An Adaptive Model for Reconfigurable Autonomous Services using Profiling

SÉBASTIEN TRUCHAT, GERHARD FUCHS, FALKO DRESSLER
University of Erlangen-Nuremberg, Department of Computer Science 7,
Martensstr. 3, 91058 Erlangen, Germany
email: {sebastien.truchat,gerhard.fuchs,dressler}@informatik.uni-erlangen.de

STEFFEN MEYER
Fraunhofer Institute for Integrated Circuits IIS, Project group Interoperative Systems,
Nordostpark 93, 90411 Nuremberg, Germany
email: mey@iis.fraunhofer.de

Abstract— **The importance of mobile services in our everyday life is growing while at the same time new interoperability issues arise due to hardware and software heterogeneity. Therefore, new architectural paradigms and models are needed to enhance software engineering methodologies with regard to platform independence and interoperability. This paper describes an UML pattern based approach for developing reconfigurable autonomous mobile services. Through the analysis of an m-commerce project, the relevance of our proposed architecture will be explained. Our focus lays on a generic reconfiguration mechanism based on profile matching from software modules. This profiling part will be further described and discussed. Finally, the applicability of our approach is investigated within a project about reconfigurable indoor navigation computers and a project about robot assisted sensor networks.**

Index Terms— **Mobile services, software modelling, interoperability, autonomous pervasive systems**

I. Introduction

In the frame of a research in the area of interoperative systems, our department has been involved in the development of the m-commerce project HORN (Home and Office Replenishment - Nürnberg) [1], which is an initiative of the Fraunhofer Institute IIS. The goal of our research is to enhance the development of mobile services in a platform independent way with regard to interoperability features (i.e. heterogeneous environments). The analysis of the architecture of the HORN project will lead to the presentation of our Mo.S.I.S. project (Modular Software-engineering for Interoperative Systems), which is a pattern based approach to make mobile services reconfigurable in heterogeneous environments. The main focus of this paper relies on the profile based reconfiguration. Finally, the work being done in two related projects using this reconfiguration mechanism will be presented.

A. HORN

The Home and Office Replenishment service in Nuremberg supplies consumers with dry packaged and fast moving consumer goods (FMCG). The consumer buys his stock of products to be replenished once at the beginning of the service process, and defines maximum and minimum stock as well as the medium range for each product using a Pocket PC that is equipped with a barcode scanner and an additional DECT module that provides the connection to the service provider's data base via a residential gateway connected to the Internet through a modem. After consumption of a product, the EAN article number that is printed on the product package is scanned with the same handheld, and a message is generated automatically to be sent to the service provider. Here the consumption messages are gathered and stored in a data base of the residential gateway. Once each day, a special decision support software algorithm, that takes various system and customer specific restrictions into account, checks the data base and generates picking & packing orders and shopping lists that are sent to a professional shopper who is also equipped with a wireless pocket PC with integrated GSM module and barcode scanner. The shopping is done, products are paid and packed into transport boxes, and carried to delivery boxes for unattended delivery. The consumer is supported with an SMS on his mobile phone that contains dispatch information and an electronic bill of delivery that is sent to his wireless Pocket PC. After taking out the items, the delivery is accepted or not, an acknowledgement of receipt is sent back to the service provider who then initiates the invoice.

The study from an economic and logistic point of view can be found in [2]. The pilot trial started at the end of August 2003 and ended at the end of March 2004.

Why do we need this kind of infrastructure? Commonly, mobile services use the Internet model [3]: the application runs on an Internet server, while the mobile client has an access to this service through a standard browser. This client browser is used as a visual human machine interface, while data are actually processed on the server side. One obvious advantage of this architecture, from the point of view of software engineering and software maintenance, is that the software only needs to be implemented for the server platform. Processing power restrictions are to be considered on the server side. Mobile clients only need a browser to use the service. One drawback of this strategy is that there must be a browser implementation for every new mobile platform, and the system must agree with the restrictions of this browser, but the major problem is that the service accessibility depends on the communication to the server: it must be fast enough, reliable, and always available. As the consumer wants to be mobile everywhere in his house, even e.g. in the cellar where there is no reception of any kind, to manage his beverage stock for example, the mobile service has to be an autonomous application on the mobile terminal. This is one essential characteristic why HORN should be considered as an m-commerce initiative rather than e-commerce: most of the transactions are made by the customer in an offline situation [4]. Furthermore, the price of the communication to the Internet can be restrictive, and the battery life reduces with the power consumption of the communication interface that has to be powered during the time of connection to the server.

Some approaches, not constrained with local applications, combine the advantages of a server based solution with short range charge-free wireless communication such as the Flower framework [5]. Some approaches combine server based services with some client specific software pieces using existing WSDL standards [6]. Our architecture focuses on local applications and short range wireless networks.

The "HORN architecture" would reveal expensive for the operators of the residential gateways and servers, and as a consequence for the consumer, if it could only be used for this one single service, so the goal of our research is to enhance the reusability and interoperability of existing infrastructures of that kind.

In our research infrastructure, the mobile device has no direct contact to the service provider but can start a communication to a residential gateway. Our "residential gateway" can rather be considered as an association of a local server that stores data for the mobile terminal and a residential gateway to access the internet on demand (no permanent connection to the internet). Since the mobile device has only sporadically a connection to a (local) server, the applications must run directly on the device without further external control and management. That is what we call an autonomous service.

So the "service provider", who offers applications for the mobile clients, must implement software modules for every spectrum of devices on which he wants his application to run. Moreover, he may want to update some of these modules sometimes. Once the infrastructure is standing, and when there is a possibility to update or reconfigure the mobile devices, there might be other service providers who could want to use this infrastructure to offer their services. Thus our goal is to find a way to reconfigure most mobile devices in that infrastructure, independently of their operating system, and to improve the software development process for such mobile applications.

To realise the reconfiguration software in a platform independent way, one can choose a middleware or virtual machine (e.g. Java) based approach. The idea is to compile the software once, assuming that the binary will be able to run on any platform. There are two reasons why we could not use this approach. First, this middleware (or virtual machine) has to be present on any platform. Unfortunately, as new mobile devices are appearing fast, one can not always take this for granted. Some devices are even too resource constrained to choose this approach. The other reason is, as mobile devices differ a lot regarding their hardware interfaces (commencing with the display), the code for a new platform would have to be adapted and recompiled anyway. So another way would be to implement the software in a language for which a dedicated compiler exists for every platform. Here again the code would have to be adapted regarding the hardware interfaces. The idea is to implement the software once, to adapt it slightly and compile it for new platforms. Here again one can not take it for granted that the right compiler is available for any new platform. As we assume that new mobile platforms are appearing fast and mostly have a short lifetime, our goal is a fast software engineering. May it be a middleware or a compiler to realise for a new platform, the initial work would be too high.

So our approach is to concentrate on the base mechanisms we need for a generic reconfiguration, to find adequate architectures, and to discuss the possible implementations. By describing these in design patterns, our goal is to enhance the software development process.

B. Mo.S.I.S.

The goal of the Mo.S.I.S. (Modular Software-engineering for Interoperative Systems) project is to make the development of mobile services in general, and especially for HORN-like architectures and systems with few resources, more cost efficient [7].

First of all, there is a need for a generic reconfiguration framework for the mobile devices and residential gateways (i.e. local servers), since there does not exist a standard for software deployment that works for every combination of hardware and operating system platform, especially very lightweight devices. To achieve this, an ontology to relate the profiles from devices to software modules with regard to the description of application profiles has been developed. These reconfiguration rules can be considered as the necessary clear invariants that govern the entire system (known as volatility principle [8]).

The second step was to develop patterns permitting a rapid prototyping of the reconfiguration software for any new platform (for new residential gateways as well as for new mobile terminals), using the rules of this ontology. This is also the preliminary condition to manage a safe reconfiguration and context awareness mechanism. The benefits of architectural patterns related by Christopher Alexander [9] can also be applied to software engineering. A design pattern is a generic solution to a problem that occurs again and again in various forms. To describe our software design patterns methodically, we chose the way proposed by Gamma [10].

To minimize the work of code adaptation i.e. to improve the code portability, this pattern study also includes the general architecture of the mobile applications, and the programming of the communication interfaces.

Cost efficiency can be reached by two aspects: first, by a dedicated "business model" (as shown by the example of

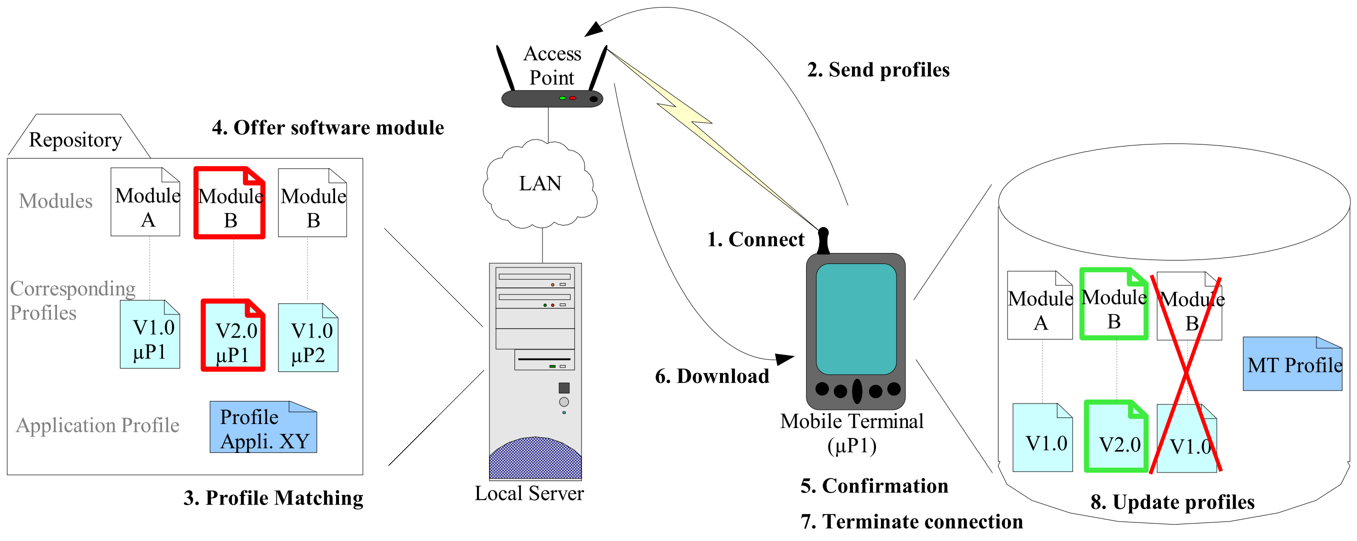

Fig. 1. Reconfiguration process of the first prototype.

"B4U" [11]) where the residential gateway operators can lease the right to use their infrastructure to service providers, and second, by shortening software development time on both the service provider side and the residential gateway side by the use of dedicated design patterns, and code framework reuse, when possible, to implement the mobile services (i.e. local applications in a HORN like infrastructure) for the desired spectrum of devices.

The following part describes the reconfiguration concept of Mo.S.I.S., where the "local server" of our first prototype plays the role of the residential gateway of the HORN project (which in fact has a kind of local server role rather than just a gateway function).

For the first tests, the local server part is played by a standard PC connected to a WLAN access point. In the future, this part might be embedded in the residential gateway platform, which assumes of course much less memory resources and computing power. The mobile terminal can try to connect to any accessible local server over WLAN on request of the user. This approach has been preferred to a periodical scanning in order to save battery life, though it assumes the user must be aware of the presence of a potential local server.

The reconfiguration process can be described as follows (Fig. 1): 1) the user asks the mobile terminal to try to connect to some local server. 2) if successfully connected, the mobile terminal sends its profiles to the local server. 3) the local server matches these profiles with those of available software modules in its repository. 4) as soon as there is a module that matches, it is offered for download. 5) if the user acknowledges, the module can be downloaded. 6) the module is being downloaded. 7) when all matching modules have been downloaded, the connection is being terminated. 8) after successful download, the software profiles of the mobile terminal can be updated according to the new modules.

C. *Related Projects*

Besides the first prototype described earlier, two new projects use the Mo.S.I.S. work for reconfiguration.

One project deals with a PDA based navigation terminal, which has to download data (e.g. navigation maps) context dependently, and to be updated automatically sometimes. This work will be further described in part III.

The second project is more resource constrained. It aims at automatically reconfiguring a sensor network using mobile robots in order to preserve a functional density of the network. This will be further described in part IV.

The following section describes our reconfiguration mechanism. First, the base patterns of Mo.S.I.S. will be presented, then the focus will lay on the profile matching part. After a short introduction to profiling, the nature of our mechanism will be formally described, to finally present the dedicated design pattern.

II. Generic Reconfiguration

A. *Patterns for Reconfigurable Autonomous Services*

Our Mo.S.I.S. concept relies on four basis patterns:

- Profile comparer
  This pattern builds the base of our profile matching concept. Mobile devices and software modules are characterised by profiles. Through matching of these profiles, diverse devices can be reconfigured automatically, even in heterogeneous architectures. This part will be presented in this paper.
- State based RPC
  For communication between heterogeneous devices, Remote Procedure Call seems to be the best solution. In order to make our reconfiguration process work even on resource constrained platforms, we defined a lightweight non-blocking RPC mechanism. The principle is to define only the necessary RPC commands, and to implement on both sides (mobile device and local server) a client and a server stub, the behaviour of which is determined by a state machine. The state machine on mobile device side and local server side are depending on each other, and define the global behaviour of the system. In that way,

type converting of parameters is implemented on demand within the states, and the system is not blocked by a RPC call, since the response to the call will be determined by the resulting state.

- Adapted components
  In order to enhance interoperability between software components written in different languages, we describe a method to define data oriented interfaces and the corresponding wrapper code. This is more lightweight and shorter to implement than a complete component model.
- Protocol dependent communication prototype
  This pattern is useful when the communication interface cannot be interrupt controlled (e.g. a modem based communication card, and the interrupt signals are not correctly emulated on the mobile device). This means, the data flow has to be parsed for commands. The overlaying protocol (e.g. RPC) has to be defined in a way that data flows containing the reserved "command words" can be transmitted although.

Once all these patterns were defined, the next natural step was to define a UML profile for Mo.S.I.S. (i.e. a UML profile for autonomous reconfigurable mobile services) in order to facilitate software engineering and documentation of projects using these mechanisms.

In the following part, the profile matching concept will be explained.

B. What is Profiling?

What we call profiling consists of two parts:

1) To define profiles that characterise a software service (e.g. software modules, applications), and profiles that characterise environments i.e. platforms on which services can be offered (e.g. mobile devices, user preferences).
2) To define the profile matching rules defining how these platforms can be reconfigured with these services. The word reconfiguration stands here in general for any new software configuration (in the sense of loading new software), even if it is only updating.

The reason why we call our services autonomous is that they have to run autonomously on a (mobile) target platform, in opposition to server based services (e.g. web services). Nevertheless, as the profile matching aims at reconfiguring devices automatically even in heterogeneous architectures, this work concerns the topic of autonomous computing too, since the goal is to offer new or updated services without any supplementary administration work.

An ontology can be seen as a formal specification on how to represent objects or entities, and the defining of rules on how they stand in relationship. So one can say profiling is a kind of interoperative reconfiguration ontology.

Composite Capabilities / Preference Profiles (CC/PP) [12] offers a way to describe profiles. One typical application for CC/PP is content adaptation: a client sends a request in HTTP with its profile, and the HTML server matches the document profile with the device profile to adapt the document that is sent in the HTTP response. As meanwhile CC/PP is a widespread standard, it has been used for the implementation of the first prototype. Another standard in the domain of management information is the CIM (Common Information Model) from the DMTF (Distributed Management Task Force) [13].

Nevertheless, for the implementation of the new PDA based prototype (navigation terminal), we chose to define a more simple XML based way to describe our profiles. As for the project on sensor/actuator networks (ROSES), we had to define a byte oriented profile description in order to meet the demands of the very limited resources anyway.

A similar functionality exists for Java platforms: Java based over-the-air provisioning permits to install MIDlet suites through the use of java application descriptors (JAD) which are kinds of software profiles. The process is controlled by the application management software (AMS). This architecture needs a JAD-server, a JAR server (containing the .jar archive), and a notification server [14].

The following section deals with the formal description of the reconfiguration process of Mo.S.I.S.

C. Profile Matching

Let ME (Mobiles Endgerät in German) be the set of all possible mobile devices in the considered scenario. A mobile device is characterised by a tuple of properties.

Let Eig_{ME} be the set of possible properties (the German word for property is Eigenschaft) of a mobile device (e.g. Processor xy, Displaying abilities, RAM, ...). Within Eig_{ME} there are equivalence classes (e.g. subset of properties that define a processor). There are n equivalence classes, so the mobile device is characterised by a tuple of dimension n, which we call the profile of the mobile device.

Let us extend Eig_{ME} with an element nil. This element is used in a profile tuple when a property does not exist (i.e. is not relevant, e.g. a mobile device has no display, just an acoustic output). Let Eig_{ME0} be this new set: $Eig_{ME0} = Eig_{ME} \bigcup nil$. (the element nil builds an equivalence class by itself in Eig_{ME0}).

As there is at most one element from an equivalence class that corresponds to a profile element of a mobile device, there exists a function that associates its profile to every mobile device me. Let $Profil_{ME}$ be this function.

Let $\dot{e}me1$ till $\dot{e}men$ be the equivalence classes of Eig_{ME} and $\dot{e}me1_0$ till $\dot{e}men_0$ these equivalence classes extended with the element nil.

$$Profil_{ME} : ME \rightarrow \dot{e}me1_0 \times \dot{e}me2_0 \times ... \times \dot{e}men_0$$
$$me \mapsto (eme1, eme2, ..., emen)$$

Let MOD be the set of all software modules. Additionally to the properties describing the nature of the module (e.g. for which processor it has been compiled), a software module needs an identification (e.g. a name) defining its role in an application. Let ID_{MOD} be the set of identifications. In the same way as for the profile of a mobile device, we define a tuple of dimension m that characterises the profile of a software module. The first property in this profile is the identification of the module. Let $Profil_{MOD}$ be the function that associates its profile to every software module:

$$\begin{aligned} Profil_{MOD} : MOD &\to ID_{MOD} \times \dot{e}mod2_0 \times ... \times \dot{e}modm_0 \\ mod &\mapsto (idmo, emod2, ..., emodm) \end{aligned}$$

In a similar manner, we define the profile of an application. Let APP be the set of all applications, and ID_{APP} the set of identifications for applications. Following properties are needed to define an application:

- an identification $idapp$ $(idapp \in ID_{APP})$.
- k elements ($k \in \mathbb{N}$), that define the very own properties of an application (e.g. version).
- a list of q software modules ($q \in \mathbb{N}^*$), that compose the application. Of course, this is a list of distinct software modules i.e. in this list there are no two elements that have the same identification $idmo$. So we define here that a module with a given identification (that means a given functionality) is needed only once (e.g. only in one version, or with one displaying resolution) on the mobile device.

Since not every application is composed of the same number of modules, it is important to notice that q can get different values depending on the application profile.

Let p be the dimension of an application profile. Since an application profile is composed of an identification, k properties, and a list of modules, $p = k + 2$. One could also say $p = 1 + q + k$, but it is more convenient to consider the list of software modules as one property, since in this way the dimension of an application profile remains unambiguous, and since this list of modules has a particular role during profile matching (see application rules).

So let $List_{APP}$ be the set of possible lists of software modules for an application. A list of software modules $moliste$ is a subset of the set of software modules, where the modules of this list are represented by their identification.

$$List_{APP} = \{moliste \mid moliste \subset ID_{MOD}\}$$

Let $Profil_{APP}$ be the function that associates its profile to every application:

$$Profil_{APP} :$$

$$\begin{aligned} APP &\to ID_{APP} \times \dot{e}app2_0 \times ... \times \dot{e}app(p-1)_0 \times List_{APP} \\ app &\mapsto (idapp, eapp2, ..., eapp(p-1), moliste) \end{aligned}$$

In a similar manner, we define the profile of a user. In this profile there are on the one hand properties that define the user (such as his name), and on the other hand his preferences (e.g. preferred languages such as German, English, in this priority sequence). As seen in the application profiles, a preference property can be a list (possibly ordered). The user profile is mainly a preference profile, and stands for every kind of profile that defines preferences (e.g. context profile).

Let B be the set of possible users. A user is characterised by a tuple of dimension r. Let $Profil_B$ be the function that associates its profile to every user:

$$\begin{aligned} Profil_B : B &\to \dot{e}b1_0 \times \dot{e}b2_0 \times ... \times \dot{e}br_0 \\ b &\mapsto (eb1, eb2, ..., ebr) \end{aligned}$$

The goal of the profile matching is to reconfigure mobile terminal devices automatically, that is to say, provide them automatically the right software modules to be downloaded during a reconfiguration process. To this end, rules are necessary to define how the information in the profiles have to be analysed, and which correlations have to be considered. The rules aim to define a subset from the set of all software modules present on the local server, which is meant to reconfigure the mobile device.

Three types of rules define the profile matching:

- the **module rules** that define which modules from a set of software modules is compiled for a mobile device at all.
- the **application rules** that define which modules from a set of software modules are able to run in the frame of an application, i.e. which applications can actually be built from a set of modules. For example: if an application is composed by three software modules, but only two of the modules are available in the considered set of software modules, even these two modules are unnecessary to be downloaded. The considered set of software modules consists, in the case of a reconfiguration, of the set of modules present in the repository of a local server, and the set of modules already installed on the mobile device.
- the **priority rules** that define which software modules are preferably being downloaded. In this case, the goal is to determine only one module to be downloaded, among a set of modules possessing the same identification (i.e. the same functionality). The module having priority can be found out through the preference profiles (e.g. user profile). It can be a module displaying information in a specific language for example.

In the following part, we will describe how a mobile device can be reconfigured from a module pool on a local server by applying these rules.

Let $me \in ME$ be the mobile device that is to be reconfigured. On me, its profile is stored, besides the already installed software modules:

$$Profil_{ME}(me) = (eme1, ..., emen)$$

Let $SWBibme$ be the set of modules already installed on the device. $SWBibme \subset MOD$. Profiles of these modules are stored on me too, since at least the identifications of the modules have to be known during the reconfiguration process, to determine which modules are already installed on the device. Although, not all the information of the original module profiles have to be stored on the device, since information about the target platform is already present in $Profil_{ME}(me)$.

Let MOD_{LS} be the set of software modules that are available for reconfiguration on the local server. $MOD_{LS} \subset$

MOD. The profile of these modules and the profiles of the applications corresponding to these modules are also stored on the server. Let APP_{LS} be the set of applications on the local server. $APP_{LS} \subset APP$.

The profile matching is being realised in three steps corresponding to the three types of rules. By the use of each rule, a subset of software modules is being generated, that becomes the set of modules on which the following rules are used.

- **Module rules:**
  The module rules define which modules of MOD_{LS} are compiled for me. There are t rules, $t \in \mathbb{N}^*$. In a rule $i, i \in \mathbb{N}^*, 1 \leq i \leq t$, it is verified, for every module mod of MOD_{LS}, whether a_i properties of the device profile are in relation Rel_i with b_i properties of the profile of this module, $a_i \in \mathbb{N}^*, b_i \in \mathbb{N}^*$. Rel_i can be an equivalence relation or an ordered relation. Let $Proja_i$ be the projection of $Profil_{ME}$, on the a_i relevant properties of me, and let $Projb_i$ be the projection of $Profil_{MOD}$, on the b_i relevant properties of mod.
  Let $ModReg$ be the subset of MOD_{LS} resulting from the module rules. $ModReg$ is the set of modules of MOD_{LS} obeying to the t module rules, minus the set of modules already present on the device.

$$ModReg = \bigcap_{i=1}^{t} \{mod \in MOD_{LS} \mid$$

$$(Proja_i(Profil_{ME}(me)))\ Rel_i$$

$$(Projb_i(Profil_{MOD}(mod))\}$$

$$/\, SWBibme$$

  When the equivalence classes of the properties of the software modules are defined so that they correspond to the equivalence classes of the properties of the mobile device, there is no need anymore to verify if a_i properties are in relation with b_i properties, but if one is in relation with one. This point is explained through the selection of a central vocabulary in the description of the design pattern.
- **Application rules:**
  The set $ModReg \subset MOD_{LS}$ has been generated by the module rules. The application rules will now be applied on this subset. These rules define which software modules of $ModReg$ are actually able to run in the frame of an application of APP_{LS}, under consideration of the already installed modules $SWBibme$. Let $AppReg$ be the subset of $ModReg$ generated by the application rules. For every module $mod \in ModReg$ will be verified if there exists an application $app \in APP_{LS}$, containing mod in its list of modules and if furthermore its list is completely present in $ModReg \cup SWBibme$. Let $Proj_{AppList}$ be the projection of $Profil_{APP}$ on the list of software modules.

$$Proj_{AppList} : ID_{APP} \times$$

$$\dot{e}app2_0 \times ... \times \dot{e}app(p-1)_0 \times List_{APP} \to List_{APP}$$
$$(idapp, eapp2, ..., eapp(p-1), moliste) \mapsto moliste$$

  So $AppReg$ can be written down as follows:

$$AppReg = \{mod \in ModReg \,|\, \exists\, app \in APP_{LS} \,|$$

$$(mod \in Proj_{AppList}(Profil_{APP}(app))) \wedge$$

$$(Proj_{AppList}(Profil_{APP}(app)) \subseteq$$

$$(ModReg \cup SWBibme))\}$$

- **Priority rules:**
  From the module rules and the application rules, a set of modules has been generated in which there are possibly several modules with the same identification. Among modules with the same identification, one has to be given the priority. There exist s priority rules, $s \in \mathbb{N}$.
  This means that for every priority rule, there exists a property in the module profile that is relevant to determine the priority. For this property, an ordered relation has to be defined.
  For priority rule $i, i \in \mathbb{N}^*, 1 \leq i \leq s$ (when $s \neq 0$), we define the projection $ProjPrio_i$ that is the projection on the relevant property $emodx_i$ of the module profile, $x_i \in \mathbb{N}^*, 1 \leq x_i \leq m$. We also define the ordered relation $\leq_i$ on these properties. This means that the properties of $\dot{e}x_{i0}$ have to be fully ordered, so that one module is always given the priority. The preferences of the preference profile eventually change this order. Since there are several priority rules that can only give the priority to one module, we also have to define a priority among the priority rules, that means to define the sequence in which the rules are applied. Priority rule $i+1$ will be applied after priority rule i. Let $PrioReg_{(i+1)}$ be the set of software modules that is generated when priority rule $i+1$ is applied on $PrioReg_i$.
  Let $PrioReg_0 = AppReg$.

$$ProjPrio_i :$$
$$ID_{MOD} \times \dot{e}mod2_0 \times ... \times \dot{e}modm_0 \to \dot{e}modx_{i0}$$
$$(idmo, emod2, ..., emodm) \mapsto emodx_i$$

  $PrioReg_{(i+1)}$ is the set of software modules of $PrioReg_i$, the identification of which does not occur twice, or the property $emodx_i$ of which has the highest value according to the ordered relation $\leq_i$.

$$PrioReg_{(i+1)} = \{mod \in PioReg_i \,|\, \forall\, mod2 \in PrioReg_i$$

$$(Proj_{ID}(Profil_{MOD}(mod2)) \neq$$

$$Proj_{ID}(Profil_{MOD}(mod))) \vee$$

$$ProjPrio_i(Profil_{MOD}(mod2)) \leq_i$$

$$ProjPrio_i(Profil_{MOD}(mod)))\}$$

  $PrioReg_s$ is the final subset of MOD_{LS}, with which the device me will be reconfigured.

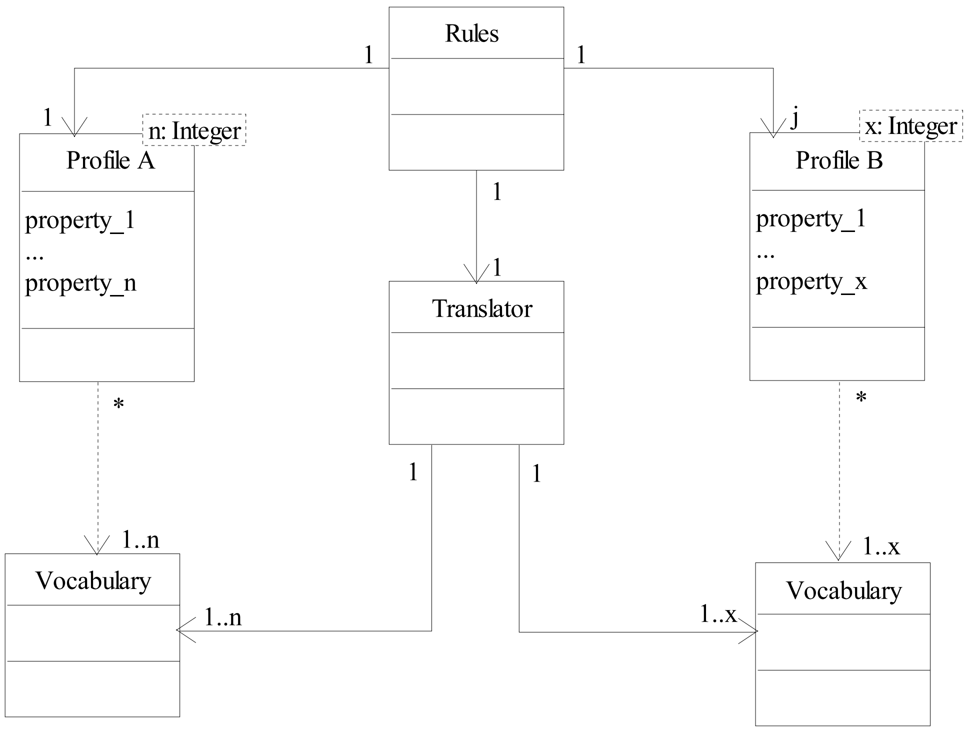

Fig. 2. Structure of the profile comparer.

D. The Profile Matching Design Pattern

The complete description of the proposed design pattern involves discussions about the various ways to implement it in a real system and the advantages and inconveniences of each solution. Detailed examples illustrate these solutions too. In this paper we concentrated only on a few general points.

Name

Profile comparer (Profilvergleicher).

Purpose

The purpose is, among a set of objects (e.g. software modules), to identify the subset of objects that can fulfil their functionality in a certain environment. The properties of these objects are characterised by so called profiles. The properties of an environment are characterised by one or more profiles too. The relations between the properties of the environments and the properties of the objects are described in rules. When the properties are described using different vocabularies, a translator defines the equivalencies between the vocabularies in order to apply the rules.

Motivation

A typical scenario would be that a service provider wants to provide services for mobile devices. These mobile devices do not dispose of a permanent connection to a server of the service provider. Consequently, the application that realises this service has to be installed locally on the mobile device.

Now the service provider wants to update his service, or modify it, or offer new services. To this end, he uses the sporadic connection that a mobile device sometimes builds up to some of his servers (a so called local server) to reconfigure the device automatically. Since the environment can be heterogeneous (different local server platforms, different mobile devices), the reconfiguration mechanism has to be platform independent.

An application can be split in several smaller software modules, so that only the necessary modules are exchanged during a reconfiguration process. A software module is characterised by a profile (module profile), in order to know for which platform the module has been compiled, what is his role in an application, and what its other properties are. Additionally, there exist profiles that describe of which modules an application is composed (application profiles). These modules, with their module profiles and application profiles are available for download on the server. Several types of mobile devices exist, on which services have to be provided, so a mobile device is characterised by a profile too (device profile). Some preferences can be described in a profile too (preference profile), as for example the desired language.

When the mobile device builds up a connection to a server, it sends its profiles to it in order to have this latter doing the profile matching (the profile matching should be realised on the side disposing of the most computing power, but this is not necessarily the server). Then the profile matching occurs in three steps:

1) The module rules determine which of the software modules on the server are, in principle, compiled for the mobile device platform. Here, one profile comparer compares the device profile with several module profiles. Additionally, it has to be considered that the modules that are already present on the device do not have to be downloaded. This generates a subset of modules.

2) The application rules determine which modules from this subset of software modules are able to run in the frame of any application characterised by the application profiles on the server. Here, one profile comparer compares each application profile with several module profiles. This generates a new subset of modules.
3) The priority rules verify if, in this subset of modules, there are modules with identical functionality (i.e. the same identification) but differing properties. From these modules with identical functionality, one is given the priority. Here, one or more profile comparer compare one preference profile at a time with several module profiles. This generates the final list of modules that have to be downloaded on the device.

Applicability

A profile comparer is useful when following conditions come together:

- Different types of mobile devices must be reconfigured automatically. The properties of the mobile devices are so different, that different software modules are necessary.
- It is not possible to provide one software for the whole spectrum of mobile devices through a virtual machine mechanism because of one of the following reasons:
  - there is no standardised virtual machine for all the devices
  - the software modules have to be compiled in a native format anyway [15]
  - the interfaces of the devices are so different that different software modules are necessary [16]
  - the terminal devices do not have enough resources to host a virtual machine
  - the virtual machine does not exist at all for some devices
  - some services that have not been compiled for a virtual machine have to be reconfigured too
- For the desired spectrum of mobile device there exists no synchronisation software that can be easily automatised (above all in the special case of PDAs), and that runs on every mobile device platform as well as on every local server platform.

Structure

Figure 2 shows the structure of a profile comparer in a class diagram.

Participants

- Profile A
  - represents the properties of the environment, for which a list of matching objects has to be found
  - is a pure profile consisting of n properties
  - every property is an element of a reference vocabulary
- Profile B
  - represents the properties of an object in the set from which the subset of objects matching to the properties represented by Profile A have to be identified
  - is a pure profile consisting of x properties
  - every property is an element of a reference vocabulary
- Vocabulary
  - defines the vocabulary used to represent a property
- Translator
  - is the central reference, hat transforms all the vocabularies in a central reference vocabulary
  - knows all the vocabularies to be translated
  - offers an interface to transform the properties of a vocabulary in properties of the central reference vocabulary
- Rules
  - defines the rules on how properties of Profile A have to match with properties of Profile B using the central reference vocabulary of the Translator
  - uses a Translator to transform the properties of Profile A and Profile B into properties according to the central reference vocabulary in order to compare them through the matching rules
  - has an access to all the profiles for this purpose
  - offers an interface to give back the list of Profile B objects matching to the properties of Profile A according to the rules

A Profile B object has x properties. This number of properties remains constant in the frame of one profile matching process (e.g. all the software modules have the same number of properties). The same can be said for the n properties of Profile A. Nevertheless, it is theoretically possible that several properties use the same vocabulary, that is why Profile B stands in a dependency relation with 1 to x vocabularies, and Profile A with 1 to n. It is important to notice that the elementary profile comparer compares one Profile A with j Profile B objects. To realise a complete profile matching process, a complex composition of several elementary profile comparer is necessary. The profile comparer pattern only represents the general elementary foundation-stone.

Interaction

- The class Rules wants to verify a list of rules. A rule i will verify if a properties of Profile A are in relation $Reli$ with b properties of Profile B, and this for every Profile B object.
- To this end, Rules uses Translator to translate the a properties of Profile A in a property $RefEigA$ of the central reference vocabulary, and the b properties of Profile B in a property $RefEigB$ of the reference vocabulary.
- Afterwards, it will be verified if $RefEigA$ is in relation $Reli$ with $RefEigB$. The result is a Boolean.
- This procedure is repeated for every i rules. If the result is always $True$ the Profile B object matches the properties of Profile A and gets on the list of suitable objects.

Consequences

- During a reconfiguration process, it can be decided whether the user is given the possibility to modify the generated list of suitable modules or not. Giving the user this possibility bears the risk that an inconsistent list of modules is being downloaded on the mobile device (e.g. if the user chooses only some modules of an application against the application rules). One solution would be to say the user is responsible for his actions. The other would be to send this list again to the server to check it.

- If it is possible, during the design phase, to define the profiles in a way that they all use a central reference vocabulary, this solution should be given the priority. In this way the translator becomes unnecessary, which makes the implementation considerably easier.
- It is possible to integrate new mobile devices, with profiles that must use other vocabularies, in an existing infrastructure through the use (or extension) of a Translator. In the same way, a new service provider can use an existing infrastructure to offer his new services.
- The content of the profiles is strongly influenced by the chosen developer tools. For example, if a compiler always compiles a module for a given family of processors, this processor family will appear in one property of the profile.

Implementation

This section deals with some aspects to be considered for the implementation.

- The profile comparer only generates a list of software modules to be downloaded. It does not provide an installation process. In many cases it will be sufficient to download modules on the mobile device, but sometimes a specific installation process will be needed. An installation process can vary significantly from one platform to another. The simplest solution to this problem is to write installation scripts that are also considered as software modules to be downloaded. These installation scripts can be either executed by the user, or executed once automatically after the download process.
- The profiles are stored in files (or memory regions). These profiles must be loaded in objects (or any entity readable in the given programming language) to apply the rules. Many programming languages offer serialisation mechanisms, but it offers some advantages to write the profiles in XML:
  - the profiles can be edited in a normal text editor
  - XML is platform independent
  - there exist XML parser for most programming languages and even for embedded systems [17]

  Of course, some systems exist that are so resource constrained that even XML does not fit.
- After the reconfiguration process, it is important to actualise the software profile on the mobile device side, i.e. the profiles of the installed modules. This means on the one hand to delete the old profile of modules that have been replaced, and on the other hand to download the profiles of the new modules. Here, it is important to notice, that not all the information of the module profiles must be downloaded, since the information relative to the target platform are already present in the device profile. This can shorten the download time and save memory place on the mobile device.
- Modular software engineering does not mean necessarily to download software modules, but eventually by very lightweight platforms, to choose modules (in this case code fragments) to put together in order to compile a complete application for the target device [18].

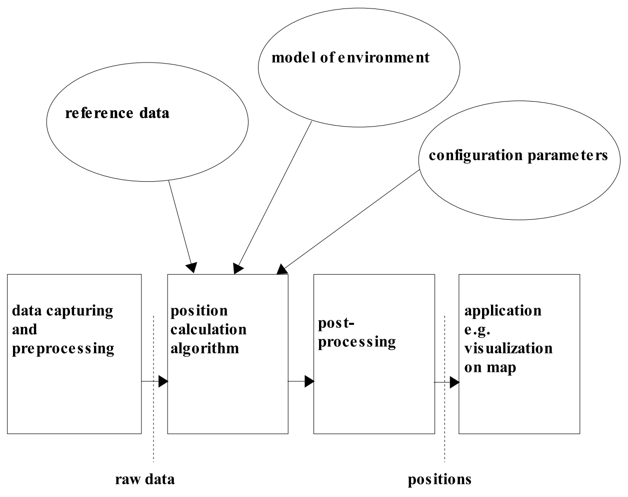
model of environment
reference data
configuration parameters
data capturing and preprocessing
position calculation algorithm
post-processing
application e.g. visualization on map
raw data
positions

Fig. 3. Position Calculation Overview

III. The Navigation Terminal Project

Another real world application for the presented reconfiguration mechanism is a indoor positioning system developed at Fraunhofer Institute IIS. Cellular local networks like DECT and 802.11 WLAN are widely spread. So a software-only positioning system using these existing standards can be widely deployed at low costs. The developed positioning technology [19] is able to locate DECT- or WLAN-enabled devices using standard wireless network components. It uses received signal strength (RSS) information in combination with fingerprinting for position calculation and can be used with any kind of cellular local network that is capable of scanning for communication partners around and reporting their signal strength. As this feature is needed for roaming between base stations, most cellular networks implement it. This includes 802.11 WLAN and DECT.

Position calculation is done completely on the client side. There are several steps needed as shown in figure 3. The mobile device frequently scans for base stations (access points). During the scan, a list of found base stations with their signal strength information is generated. This data is preprocessed to minimize measurement errors. It is then handed over to the position calculation algorithm. This calculates a raw position by matching the raw data to reference data captured before. Also a model of the environment may be used to perform plausibility checks and optimized routing. A postprocessing step transforms raw positions into an application specific position information. This may include coordinate transformation or position to location mapping (e.g. room finding).

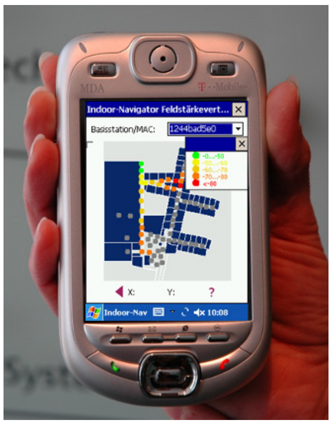

Fig. 4. Reference Data in Office Building.

A. Capturing Reference Data

Capturing reference data is necessary for building up a positioning system based on fingerprinting. The capturing process basically consists of recording RSS data for all base stations at a certain set of positions (”reference points”) and storing them for matching with data measured during localization. The set of reference points is called reference carpet. As the position calculation is done on the client side, reference data has to be stored on each client. Updating this database if necessary is one of the needs of the localisation system.

B. Modelling the Environment

A key feature of the system is the use of a model of the environment. This model is used for two main purposes. It helps the navigation algorithm to minimize errors as it limits the number of directions a user can possibly move to (e.g. a user cannot move through a wall, see office algorithm below). Second the model is used to create a customized map view of the environment for displaying location information. The model contains geometric information about buildings, floors, rooms, walls and any type of object that seems of importance. It is extensible, XML-based and can vary in detail to fit the application’s demands.

Again, as the position calculation is done on the client side, a model representing the actual environment has to be stored on each client. Updating this model and downloading additional models if necessary is one of the needs of the localisation system.

C. Target Platform

The location system is implemented in Java. Although Java is not so popular on PDAs yet, it is very popular on the widely spread mobile phones. As integration of 802.11 WLAN into mobile phones has started (first devices are already available), they are expected to be an attractive target platform soon. Goal was to implement all functionality in Java, only access to the wireless device has to be done in native code. The location software now runs on PDAs and on Laptops. The reconfiguration mechanism has to update java packages and system dependent libraries (e.g. dlls). Therefore it is essential that the reconfiguration is done both language and platform independent.

D. Accuracy and Performance

Positioning accuracy of the WLAN positioning system cannot be specified in general. There are many parameters that influence the results, mainly the number of access points per area and the type and material of obstructions in the building. Also the type of WLAN adapter used may have an impact. Tests in a typical office building (1000 square meter) with 5 access points showed an average position accuracy of about 1-3 meters (office algorithm). In halls accuracy is about 10m due to the lack of obstructions. Obstructions increase accuracy if signal strength values before and behind are significantly different. As the WLAN adapter needs at least 200ms for scanning, the software is slowed down to calculate 5 positions per second.

E. Reconfiguration

For the definition of the profiles, we decided to agree on a central vocabulary, since this is the simplest way. If someday other platforms are introduced, a new Translator can be implemented afterwards if necessary. We defined our profiles as follows. An application profile contains following elements:

- name of the application
- path on the mobile device
- list of modules

The application profile is written in XML style in order to be parsed by tags on any platform (even without XML parser):

```
<AppliProf>
   <Name> Name </Name>
   <Pfad> Path </Pfad >
   <Liste> List </Liste >
</AppliProf>
```

The list of modules itself is written:

```
<Modul> Name of module 1 </Modul>
<Modul> Name of module 2 </Modul>
...
```

A module profile contains following elements:

- name of the module
- path within the application path
- version
- processor (for which it is compiled)
- operating system (for which it is compiled)
- needed RAM memory
- needed resolution of the display

The module profile is written:

```
<ModulProf>
   <Name> Name </Name>
   <Pfad> Path </Pfad >
   <Version> Version </Version>
   <Prozessor> Processor </Prozessor >
   <OS> Operating System </OS >
   <RAM> Memory </RAM >
   <Display> (x, y) </Display >
</ModulProf >
```

The local server also hosts a context profile which contains his name. This is only needed for authentication, which is not very complex for the moment.

These modules and profiles must be stored on the local server according to a defined structure in a repository, so that the profile matcher can always find them.

On the mobile terminal side, the device profile contains following elements:

- processor
- operating system
- memory
- resolution of the display

The mobile terminal profile is written:

```
<MEProf>
   <Prozessor> Processor </Prozessor >
   <OS> Operating System </OS >
   <RAM> Memory </RAM >
   <Display> (x, y) </Display >
</MEProf >
```

The user profile only contains a list of trusted local servers. The module rules are following:

- a module has to be compiled for the processor of the device
- a module has to be compiled for the operating system of the device
- the device must meet the required memory amount
- the display of the device must meet the required resolution

In this particular case, a software module belongs at most to one application. This facilitates the implementation of the application rules, since when one module is missing for an application, all the other modules in the application list can be deleted of the list of modules that have to be downloaded.

The priority rules choose the module with the highest version, among modules with the same name.

IV. ROSES

A. Project Description

The development and the control of self-organizing, self-configuring, self-healing, self-managing, and adaptive communication systems and networks are primary research aspects of the Autonomic Networking group at the chair for Computer Networks and Communication Systems. In the frame of the ROSES (Robot Assisted Sensor Networks) project, we study these aspects on a combination of mobile robots and stationary sensor networks that are usually called mobile sensor/actuator networks.

In this context, we distinguish between sensor assisted teams of mobile robots and robot assisted sensor networks. An example for the former scenario is sensor-based localization and navigation. We developed a robot control system named Robrain for general purpose applications in multi-robot systems. Part of this work was an interface between the robot systems and our sensor motes (see below). This allowed us to study the applicability of the ad hoc sensor network for localization assistance [20]. An example for the latter scenario is assistance for maintenance and deployment of sensor nodes as well as for task and resource allocation [21]. Currently, we are investigating methods for adaptive re-configuration of sensor nodes using mobile robot systems. Two separate goals should be achieved using these techniques: calibration of sensor hardware and re-programming based on changes in the environment. In order to address these issues, we apply profiling mechanisms as described in the following.

B. Adaptive Re-Programming

In our laboratory, we use the Robertino robot platform developed at the Fraunhofer Institute AIS running Embedded Linux and the Mica2 sensor motes running TinyOS [22] developed at the University of Berkeley. We have connected a MIB510 programming and serial interface board with the Robertino and installed a Mica2 node as a base station. This enables our robot to directly communicate with the wireless sensor network. In the following, we concentrate on the reprogramming of the sensor nodes for dynamic adaptation to environmental changes.

For re-programming, we prepare our sensor motes with an initial binary, which contains a module for profiling concerns. The robot can use this module to receive information about the hardware configuration and the currently installed applications of the sensor mote, e.g. Mica2 / Mica2dot, temperature measurement / localization. On the robot, we store nesC-code and code templates that are described by profiles. This enables the robot to select and adapt the source code concerning the current context and requirements and, finally, to create a new binary for the sensor node. The robot can install the image over the air. Figure 5 shows the principal concept of reconfiguration:

(a) Depending on the goal, the robot drives to the position in the sensor network, where reconfiguration is necessary.
(b) The robot collects information about the environment (e.g. current temperature), builds the context and explores his neighbourhood.
(c) All motes, which have received the exploration message, send their profiles.
(d) The robot uses the information gathered in steps b) and c) for profile matching, and to assign the roles of the sensor motes, optimized for the current goal. As a result, it creates the new binaries of the sensor motes.
(e) The robot reprograms the selected sensor motes over the air.

C. Profiling

In the sensor net project ROSES, we had to define a byte oriented profile in order to meet the demands of the very

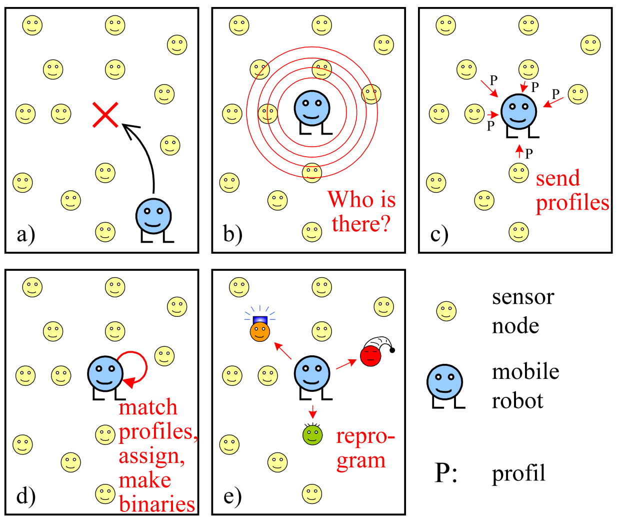

Fig. 5. Application scenario for dynamic reconfiguration in sensor/actuator networks.

limited resources. Each hardware element plugged in a node is defined by an ID (identification number). An ID can be stored in one byte. In the same way, each software module is defined by an ID stored in one byte. As we decide that a sensor node can only contain three different hardware elements, and host up to five software modules, we consequently need only eight bytes to characterise the hard- and software of one sensor node. So the profile of a sensor node is written in the following way: three bytes defining the hardware (one byte per hardware element) and five bytes characterising the software configuration (one byte per module). As we only need a few RPC commands, we reserve only one byte for the RPC command, so a communication datagram for reconfiguration is nine bytes long.

The Robot having definitely more resources, it hosts the database where the complete profile description related to an ID can be found. In that way, once the node has transmitted its profile, the robot can decide with which software modules to reconfigure the node through profile matching.

One can say, the "translation" process into a central reference vocabulary, as explained in the pattern description, is very simple here, since it just consists in converting an ID into the profile description in the database. Application profiles and application rules are not necessary yet, since in these first tests, the modular application structure is static. Only module rules are necessary for the moment, to decide which of the modules at all are meant for a given functionality ("localisation" or "acquiring sensor data") on the given platform. It is obvious, that the definition of a static modular application structure replaces both an application ID and a module list. Nevertheless, for future works, as the complexity of the system grows, it may become necessary to introduce real dynamic applications as foreseen in Mo.S.I.S.

V. Conclusions and Further Work

When it comes to interoperability between heterogeneous platforms, the most platform independent way to "develop" software turns out to be a pattern based approach. Indeed, considering the general mechanisms about reconfiguration permitted us to design patterns, the use of which allows enhancing the development of automatic reconfiguration software as well for PDA based architectures as for resource constrained sensor networks. Of course, the needs and constraints related to these systems vary significantly, which makes the discussion part so important during the elaboration of patterns. The definition of an UML profile based on these patterns turns out to be helpful, since in one application field, the software structures and rules governing the system remain similar. After the first version for ROSES, the profile matching algorithm will grow more complex in order to reach the conservation of a complex functional density. As for the navigation terminal project, we would like to investigate the possibility to edit profiles directly in a UML tool as objects, in order to run profile matching rules written in OCL.

References

[1] HORN - Home and Office Replenishment Nürnberg. Available (September 2005): http://www.horn-nuernberg.de/

[2] A. Pflaum. Die Zukunft des "E-Fullfillment" für Lebensmittel: Versuch einer Prognose, *Logistik Management, 5. Jahrgang 2003, Ausgabe 1*, pp. 25–39.

[3] U. Hansmann, L. Merk, M. S.Nicklous, T. Stober. *Pervasive Computing Handbook*, Berlin: Springer, 2001, pp. 327–340.

[4] J. Zobel. *Mobile Business und M-Commerce*, Hanser, 2001, pp. 3–4.

[5] T. Hakkarainen, A. Lattunen, V. Savikko. Flower – Framework for Local Wireless Services, *ERCIM News Number 50, July 2002*, pp. 51–52.

[6] M. Hillenbrand, P. Müller and K. Mihajloski. A Software Deployment Service for Autonomous Computing Environments, *International Conference on Intelligent Agents, Web Technology and Internet Commerce*, Juli 2004.

[7] S. Truchat. Interoperative Systems for Replenishment (doctoral colloquium of the Pervasive 2004 conference, April 2004), in: Alois Ferscha, Horst Hörtner, Gabriele Kotsis (eds.): *Advances in Pervasive Computing*, pp. 161–166. Vienna: Österreichische Computer Gesellschaft.

[8] T. Kindberg and A. Fox. System software for ubiquitous computing, *IEEE Pervasive Computing, January-March 2002*, pp. 70-81.

[9] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King and S. Angel. *A Pattern Language*, pp. x. New York: Oxford University Press, 1977.

[10] E. Gamma, R. Helm, R. Johnson and J. Vlissides. *Entwurfsmuster - Elemente wiederverwendbarer objektorientierter Software*, pp. 8-10. Addison-Wesley 2004.

[11] T. Haaker and O. Rietkerk. Introducing New Mobile Services Faster, *ERCIM News Number 54, July 2003*, pp. 44-45.

[12] Composite Capabilities / Preference Profiles: Requirements and architecture, W3C Working draft 21 July 2000. Available (September 2005): http://www.w3.org/TR/2000/WD-CCPP-ra-20000721/

[13] Common Information Model (CIM) Standard. Available (March 2006): http://www.dmtf.org/ standards/cim/

[14] K.-D. Schmatz. *Java2 Micro Edition Entwicklung mobiler Anwendungen mit CLDC und MIDP*, pp. 29-30. Heidelberg: dpunkt.verlag, 2004.

[15] A. Austermann. Java zum Mitnehmen. *Toolbox – PDAs in Java programmieren, Juli-August 2001*, pp. 6-21.

[16] D. S. Kochnev and A. A. Terekhov. Surviving Java for Mobiles, *IEEE Pervasive Computing, Volume 2, Number 2, April-June 2003*, pp. 90-95.

[17] C. Gordon. Embedded XML eases networking and comms, *Embedded Systems, February 2002*, pp. 33-37.

[18] G. Fuchs, S. Truchat and F. Dressler. Distributed Software Management in Sensor Networks using Profiling Techniques. *Proceedings of 1st IEEE/ACM International Conference on Communication System Software and Middleware (IEEE COMSWARE 2006): 1st International Workshop on Software for Sensor Networks (SensorWare 2006)*, New Dehli, India, January 2006.

[19] Lakale Navigation. Available (September 2005): http://www.iis.fraunhofer.de/ec/navigation/indoor/

[20] F. Dressler. Sensor-Based Localization-Assistance for Mobile Nodes. *Proceedings of 4. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze*, Zurich, Switzerland, March 2005, pp. 102-106.
[21] F.o Dressler and G. Fuchs. Energy-aware Operation and Task Allocation of Autonomous Robots. *Proceedings of 5th IEEE International Workshop on Robot Motion and Control (IEEE RoMoCo'05)*, Dymaczewo, Poland, June 2005, pp. 163-168.
[22] TinyOS. Available (September 2005): http://www.tinyos.net/

Sébastien Truchat received a diplôme d'ingénieur (M.Sc.) from the Ecole Supérieure des Sciences Appliquées pour l'Ingénieur - Mulhouse (ESSAIM) in France in 1997. Afterwards, he joined the Siemens Automation & Drives department as a software developer. In 2000, he joined Computer Networks and Communication Systems Group, Dept. of Computer Sciences, University of Erlangen as a research assistant. He is currently working towards his Ph.D. in the field of software engineering for reconfigurable mobile services and interoperative systems.

Gerhard Fuchs received his M.Sc. in computer science from the University of Erlangen in 2004. Afterwards, he joined the Autonomic Networking Group at the Dept. of Computer Sciences, University of Erlangen as a research assistant. Presently, he is working towards his Ph.D. in the fields of task allocation and resource management in the area of sensor/actuator networks.

Steffen Meyer studied Computer Science at the University of Erlangen-Nuremberg where he graduated with a diploma degree (M.Sc.) in 2002. Since then he has been working in the field of indoor navigation at the Fraunhofer Institute for Integrated Circuits at Erlangen.

Falko Dressler is an assistant professor leading the Autonomic Networking Group at the Department of Computer Sciences, University of Erlangen-Nuremberg. He teaches on self-organizing sensor/actuator networks, network security, and communication systems. Dr. Dressler received his M.Sc. and Ph.D. degree from the Dept. of Computer Sciences, University of Erlangen in 1998 and 2003, respectively. From 1998 to 2003 he worked at the Regional Computing Center at the University of Erlangen as a research assistant. In 2003, Dr. Dressler joined the Networking Group (Chair for Computer Networks and Internet) of Prof. Dr. Georg Carle at the Wilhelm-Schickard-Institute for Computer Science, University of Tuebingen as an assistant professor. In 2004, he joined the Computer Networks and Communication Systems Group of Prof. Dr. Reinhard German at the Department of Computer Sciences, University of Erlangen-Nuremberg. Dr. Dressler co-authored more than 50 reviewed research papers. He was co-chair and PC member for various international conferences (ACM, IEEE, GI, ITG). He is a member of ACM, IEEE, IEEE Computer Society, and GI (Gesellschaft fr Informatik). Dr. Dressler is actively participating in several working groups of the IETF. His research activities are focused on (but not limited to) Autonomic Networking addressing issues in Wireless Ad Hoc and Sensor Networks, Self-Organization, Bio-inspired Mechanisms, Network Security, Network Monitoring and Measurements, and Robotics.