications. MUPE server consists of the connection middleware part and the service logic part. Connection middleware implements various communication means between server and the client and to the external context sources. Communication between middleware components is handled with TCP connections.

**Figure 3: The MUPE application platform structure.**

MUPE platform allows anyone to easily create their own context aware mobile services and applications. MUPE application/service development is based on the use of Eclipse environment. A special MUPE project is available for the Eclipse environment and thus all the necessary files are automatically created for the developer. Several tools exist within Eclipse for the further development and deployment of end user services. The service development for the MUPE can be divided into three parts; MUPE client, service logic and various contexts/context sources.

MUPE client is a static single installation browser type of an entity running at the mobile device. Client is developed by professional developers and as such not changed by the end users. Client functionality is fully defined at the server end and communicated through XML files. However, client may use the context aware information available at the mobile device by adding available code snippets i.e. plugins to the appropriate XML file. Plugins exist for the most common device properties like nearby network addresses (Bluetooth), GPS coordinates or general location as well as local sensors e.g. camera. Deployment of the client-side contexts is already at the required technology agnostic level for the end user enabled service creation.

Service logic consists of the main parts of the MUPE service. Although the basic structure of a service is automatically created for the user this part is still far away from technology agnostic service creation. In order to create fully new service the user needs to extend the basic structure and logic to fulfill the requirements of the proposed service. All this is pure Java and is supported by the Eclipse environment. As MUPE is an open source approach, there exist several readily implemented services that anyone may extend and modify.

Server side, i.e. external, contexts extend the available context information to almost any context source. Typical external sources include web pages that are parsed for the necessary information. Examples of this type of a context sources are weather and stock information as well as search engines. The use of server side contexts is a bit harder than client side contexts as the location parameter such as the web page address need to be modified. If suitable parser is not yet implemented, this needs to be done by implementing the parsing methods. Luckily several external context sources have been implemented and added to the Eclipse environment. Context Selection tool allows the use of readily implemented contexts within MUPE projects.

The idea of the Context Selection tool is to automate the process of adding server side contexts to different MUPE applications. The main Context Selection plugin provides an extension point to ease the creation of different contexts. The user interface of the tool is based on a build-in wizard that includes all found extender plugins to the main page from where the user can easily select the desired contexts to the application. The extender plugin provides client method source code which will be included in the context manager class, and corresponding context rules in an XML format. Both of these sources should be treated as examples so that they contain all possible information that can be obtained from the context producer this context is meant to be used with. The idea behind this is that the creator of context producer would also create an example of how to access the information it fetches as they have the best knowledge of their own work. By doing this the reusability of once implemented context sources is improved. The process of adding a context to an application is presented in .

MUPE platform has been used for user based application creation in two separate four day long code camps: one arranged in Lappeenranta, Finland and the other in Cairo, Egypt. The courses given in each place were as identical as possible. All students had some background on JAVA programming but no prior knowledge on MUPE platform. Both courses consisted of one tutorial day, during which the students got the basic knowledge of the platform. Based on the basic knowledge the students defined their application/service under the code camp topic- mobile community services. Next two days were used for coding and
the last day was used for presentations and analysis of the solutions. The results were encouraging. All 6 groups at Lappeenranta and all 11 groups in Cairo were able to produce working services under the code camp focus. One of the key results of these code camps is the wide variety of the user defined services. With the same code camp topic students created services from local to global services, from single user to community applications.

V. CONCLUSION

The current service development and creation are mostly limited to professional developers only, because the development process is known as complex and resource demanding. In our research vision, we foresee a user-centric environment. Users should also be given the chance to create services, even if they do not posses sufficient technical know-how.

As an answer to this vision, we have proposed a solution where easy service creation can be made possible for everyone. A conceptual architecture was proposed to realise technology agnostic service creation, where a user can create services without the need of any programming-related knowledge. The implementation is a service creation platform that provides tools for both developers and users. Professional developers will be assisted with guidance to deploy their products as service building blocks in the service execution environment. End-users can express their service needs by using a graphical service creation tool. Semantic service description will be responsible in linking the services created by end-users to the designated service building blocks. The introduction of the semantic representation for services will further simplify the service creation process, hence making it possible and available for almost everyone.

The prototype tools are our first step to realise the vision. Through the implementation and demonstrations carried out, feedback and suggestions were collected. Based on them, future improvements and tests will be performed. User evaluation is planned to get hands-on feedback to further enable us to produce service creation tools that can be applied in potential markets.

ACKNOWLEDGMENT

This work at Kassel has been performed in the framework of the IST project SPICE, which is sponsored by the EU under contract IST-027617, and in the S4ALL project, which is partly funded by the German "Bundesministerium für Bildung und Forschung" (BMBF). The authors acknowledge the contributions of their colleagues in the above projects.

The work at Lappeenranta has been conducted under the contract with Nokia. The authors acknowledge the contributions of their colleagues in the above project

REFERENCES