How to take the user on his word

Expressing and interpreting user intention
in a pervasive service environment

Master thesis

Pascal Bihler

Supervisors:
Vasile-Marian Scuturici
Lionel Brunie

Master Recherche d'Informatique
INSA de Lyon

June 17, 2005

Abstract

The introduction of pervasive computing environments in everyday life will not just be a big step for users, but also for application designers. The well defined interaction interfaces "keyboard" and "mouse" will make place for other, more intuitive ways of interaction. It is the challenge for a pervasive system middleware to capture and model the user's intention in a smart way and to solve ambiguousness in the user's expression of a pervasive action. This master thesis introduces the Pervasive Service Action Query Language (PsaQL). PsaQL formalize the description of a user intention using composed pervasive services. The thesis describes a way of translating the user intention into an executable action and propose algorithms performing this translation. Considerations to implement this process are given within the scope of PerSE, a pervasive service environment developed by our research group, together with general evaluation metrics for such algorithms and prototype benchmark results.

Résumé

L'introduction des systèmes pervasifs dans la vie quotidienne n'est pas seulement un grand pas pour les utilisateurs, mais aussi pour les développeurs de logiciels. Les sytèmes pervasifs proposent de remplacer les interfaces d'interaction traditionelles comme ≪ le claviern≫ ou ≪ la souris ≫ par d'autres modes d'interaction plus intuitifs. Le défi pour l'intergiciel d'un système pervasif et alors de capturer et modéliser l'intention de l'utilisateur et d'en résoudre les ambiguïtés. Dans ce cadre, ce mémoire de Master introduit le Langage de Requêtes d'Actions de Services Pervasifs (PsaQL). PsaQL formalise la description de l'intention de l'utilisateur en utilisant des services pervasifs composés. Le travail propose une manière de traduire une intention de l'utilisateur en une action exécutable ainsi que les algorithmes qui réalisent cette traduction. Dans le cadre de PerSE, un Environnement de Services Pervasifs développé par notre équipe, des considérations pour implémenter ce processus sont données, ainsi que les mesure d'évaluation de tels algorithms et les résultats de tests de performances du prototype.

Zusammenfassung

Die Einführung von pervasivem Computing in das tägliche Leben bedeutet nicht nur für die Anwender, sondern auch für die Anwendungsdesigner eine große Herausforderung.Bisher wohlbekannte und präzise defininerte Schnittstellen wie „Tastatur“ und „Maus “ machen Platz für andere, intuitivere Interaktionsarten. In einer pervasiven Dienstumgebung ist es Aufgabe der Middleware, „smart“ die Absicht eines Benutzers zu erkennen und zu modellieren sowie Mehrdeutigkeiten bei der so gegebenen Definition einer pervasiven Aktion aufzulösen. Diese Masterarbeit präsentiert die Anfragesprache für Pervasive Dienst-Aktionen (PsaQL). PsaQL bietet eine Formalisierung, um die Absicht eines Benutzers bei der Verwendung verketteter pervasiver Dienste auszudrücken. Diese Arbeit beschreibt eine Vorgehensweise, diese Benutzerabsicht in eine ausführbare Aktion zu übersetzen und präsentiert Algorithmen, die diese Übersetzung realisieren. Am Beispiel von PerSE, einer von unserer Forschungsgruppe entwickelten Platform zur Verwaltung von pervasiven Diensten, werden Möglichkeiten der Implementierung dieses Prozesses aufgezeigt und Überlegungen zur Performanzmessung solcher Algorithmen gegeben, gefolgt von konkreten Ergebnissen einer solchen Messung an einem Prototyp.

This master thesis has been submitted in a synthesized form as the research group's paper proposal for *The 2005 IFIP International Conference on Embedded & Ubiquitous Computing (EUC'05), Nagasaki City, Japan.*

Typeset of this work using LaTeX:
teTeX-Distribution on darwin,
TeXShop and BibDesk
Lyon, 17/06/2005

Contents

1 Introduction

Ubiquitous computing names the third wave in computing, just now beginning. First were mainframes, each shared by lots of people. Now we are in the personal computing era, person and machine staring uneasily at each other across the desktop. Next comes ubiquitous computing, or the age of calm technology, when technology recedes into the background of our lives. [Wei96]

1.1 Problem statement

"Pervasive" or "Ubiquitous" Computing – as Mark Weiser calls it in his trendsetting paper "The Computer for the 21st Century" [Wei91] – left its cache and starts to become everyday reality. A network of omnipresent, highly embedded computing machines (possibly with limited power and communication resources) seems reasonable in the upcoming century in a manner, that one even does not recognize the presence of these computers anymore. In the way, one perceives (or don't perceives anymore) his intelligent environment, the way one interacts with it has to change as well. The shift is clearly defined from teaching the user how to interact with a computer to teach the computer how to interact with a user. In our vision of pervasive computing, the human-machine-interface becomes so far intuitive and natural that even uninformed people can benefit from their enhanced environment without reading a single line of a user guide. The usage of a pervasive environment should not be intrusive and gain attention, but rather just help the user to live better.

Computing devices will move from reactive actions to handling proactive ones (cmp. figure 1).[1] Today, most machines react to more or less formal commands given by the user, e.g. switching the television channel if the user presses some button on his remote control at the beginning of the advertising block. In the future, knowing the user's dislike for this kind of information service, a proactive television could immediately switch the channel, mute the screen and sound for the duration of the advertisement or present background information about the interrupted broadcasting from the Internet.[2]

In our understanding, "being proactive" means for a computing system "understanding the user's intention", "learning from its action history", and "proposing an action" or "acting". The two first aspects touch the two ways of deriving an action intention, either from an explicit user directive (voice command or similar) or by comparing the context information with the action history. In a pervasive environment bringing different services on different machines together, it should not be the concern of the service developer to care about user intention interpretation, but rather he should rely on a well defined service interface and concentrate on doing the service work well.

At the LIRIS laboratory in Lyon, we develop a pervasive service environment platform called PerSE. This master thesis presents the first step towards interpreting a user intention and describing in a formal way a corresponding action using service composition. The proposed approach is integrated in the PerSE environment.

[1]For sure, the technical designers need to ensure, that a proactive digital world doesn't give the user the feeling of being domineered in his actions by technology.

[2]Maybe, the advertising industry won't be very happy about a proactive system like this, so unfortunately it is unlikely that a big TV supplier will promote a handy solution like that, but the development of the "Werbefee" (translated literally "advertisement fairy") and its successor "Tvoon" in Europe and several similar products in the U.S. show that there is a market for these products [Ohl03], [Unt05].

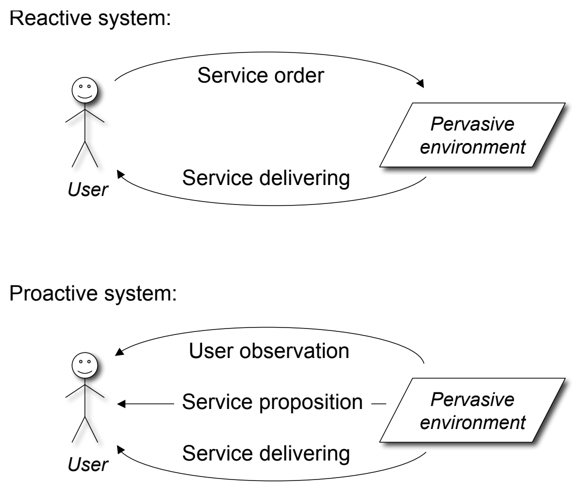

Figure 1: Reactive and proactive systems compared: A reactive system waits for user commands before it delivers a service whereas a proactive system tries to foresee useful services by evaluating context information.

1.2 Principal results of this master thesis

This master thesis addresses the process of transforming a user intention into an executable action in a pervasive service environment. The principal contributions of this work are:

- Modeling of the concepts *partial action*, *action graph*, and *complete action* for PerSE
- Development of *PsaQL*, a formal language to describe the user intention.
- Identifying of *algorithms* transforming a user intention into an executable action.
- Propositions for an *implementation* of the transforming algorithm.
- Considerations about *performance measurements* and *benchmark results* for a prototype.

1.3 Organization of this master thesis

This master thesis is organized as follows: After naming shortly related work in this research area in chapter 2, the challenges and constraints of a pervasive service environment are presented by introducing the main aspects of PerSE in chapter 3. In the following chapter 4, the process of handling the user intention in a pervasive service environment is presented. We introduce an intuitive formal language which can work as an intermediate step between the user's expression of an action intention and the system internal action representation. The translation process between the user's and the machine's interpretation is performed by an algorithm presented in chapter 5. This is followed by some benchmarks guidelines and prototype results in chapter 6, leading to a conclusion and a set of open questions in chapter 7. In the appendix, exemplary results for requests on the PerseBase description interface are given and two algorithms which arise in the context of this master thesis are presented: The first algorithm (appendix B) tries to allude to one of the open questions by reflecting on how the basic information about available services in the network are acquired by the calculating machine and how the local database can be populated. The second algorithm (appendix C) was used to generate test-data for the benchmarking process.

2 Related work

The execution of pervasive applications in a "user-aware" way has been worked out in [SG03] and [SPP+03]. Similar as PerSE does, El-Kathib et al. design in [EKvBS04] a platform trying to maximize user satisfaction while adapting the content of multimedia data with a dynamically estimated path of enchained transcoders. His solution also relies on graph base composition, whereas he does not present a formal language to define the adaptation requests. Currently optimized for fixed network structures like available in the Internet, the platform is already resilient against service failures and hereby maybe as well applicable in a pervasive context.

Other user-oriented systems for managing pervasive environments have been developed, for instance Gaia [RHC+02] or Aura [GSSS02]. Gaia proposes a programming language to construct executable tasks based on the interoperability of services, whereas Aura tries to avoid any interaction with the user and does not present a model for user intention.

The Ninja Environment [GWvB+01] introduced the idea of composing services on distributed adaptation paths, where [BB03] added path optimization considerations for content adaptation. In [VRV05], M. Vallée et al. introduce a system for dynamic service composition in intelligent environments: they are following a related way as PerSE does (see section 3), from a partial action (what they call *abstract plan*) through a composition algorithm to a concrete action, in their words *detailed plan*. They have worked out well the mechanisms for service descriptions and service composition, while they do not present a formal way to express intuitive and computer-interpretable user intention in form of a partial action. They rely on there part on a library of predefined abstract plans, which can be interpreted as a kind of predefined execution history, but this history is not taken directly into account when composing services.

C. Linnhoff-Popien et al. present in [PM94] a language to describe service request in computer networks. This language specifies in details the semantic of a requested service but has not the intention to support service enchainment as multistep action in a pervasive service environment.

Another widely explored field of research inspiring the development of this work are Semantic Web Services, as presented for instance by [MSZ01]. Ontologies as defined by OWL-S [MvH04] can help to create correspondent and valid action graphs and maximize the user satisfaction with a calculated solution. Semantic composition of Web Services is as well introduced by [SvdAB+03], [SdF03], [SPAS03], [VR04], and [Bra05]. The general challenge of matching several demands semantically on web resources based on their descriptions is treated by [NSDM03].

A pervasive environment will be characterized by the availability of application context information [MYA+05]. PerSE will use the contextual information to build an appropriate action graph and to select the best solution as a response to a user intention. Earlier approaches of introducing context based adaption into pervasive environments are presented in [RC03], [Mos03], and [VR04]. When one wants to use public available services in a pervasive network seamlessly, the security is one of the main points. First attempts are made by introducing a smart authentication into PerSE liked worked out in [SPB05].

In the scope of this master thesis, we have payed special attention to work out the meanings of the terms "context" and "adaptability" in pervasive contexts. In general, context-awareness in pervasive environment is a widely explored field of research, but there is not already one best practice solution to incorporate it into real world applications. We published an extended state of the art of approximately 30 pages for this subject at `http://liris.wh4f.de/adaptation.pdf`.

3 PerSE: A Pervasive Service Environment

In this chapter we present the pervasive service environment PerSE developed by our research group. This allows us to introduce in an informal way the key concepts of this paper that will be formally defined in the next chapters.

Pervasive service environments support the interaction of independent services[3] collaborating to perform an intended action. Examples for such services are a filesystem interface, a translating service, a laser pointer acting as an input device or a video projector to visualize information. It is the task of the middleware to connect the services in a pertinent and efficient way. We call this combination of services *complete action*.

When constructing this action, the middleware has to respect the special constraints and features of pervasive systems. Due to wireless communication needs, the underlying transport protocols normally do not allow a transmission speed comparable with wired solutions. Actual protocols are Bluetooth or IEEE 802.11b/g, both *remarkable slower* than the widely used Ethernet 100mps and by far slower than the upcoming gigabit networks. Pervasive devices are normally embedded and mobile, so relying on a mobile power supply. The usage of *battery power* as well as the *limited size* of most pervasive entities leads to a usage of energy and space resources in a responsible manner, therefore *most CPUs in pervasive devices are slower and less powerfull* than the processors of Personal Computers. Finally, the mobile design of pervasive devices imply a *highly dynamic network layout*, devices are appearing and hiding at any moment, the network connections are frequently updated. A middleware managing the applications running in a pervasive environment has to react to all of these constraints when defining an action to perform. The developer of the managing system must take into account these limits when designing the underlying algorithms.

PerSE models such a system and offers the interfaces needed to use the managed services without worrying about the limits of the pervasive environment. Figure 2 shows the basic architecture of PerSE: Each *service* s_i is managed by a base-application called *PerseBase* b_i. This entity is responsible for managing the services it proposes and is capable to construct and start *complete actions*, the system's responses to a user intention. These complete actions are modeled in PerSE as connected graphs of services.

The environment must select the best action as answer to a user intention with respect of the constraints of the whole system (i.e. data transmission speed, reliability, ...). To describe a user intention, we introduce the concept of a *partial action*, that means a description of the action containing (more or less) exactly defined the data source and the data sink and maybe some steps between. The PerseBase in charge has to derive a complete action from the partial action and the knowledge it has about the available services in the network. This process is sketched in figure 3. The challenge of deriving this complete action from a partial user input is studied in this work. Detailed information about this process can be found in chapter 4.

[3]These services can be implemented as Web Services, adapted to the requirements of the environment-middleware, but they needn't be. In fact, any (long-term or short-term) process with inputs and/or outputs and a well defined behavior can be used as a service.

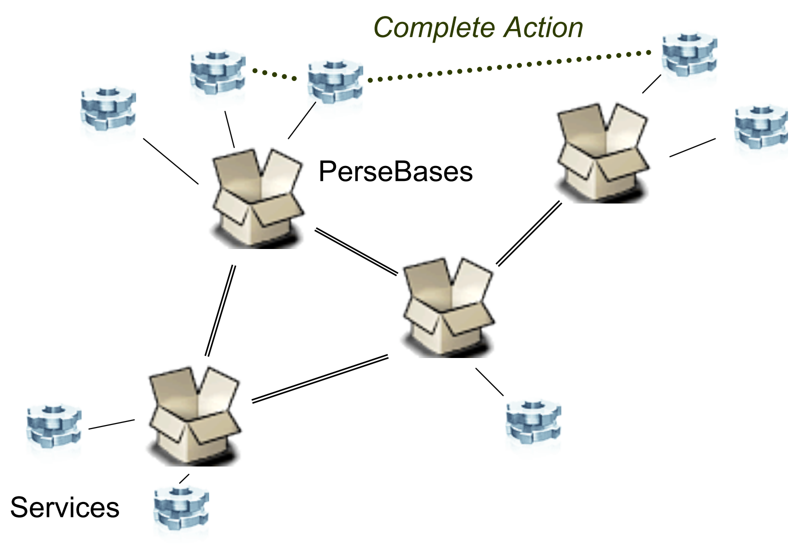

Figure 2: The global architecture of PerSE: Several bases managing services are interconnected. In this example, three services are associated to a complete action.

The process of transforming a user intention into an executable series of commands for the pervasive environment can easily be understood with the following example, which will accompany us through the rest of this work:

A person enters a room, for instance a seminar classroom, and wants to display some presentation about his vacation from his personal notebook. With today technologies and softwares, he has to connect his computer to the local network and copy the file containing the presentation to the local server connected to the video projector. Then he has to find the appropriate program on this machine to present his slides. Alternatively, he can deconnect the projector from the local machine and connect it to his own mobile computer, adjust the screen settings and so forth. A PerSE-*enabled, smart classroom, does not provide the ability to deconnect the local projector cable or to access the room server, because these entities are "invisibly" embedded into the room arrangement. Instead, the user expresses his desire by saying or typing "Show the sunrise presentation on the projector". His notebook, which runs a PerseBase as well and communicates with the local resources via Bluetooth or WLAN, interprets the user command as a partial action with a service delivering a* presentation*, the attribute* sunrise *for this service, and a connection with a service called* projector*. Using the information about the local available services, context information (for instance the room the user has entered) and the action history (maybe the user has already ran a request like this in the past, so if this action is still valid, it can be reused), the PerseBase constructs a graph of all possible (and reasonable) service combinations, each representing a complete action matching the given partial action (see figure 4). From these combinations, an algorithm (see section 5.1) selects the "best" one depending on a given cost-function, e.g. the volume of transferred data. Finally, this complete action is executed: the presentation is displayed on the projector.*

In this chapter we sketched the behavior of PerSE. In the next chapter we will formalize the introduced notions.

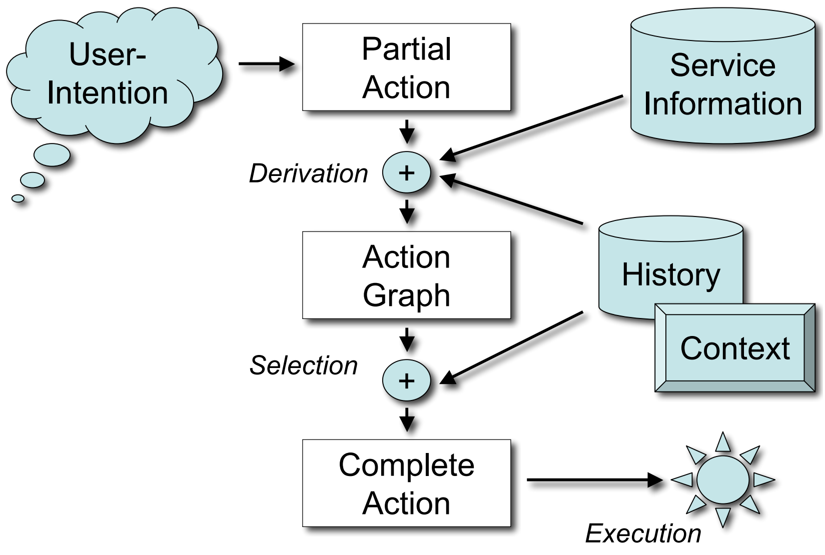

Figure 3: A user intention is transformed into a complete action by selecting the optimal action from all possible complete actions (action graph) matching the user intention, using the knowledge about available services, the context information and the execution history.

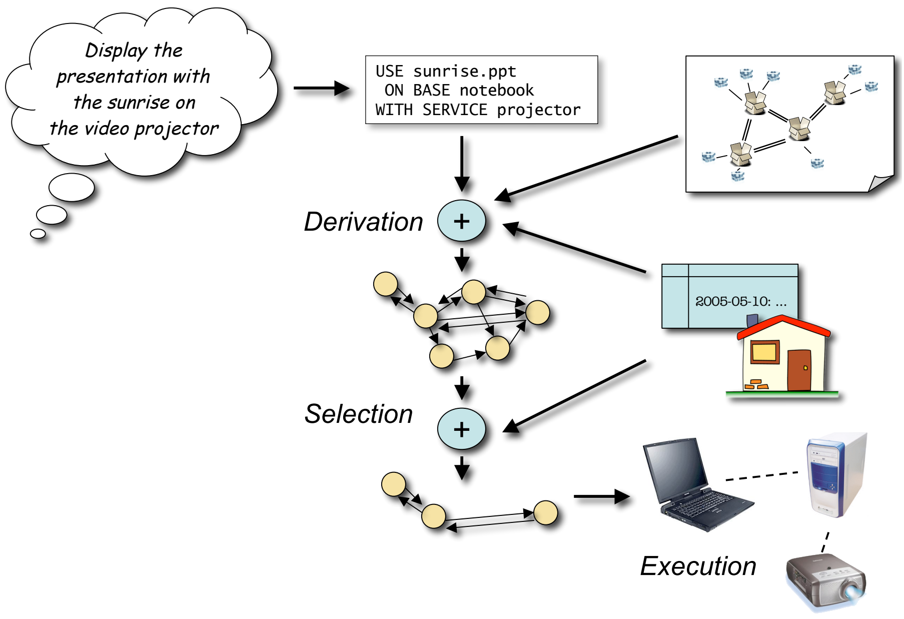

Figure 4: A presentation is displayed with the help of the stationary video projector.

4 Modeling User Intention - PsaQL

When someone wants to use a computing system, most of the time he has a very concrete idea of what he wants to do, even if he does not exactly know how to do it. The action which the system should execute seems very clear for him. Unfortunately, this action is not complete that is it lacks some information to be executable.

An action, as we have modeled it in PerSE, consists of several services corresponding to a connected graph (*complete action*, figure 5 shows the simplest case, a chain of interconnected services). Each service resides on a PerseBase, which means that for a communication, the base information has to be present in a complete action description. In addition, to configure an adequate handling of the data-flow by the participating services, the services need to be adapted to the specific task. Special attributes have to be expressed with each service entity to complete the action description.[4]

It is the challenge for the embedded user-intention-interpretation algorithm to create, based on the little information from the user expressed in a partial action, a complete action which suits best the user intention, maximizing his level of satisfaction. We introduce an object-oriented way of modeling complete actions as well as a language to express partial actions in a system-independent and intuitive way: PsaQL.

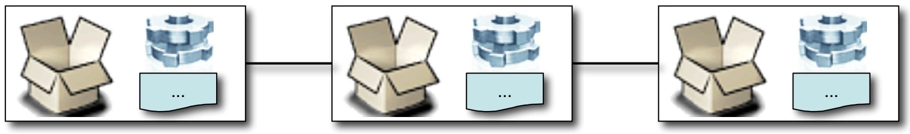

Figure 5: An action consists of several services on PerseBases, called with specific attributes and forming a connected graph. If the action is only partially defined, some of the parameters are not already fixed.

Even if the user interface once will be hidden completely into the background, there will still be the need of expressing a partial action in a formal way. This representation can be used directly to express an intention or as an exchange format used by parts of the PerSE middleware system. Therefore we decided to define a formal language, PsaQL (Pervasive Service Action Query Language), which expresses in an intuitive way the user intention. PsaQL plays a similar role in the PerSE-enabled pervasive environment as SQL [Dat86] does for accessing relational database management systems. PsaQL also looks similar to SQL on the first glance:

```
USE sunrise.ppt ON BASE notebook WITH SERVICE projector
```

In this example, the user intends to display a sunrise presentation on a projector (see chapter 3).[5]

[4]In the example introduced in chapter 3, *sunrise.ppt* would be the attribute for the service *filesystem* in order to advise the system to deliver the content of this specific file.

[5]The user already has the knowledge, that his presentation is stored in a file named "sunrise.ppt" on his computer accessible as "notebook", and that the video projector is named "projector". If the user is not sure about these parameters, his request could be expressed using "LIKE"-statements.

The language-definition of PsaQL in BNF is:

```
<partial_action> ::= USE <action_part> [<ext_action>]
<ext_action> ::= WITH <action_part> [<ext_action>]

<action_part> ::= <attr_constr_def>[ FOR <service_constr_def>]
                                   [ ON <base_constr_def>] |
                  <service_constr_def>[ ON <base_constr_def>] |
                  <base_constr_def>

<base_constr_def> ::= BASE <base_constraint>[ AS <name>]
<base_constraint> ::= <name> | LIKE "<partial_name>"

<service_constr_def> ::= SERVICE <service_constraint>[ AS <name>]
<service_constraint> ::=  <name> | LIKE  <partial_name>

<attr_constr_def> ::= (<name> | LIKE <partial_name>)[ AS <name>]

<name>  ::= {<‘a’-‘z’, ‘A’-‘Z’, ‘0’-‘9’,‘_’>}
<partial_name>  ::= {<‘a’-‘z’, ‘A’-‘Z’, ‘0’-‘9’,‘^’,‘$’,
                           ‘(’,‘)’,‘[’,‘]’,‘.’,‘+’,‘*’,‘?’, ...>}
```

To give a second example, another action (the command to send some file from a PDA-computer as e-mail) could be expressed as follows:

```
USE mail.eml ON BASE pda
    WITH SERVICE LIKE "mail$" AS mail
      ON BASE LIKE "mailgateway" AS gateway
```

The LIKE-statement currently advises the algorithm in charge to interpret the associated character chain as a regular expression, which will be applied to the entity's name as well as to keywords describing the service/base or a literal description. The AS-statement is used to mark specific parts of the request in order to identify them in the resulting complete action. In a future version of PsaQL, the language will be extended to allow by this referencing inside a statement.

Based on this description of a partial action, an action graph of all possible solutions is created and finally, a complete action is selected (cmp. figure 3). This solution action can be executed by the pervasive environment. With PsaQL, actions can be defined in any granularity from a very ambiguous expression up to an almost complete action. We have defined an XML file format containing all the information needed to execute a complete action (see chapter 5.3). This file can be used to exchange the action in a platform-independent way between different modules of the middleware.

A prototype to demonstrate the process of creating a complete action based on a partial action can be found at `http://liris.wh4f.de/perse`.

5 From user intention to user satisfaction

5.1 Translating a partial action into a complete action

We present in this chapter an algorithm to translate a partial action into a complete action (cmp. figure 3). A *partial action* is a formal representation of a user intention whereas a *complete action* is a connected graph of services, representing the best combination of services to satisfy the user intention. The translation process consists of three steps:

1. Translate the user-input (given in PsaQL) into an internal model of a partial action
2. Expand the partial action to an *action graph* using the service description database, the action history, the context information and heuristic strategies
3. Select the best solution as complete action

Algorithm 1 The abstract translation algorithm

```
action onRequestReceived(user_intention)  // General description of the translation algorithm
1: partial_action = parse(user_intention)
2: action_graph = create_actiongraph(partial_action)
3: action = select_best_action(action_graph)
4: return action
```

With some mathematical formalization, we can define step two and three of this algorithm:[6]

Definition 1 (Partial Action) *Let B be the set of bases, S the set of services and $S(b)$ the set of available services on a base b, where $S(b)$ is equal to S when $b = \perp$.*[7] *A* partial action *p, the formal expression of a user intention, can be modeled as a list of 2-tuples:*

$$p = (e_1, \cdots, e_n) \mid e_i = (b_i, s_i) \text{ with } b_i \in \{\perp\} \cup B; s_i \in \{\perp\} \cup S(b) \quad \forall i : (b_i \neq \perp) \vee (s_i \neq \perp) \tag{1}$$

Definition 2 (Service Graph) *Let $E = \{\epsilon = (b, s) \mid b \in B, s \in S(b)\}$ be the set of all available services in a pervasive service environment and $I(E) \subseteq E \times E$ the set of all possible service interactions within this environment.*[8] *Then we can define a service graph in a part $\mathcal{E}$ of the pervasive service environment as*

$$G_{\mathcal{E}} = (\mathcal{E}, \mathcal{I}); \ \mathcal{E} \subseteq E; \ \mathcal{I} \subseteq I(\mathcal{E}) \tag{2}$$

The set Γ_E of all valid service graphs in E is defined as:

$$\Gamma_E = \{(\mathcal{E}, \mathcal{I}) \mid (\mathcal{E} \subseteq E, \mathcal{I} \subseteq I(\mathcal{E}))\} \tag{3}$$

Definition 3 (Connected Service Graph) *A service graph $g = (\mathcal{E}, \mathcal{I})$; $\mathcal{I} \subseteq I(\mathcal{E})$ is called* connected *if and only if*

$$\forall \epsilon_0, \epsilon_n \in \mathcal{E}, \ \epsilon_0 \neq \epsilon_n : (\epsilon_0, \epsilon_n) \in \mathcal{I} \vee (\exists \epsilon_1, \ldots, \epsilon_{n-1} : (\epsilon_i, \epsilon_{i+1}) \in \mathcal{I}; \ (i = 0, 1, \ldots, n-1)) \tag{4}$$

[6]To simplify the model, we do not consider attributes in the following section, they can be easily added lately.
[7]$\perp$ represents "undefined".
[8]The interoperability between services is not studied here.
We assume that $(\epsilon_1, \epsilon_2) \in I(E) \Leftrightarrow ((\epsilon_1, \epsilon_2) \in E \times E) \wedge (\epsilon_1$ is interoperable with $\epsilon_2)$.

Definition 4 (Solution) *A graph $g_p^E = (\mathcal{E}, \mathcal{I})$; $\mathcal{E} \subseteq E, \mathcal{I} \subseteq I(\mathcal{E})$ is called* solution *for a given partial action p in a pervasive service environment E if and only if*

$$g_p^E \in \Gamma_E \tag{5}$$

$$g_p^E \textit{ is connected} \tag{6}$$

$$\forall e = (b, s) \in p \; \exists \epsilon = (\beta, \sigma) \in \mathcal{E} \textit{ with } \begin{cases} \beta = b & \textit{if } s = \perp, \\ \sigma = s & \textit{if } b = \perp, \\ (\beta = b) \wedge (\sigma = s) & \textit{otherwise.} \end{cases} \tag{7}$$

Definition 5 (Action Graph) *Let $\mathcal{S}_p^E$ be the set of all solutions for p in a pervasive service environment E. An* action graph *$A_p^E \in \Gamma_E$ of a partial action p in the pervasive service environment E is a connected service graph containing all solutions for p:*[9]

$$A_p^E = (\bigcup\nolimits_{(\mathcal{E},\mathcal{I}) \in \mathcal{S}_p^E} \mathcal{E}, \bigcup\nolimits_{(\mathcal{E},\mathcal{I}) \in \mathcal{S}_p^E} \mathcal{I}\ ) \tag{8}$$

Definition 6 (Complete Action) *Let $C(g, \gamma)$ be the function calculating the cost of a solution g when it is executed in a context γ. Then the* complete action *c_p^E for a given partial action p in a service environment E is defined as:*

$$C(c_p^E, \gamma) = \min\{C(g_p^E, \gamma) \mid g_p^E \in \mathcal{S}_p^E\} \tag{9}$$

Let $H(p, E, \gamma)$ be the function fetching from the execution history the complete action for a given partial action p and a pervasive service environment E in a context γ. Then, with these definitions and declarations, we propose the algorithm 2 transforming a user intention p into a complete action c_p^E.[10] Starting with an action graph (representing a solution for p), this algorithm removes step by step the connections (see line 11 of algorithm 2). When a service graph derived by this has no longer any successors which are solutions (line 16), this service graph is a possible candidate for the complete action (line 17). The candidate with the minimal cost is selected as solution of the algorithm.

The complexity of this algorithm is exponential, but it finds always the optimal action for a given partial action. For small pervasive service environments (small number of bases and services), this algorithm calculates in a reasonable time the best possible complete action for a given partial action. Nevertheless, this solution is not appropriate for pervasive service environments of bigger extend. In order to solve the query in polynomial time, it makes sense to define stricter constraints or to introduce heuristic approaches.

One such heuristic approach(algorithm 3) bases on the following coloring of the nodes:

- *Black nodes* ϵ_B: $\forall \epsilon_B = (\beta, \sigma) : \exists (\beta, \sigma) \in p$
- *Red nodes* ϵ_R: $\forall \epsilon_R = (\beta, \sigma) : \exists (\beta, \perp) \in p \vee \exists (\perp, \sigma) \in p$
- *White nodes* ϵ_W: all other nodes

[9] In some rare cases (when $\forall (b, s) \in p : \; b = \perp$) it can happen, that A_p^E is not unique. If so, algorithm 2 (see below) has to be executed for each $(A_p^E)_i$.

[10] A_p^E can be directly derived from $(E, I(E))$.

Algorithm 2 Exhaustive Translation Algorithm

```
1: $c = H(p, E, \gamma)$  // *try to find a solution in history*
2: **if** $c \neq \perp$
3:    $c_p^E = c$
4: **else**
5:    fifo A, fifo S
6:    push(A,$A_p^E$)
7:    **while** A $\neq \perp$
8:       $(\mathcal{E}, \mathcal{I}) = g =$ shift(A)  // *take a service graph from the beginning of the list*
9:       is_leaf = **true**
10:      **for each** $\iota \in \mathcal{I}$
11:         $\mathcal{I}' = \mathcal{I} - \{\iota\}$  // *remove connection $\iota$ from $\mathcal{I}$*
12:         $\mathcal{E}' = \bigcup_{(\epsilon_1, \epsilon_2) \in \mathcal{I}'} \{\epsilon_1, \epsilon_2\}$  // *keep only connected elements*
13:         **if** $g' = (\mathcal{E}', \mathcal{I}')$ is solution for $p$
14:            push(A,$g'$)  // *remember this successor*
15:            is_leaf = **false**
16:      **if** is_leaf
17:         push(S,$g$)  // *$g$ is a solution candidate*
18:   $g =$ shift(S)
19:   **for each** $g' \in$ S   // *find minimal solution*
20:      **if** $C(g', \gamma) < C(g, \gamma)$
21:         $g = g'$
22:   $c_p^E = g$
```

Algorithm 3 Heuristic Translation Algorithm

```
1: $c = H(p, E, \gamma)$  // *try to find a solution in history*
2: **if** $c \neq \perp$
3:    $c_p^E = c$
4: **else**
5:    list $F$  // *black nodes*
6:    list $R$  // *red nodes*
7:    **for each** $\epsilon = (\beta, \sigma) \in E$
8:       **if** $\exists e = (b.s) \in p : (b = \beta) \wedge (s = \sigma)$
9:          push$(F, \epsilon)$
10:      **else if** $\exists e = (b.s) \in p : ((b = \beta) \wedge s = \perp) \vee ((s = \sigma) \wedge b = \perp)$
11:         push$(R, \epsilon)$
12:   array $D$  // *calculate the minimal distance*
13:   **for each** $\epsilon_f \in F$
14:      **for each** $\epsilon_r \in R$
15:         $d =$ length(dijkstra($(E, I(E))$,$\epsilon_f$,$\epsilon_r$))
16:         **if** $d < D[\epsilon_r]$
17:            $D[\epsilon_r] = d$
      // *elements of partial action covered only by red nodes:*
18:   $p' = \{e = (b, s) \in p \mid (b = (\perp) \vee (s = \perp) \wedge (\nexists \epsilon_f = (\beta, \sigma) \in F : (b' = \beta) \vee (s' = \sigma))\}$
19:   sort($R$,$D$)  // *sort red nodes by distance*
20:   **for each** $\epsilon_R = (\beta, \sigma) \in R$
21:      **if** $\exists e = (b, s) \in p' : (b = \beta) \vee (s = \sigma)$
22:         push$(F, \epsilon_r)$  // *color the node black*
            // *removed satisfied parts from the partial action:*
23:         $p' = p' - \{e' = (b', s') \in p' \mid (b' = \beta) \vee (s' = \sigma)\}$
24:   $c_p^E =$ MCST($(E, I(E))$,$F$)  // *calculate Minimum Cost Spanning Tree on black nodes*
```

We assume that each node wears just one color, with a hierarchy given as: *Black* > *Red* > *White*. By this, we obtain a partion of E.

The solution we want to find includes all black nodes, some of the red nodes (so that there is for every $e \in P$ an appropriate node in the solution) and maybe some white nodes. To find the "best" red nodes, we use the following heuristic (see also figure 6):

First we calculate using the Shortest Path Algorithm of Dijkstra [Man89, p. 204] for each red node ϵ_R the distance to all black nodes ϵ_B by using a modified cost function $C_I((\epsilon_R, \epsilon_B), \gamma)$. Beginning with the red node of the shortest distance, we examine all red nodes in order of their distance: If there is a corresponding query part e in the partial action p, which is not already covered by any black node, we color the node black. We repeat this, until every part of p is covered with a black node. The remaining red nodes are colored white. Then, using the Minimum Cost Spanning Tree Algorithm [Man89, p. 208] we extract from $(E, I(E))$ the graph containing all black nodes as a complete action.

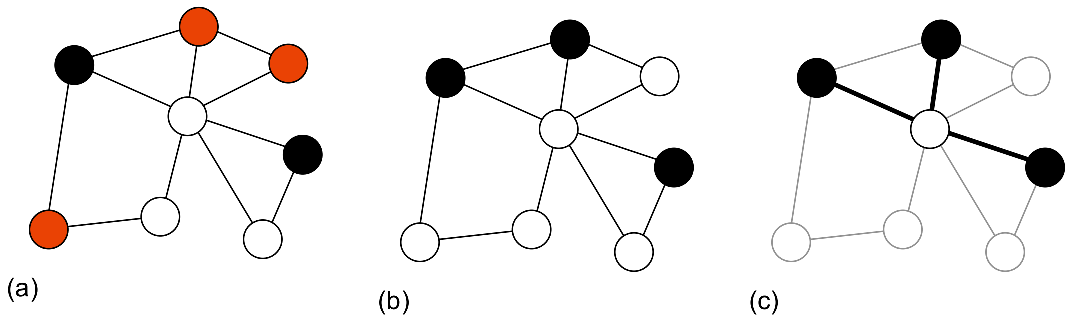

Figure 6: In this example, the partial action p contains three elements. Two of them define exactly a corresponding (black) node in the action graph and one matches to three different (red) nodes (a). From the red nodes, the one with the nearest distance to a black node is selected and colored black, the others are colored white (b). In a final step, the Minimum Cost Spanning Tree containing all black nodes is calculated (c).

5.2 Modeling and implementation of action graphs

To represent the action graph and a complete action, we need a suitable data structure. Each used service is connected with other services, therefore we model the action graph as well as its thinned out version, the complete action, as a set of channels, connecting output- and input-ports of the participating services.

Describing a complete action with n participating services as a set of $(n-1)$ channels connecting $2*(n-1)$ input/output ports with the associated services result in the description of $2*(n-1)$ services, i. e. a redundancy of $(n-2)$ service-descriptions. To avoid useless data overhead, an object-orientated modeling is suitable where logical associations result in references to the same object. In addition, each object has by definition its own identity and does not need to be marked further to be distinguished from other entities.

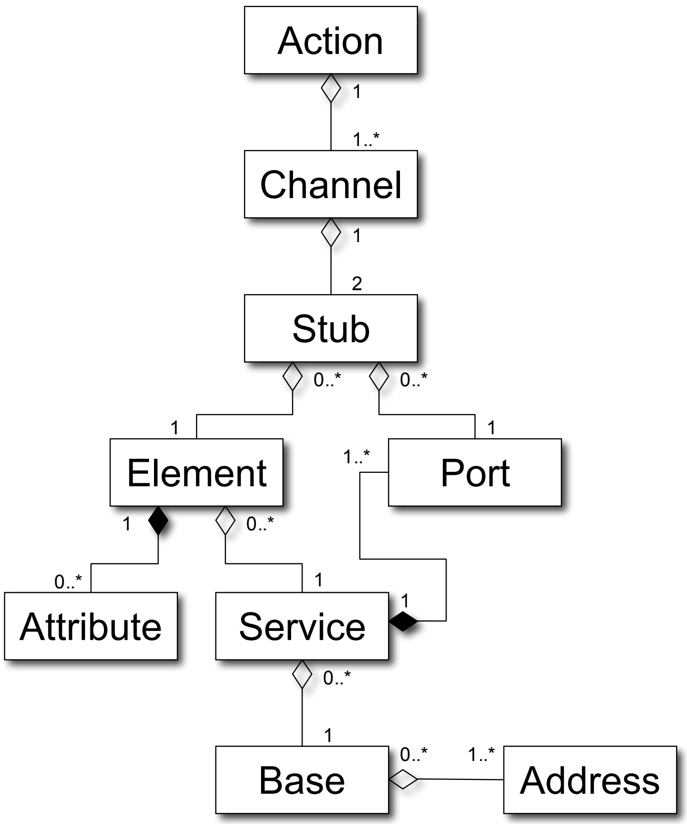

Figure 7: Class diagram for the internal representation of action graphs and complete actions.

To describe a complete action, we use the object model of figure 7. The utilized classes are:

- *Address* - Combination of a type and a valid address-string (e.g. `"ipv4"`/`"172.20.0.9:8080"`)
- *Base* - Represents a PerseBase, has one or more *Addresses* and manages *Services*
- *Service* - A service published on a *Base* proposing *Ports* for data input and output
- *Port* - Representation for data-flow input and output of a *Service*. Contains information about the type of data-flows accepted/produced and the direction of the data-flow (in/out)
- *Element* - Combination of a *Service* with adaptation information (*Attributes*)
- *Attribute* - A string transmitted to a service to adapt it to the specific needs of a PerSE-action
- *Stub* - Combination of an *Element* with a service-*Port*
- *Channel* - The two *Stubs* forming a data-flow between two services
- *Action* - A collection of *Channels*

The objects which form the action graph for a given partial action as well as the resulting complete action are created based on a database connected to the system performing the translation process.[11] This database contains descriptions of available PerseBases, services and the description of these services. Where such a database initially just contains information about local available services, each PerseBase supports a HTTP-based interface to access this information. By requesting this information in an intelligent way it is possible to extend the local a database until it contains all the data needed to build the action graph.[12]

The PerseBase description interface contains three kinds of queries:

- *allObjects* – Returns a list of all known neighbor-bases (including the requested PerseBase itself) as well as their connection information (IP-address for instance) and other meta-data
- *services* – Returns a list of local available services accompanied with meta data like name, keywords etc.
- *serviceDescription* – Returns a detailed description of a given service containing information about ports, treatable data types and so on

To distinguish bases, services and service-ports internally in a unique way and to avoid name collisions between these components, 128-bit Global Universal Identifiers (GUIDs) are used. They are oriented on Microsoft COM GUIDs and based on the Universally Unique Identifiers (UUID) used in DCE RPC. The identifiers are normally represented by 32 hexadecimal digits in five groups like {`e531bc65-f2d1-4d47-8a6e-d96d80e9fb65`}. A decentralized algorithm which assure uniqueness is used to create this identifiers [Box97].

The results of the described interface requests are transmitted using specific XML schemas. Examples for possible requests results can be found in annex A.

The interoperability of services is determined using the `accepted_data_types` information of a service description. In near future, an ontology based on OWL ([MvH04], [MSZ01]) will be established in PerSE to support inheritance of data-types (e.g. define `text/html` as a child of `text/plain`) in by this a service-combination can be based on super-types and sub-types.

5.3 Complete action representation

When a complete action is selected from the action graph in respect of the selection constraints defined by the context information and the historical knowledge, the complete action has to be executed. To exchange the modeled complete action with other, independent entities, for instance an action execution module or a module managing the execution history, we have defined an XML file format containing all the information from the internal model. The XML file is intend to represent complete actions, but can describe action graphs as well. By this it can serve as an intermediate exchange format when splitting up the translating algorithm into several independent parts being executed independently, e.g. when a user feedback needs to be included into the solution finding process.

[11]Normally, this will be a PerseBase

[12]This is part of the concrete implementation, two possibilities to do this will be introduced in chapter 6. In addition, an independent approach of filling the local database proactive is sketched in annex B.

The XML Document Type Definition we have defined is illustrated in the following example (this could be the result for the partial action introduced as example in chapter 3):

Complete action expressed in XML

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<action id="action_1117551504-11e33">

  <channel id="channel_dab">
    <out element_id="element_1caf"  port_id="port_1327" />
    <in element_id="element_24b4"  port_id="port_25ee" />
  </channel>

  <element id="element_24b4" service_id="service_c67">
    <attribute id="attribute_597">sunrise.ppt</attribute>
  </element>

  <element id="element_1caf" service_id="service_957" />

  <service id="service_957" name="beamer" base_id="base_1492" />
    <!-- This is a projector for VGA information -->
  <service id="service_c67" name="fs" base_id="base_11ec" />
    <!-- Standard filesystem service -->

  <base id="base_1492" name="rlaurini">
    <!-- This computer is fixed in classroom 101 -->
    <address type="ip4">172.20.0.72:8080</address>
  </base>
  <base id="base_11ec" name="Notebook">
    <!-- Laptop used by A. M. -->
    <address type="ip4">172.20.0.9:8080</address>
  </base>

  <port id="port_25ee" service_id="service_c67" direction="out">
    <!-- Read file -->
    <type>text/plain</type>
    <type>application/binary</type>
  </port>
  <port id="port_1327" service_id="service_957" direction="in">
    <!-- Save file -->
    <type>video/avi</type>
    <type>video/mpeg</type>
    <type>video/vga</type>
    <type>application/powerpoint</type>
    <type>application/binary</type>
  </port>

</action>
```

5.4 Evaluation metrics for a pervasive service environment

To achieve the goal of seamless integration of the pervasive computing system into the everyday life environment of a user, it is important that the user percept nearly no delay between expressing his intention and receiving the result of it by the executed action. This leads to the goal of short *execution time* for any action algorithm. This is directly related with the *user satisfaction*, the most important aspect which is nevertheless difficult to measure.

The execution time of the transforming process from a partial action to a complete action is just little connected with the complexity of the algorithm, in a pervasive network with just a couple of services data transmission costs are much more important. This defines the challenge of minimizing the *network data exchange*, viz minimizing the *size of transferred service descriptions* and the *transmission distance*. All these parameters should not blow the execution of the translating process up when increasing the number of known services and the length of the requests, that means *scalability* as a development goal.

Other important issues, when implementing the algorithms on a pervasive PerseBase on a smart device like a PDA or a wristwatch are the *minimization of CPU-usage* to save energy resources as well as minimizing the *memory usage* to save little memory resources.

In summary the following domains for measuring algorithms translating a user intention into a complete action have been identified:

- *User satisfaction* as the most important goal of algorithm design in a pervasive environment.
- *Execution time* as the factor having the biggest impact on the user satisfaction.
- *Network data transmission* as having an important impact on the execution time. Beside the *network speed* which cannot be influenced by the design of the translating algorithm, the *size of the transferred services* and the *transmission distance* constitute this value.
- *Scalability* of the algorithm when increasing the *number of available services* or the *length of the query* (see chapter 6).
- *Pervasive Computing Constraints* as an *accountable usage of CPU and memory resources.*

The process of transforming a partial action into a complete action relies on a database containing descriptions of available services in the pervasive environment. This influences directly the amount of the transferred data. Different possibilities to transfer the service description into the local database before being accessed by the translating algorithm and there impacts on the algorithmic performance are discussed in the following chapter.

6 Benchmark Results

To test our prototype implementation of a simple transforming algorithm, we evaluated the volume of the transferred data when varying the number of available PerseBases from 10 to 1000 in steps of 10 bases. To examine the *scalability* of the solution, we introduced a *scalability factor* which calculates the average distance covered by transmitted data:

$$f_{scal} = \frac{\sum_{msg} size(msg) * distance(msg)}{\sum_{msg} size(msg)}$$

A first implementation ("Algorithm A") fetches the total available service description at the beginning and performs the creation of a complete action on base of these descriptions. This implementation corresponds in its benchmark results to the exhaustive algorithm presented in section 5.1. In a second implementation ("Algorithm B") we have tried to minimize the amount of transferred data. To reach this goal, we introduced some constraints for the implementation of this prototype. The selection of the complete action from the action graph is done on-the-fly and the implementation fetches at the same time the service descriptions from the network, just until the algorithm found a solution. Another restriction of this second implementation is the composition of services by linear concatenation.

The test environments have been created with a uniform distribution of n PerseBases in a region of fixed size using the PlnGen algorithm explained in appendix C.[13] For each testbed size, three independent testbeds have been created to get average results. For each node, uniformly between two and five services have been assigned. The type of each service has been estimated normally.[14] In the same way, for each service-type the input- and output types have been selected.[15] The connections between the simulated bases try to model a heterogeneous pervasive network with short distance and long distance connections. We analyzed queries of the type "`USE BASE` x `WITH BASE` y`...`" with a length varying from two to ten enchained entities. For each query length, two queries have been defined randomly and the average results of executing this queries in the testbeds of a fixed size have been examined.

The results of our first test runs show that the currently implemented algorithm reacts very sensitively to an increasing of the number of available PerseBases. It is at most the aspect of the increasing distance between the requesting and the service-providing PerseBase raising the value exponentially. A variation of the query size has nearly no impact on the scalability factor. In all cases, algorithm B is by far better than algorithm A (which represents an upper border for algorithm B). The results are shown in figure 8 and 9.

The comparison of the two implemented algorithms shows that there remain a lot of possibilities to optimize the way of translating a PsaQL request into a complete action. The research is just started at this point, further improvements will be achieved by taking into account the execution history and the the context information, both currently not implemented.

[13] Testbed-parameters: regionSize = 1000×1000, nodeCount = $\{10, 20, \ldots, 1000\}$, minimalConnNumber = 4, standardDeviation = 3.0, backConnDistance = 20.

[14] Using the formula $f_i(n) = abs([random\_normal(1, 0, n/3)])\ mod\ n$ where n is the number of service-types (100)

[15] $n = 10$

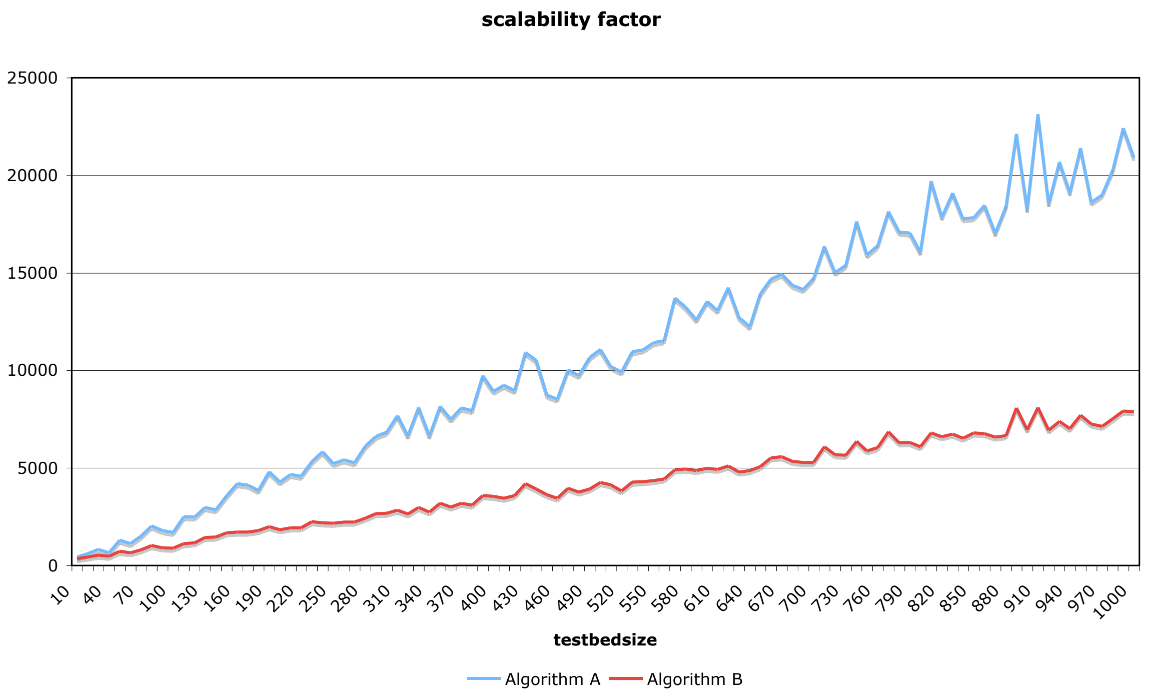

Figure 8: The progression of the scalability factor as introduced in the text when varying the testbed size.

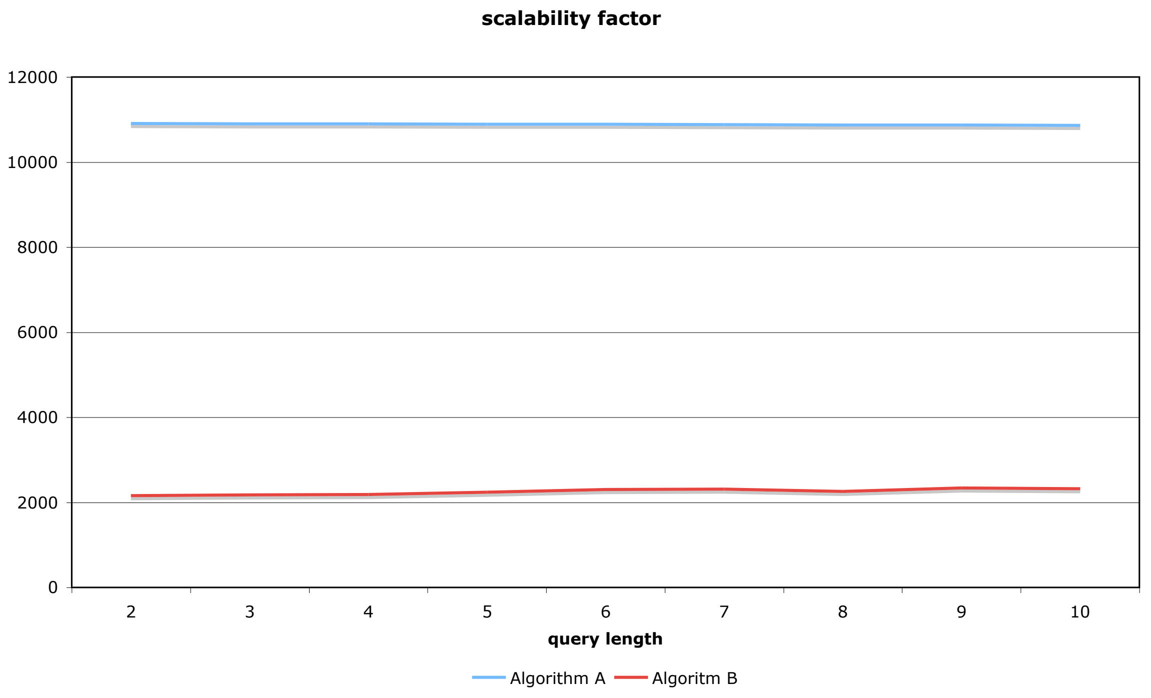

Figure 9: The variation of the scalability factor when increasing the query length.

7 Conclusion and open issues

7.1 Summary

This work has presented a strategy and a methodology to take the user intention into account when composing service-based actions in a pervasive service environment. It introduced PsaQL, as far as we know the first language to express user intention (called partial actions) in a pervasive service environment. We outlined algorithms extending this partial action to a executable graph of services (complete action) using the service descriptions, the context information and the execution history.

We presented an object-oriented modeling of complete actions as well as the definition of an interchangeable data format for these structures. The enchainment of several pervasive services located on different bases is modeled using virtual channels between the input ports and the output ports of the services, the final result of the calculation is expressed using an XML format.

Finally, some metrics supposed to model the user satisfaction has been worked out as reasonable measurement categories for algorithms creating complete actions for pervasive service environments on the basis of a user defined partial action. The work identified the scalability, based on data transmission time and distance as the most influencing part of this benchmark and therefore, a prototype implementation of such an algorithm was tested by measuring the scalability by varying the size of the simulated pervasive service environment and the scope of the user intention. The benchmark shows amongst others, that the introduction of translation limits can help to design implementations with by far better scalability results.

7.2 Contribution

- Modeling of the concepts *partial action*, *action graph*, and *complete action* in the scope of PerSE, a non-centralized pervasive service environment connecting the services provided on heterogeneous devices.
- Development of *PsaQL*, to the best of our knowledge the first formal language to describe service action queries in a pervasive environment in an intuitive and flexible manner.
- Identifying of *algorithms* transforming a user intention into an executable complete action using the service descriptions, the execution history, and context information.
- Propositions for an *implementation* of the transformation algorithm by presenting an object-orientated model to describe action graphs and complete actions, an XML-based exchange format, and examples for communication APIs.
- Considerations about *performance measurements* and *benchmark results* for a prototype implementation.

7.3 Perspective

Following this research work, a couple of issues remain open to be developed in future time. Capturing the user intention using PsaQL will not stay the last conclusion of wisdom, more seamless ways like voice or gesture recognition will be developed. Independent of the kind of expression the user selects to communicate his intention, PsaQL will be an easy interface between the intention capturing module and the module developing the complete action to execute. The Pervasive Service Action Query Language itself can be further improved by adding more language features as references inside or outside a request. Nevertheless, to protect the easy-to-read way of the language it has to be kept as simple as possible.

Currently, the context based selection process in PerSE is not completely elaborated, but the contextual platform hammered out by Ejigu in [ESB05] introduces the main principles *context filter*, *reasoning and data mining* and *decision module*. The knowledge from this platform will be introduced into the user intention processing of PerSE.

Similarly, the implemented algorithm will be further developed to take the execution history into account. This is intended to lower significantly the execution time of the translation algorithm, as the typical actions in a pervasive service environment (like the exemplary presented action to show a presentation from a mobile device on a fixed projector) are frequently reexecuted. We estimate that about 90 % of the actions can completely or partially base on already executed actions stored in the history, and just a small amount of 10 % has to be completely new calculated.

Benchmark results in chapter 6 show that the way how to request service and base descriptions is a screw to adjust user satisfaction. Different algorithms to access and update service descriptions need to be developed and compared for certain use cases. Possible parameters are: Proactive or reactive service description demands, complexity of transmitted descriptive information, time of validity, caching, hierarchical ordering of services and so on. Appendix B presents exemplary a way of distributing service descriptions proactive without congesting the pervasive network with useless information.

When a complete action is built, it has to be executed in the pervasive service environment. Further research will develop a module to execute complete actions reliably and efficiently.

As mentioned already in chapter 2, security will be one of the hottest subjects in further PerSE-development. The current presumption that all services can be used by everyone cannot be hold in real world application. A system of authentication and authorization based on roles and access control lists will be implemented in a future work. Encryption of the exchanged data can currently be easily performed by using the HTTPS-Protocol for data transfers. Nevertheless, it rests to discuss until which point additional computing effort for encryption is reasonable in a pervasive application and where it is more intelligent to transfer data unencrypted to gain a less resource consummating data transfer.

Albeit the further research which has to be done until a pervasive service environment like presented in this work will be used in everyday live, with this master thesis a step further in this direction is done. Based on the presented transit from a diffuse expressed user intention by the way of a formal representation of a complete action to the expression and finally execution of a complete action, the pervasive service environment will be the invisible system adapting to the specific situation and doing what the user intends to his satisfaction.

References

[BB03] Sven Buchholz and Thomas Buchholz, *Adaptive content networking*, ISICT '03: Proceedings of the 1st international symposium on Information and communication technologies, Trinity College Dublin, 2003, pp. 213–219.

[Box97] Don Box, *Essential com*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997, Foreword By-Grady Booch and Foreword By-Charlie Kindel.

[Bra05] Ingo Braun, *Semantic web services: A logic-based framework to dynamic and approximate composition by constraints*, Master-Thesis INSA de Lyon, unpublished, June 2005.

[Dat86] C. J. Date, *A guide to the sql standard*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[EKvBS04] Khalil El-Khatib, Gregor v. Bochmann, and Abdulmotaleb El Saddik, *A distributed content adaptation framework for content distribution networks (online: http://beethoven.site.uottawa.ca/dsrg/PublicDocuments/Publications/ElKh04c.pdf)*, 2004, last checked: 2005-06-15.

[ESB05] Dejene Ejigu, Vasile-Marian Scuturici, and Lionel Brunie, *Neighbourhood based architecture for context-aware platform in pervasive computing*, unpublished, 2005.

[GSSS02] David Garlan, Dan Siewiorek, Asim Smailagic, and Peter Steenkiste, *Project aura: Toward distraction-free pervasive computing*, IEEE Pervasive Computing **1** (2002), no. 2, 22–31.

[GWvB$^+$01] Steven D. Gribble, Matt Welsh, Rob von Behren, Eric A. Brewer, David Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R. H. Katz, Z. M. Mao, S. Ross, B. Zhao, and Robert C. Holte, *The ninja architecture for robust internet-scale systems and services*, IEEE Intelligent Systems, vol. 35, Elsevier North-Holland, Inc., New York, NY, USA, March 2001, pp. 473–497.

[Man89] Udi Manber, *Introduction to algorithms: A creative approach*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[Mos03] Soraya Kouadri Mostéfaoui, *A Context-Based Services and Discovery and Composition Framework for Wireless Environments*, Proc. of the 3rd IASTED International Conference on Wireless and Optical Communications (WOC'2003) (Banff, Canada), 14-16 July 2003, pp. 637–642.

[MSZ01] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng, *Semantic web services*, IEEE Intelligent Systems, vol. 16, IEEE Educational Activities Department, March 2001, pp. 56–53.

[MvH04] Deborah L. McGuinness and Frank van Harmelen, *OWL Web Ontology Language Overview (online: http://www.w3.org/TR/2004/REC-owl-features-20040210/)*, 2004-02-10, last checked: 2005-06-15.

[MYA$^+$05] Jianhua Ma, Laurence T. Yang, Bernady O. Apduhan, Runhe Hunag, Leonard Barolli, and Mokoto Takizawa, *Towards a smart world and ubiquitous intelligence: A walktrough from smart things to smart hyperspaces and ubickids*, International Journal of Pervasive Computing and Communications, vol. 1, Troubador Publishing Ltd., March 2005, pp. 53–68.

[NSDM03] Tommaso Di Noia, Eugenio Di Sciascio, Francesco M. Donini, and Marina Mongiello, *A system for principled matchmaking in an electronic marketplace.*, WWW, 2003, pp. 321–330.

[Ohl03] Lars Ohlinger, *plus-minus: Digitalrecorder - Die Angst der Werbeindustrie (online: http://www.sr-online.de/statisch/Programm/Fernsehen/ARD/Plusminus/20030729/ thema05.html)*, 2003-07-29, last checked: 2005-06-15.

[PM94] Claudia Popien and Bernd Meyer, *A service request description language*, FORTE'94, Chapman & Hall, Bern, January 1994, pp. 14–32.

[RC03] Anand Ranganathan and Roy H. Campbell, *An infrastructure for context-awareness based on first order logic*, Personal and Ubiquitous Computing, vol. 7, Springer-Verlag London Ltd, December 2003, pp. 353–364.

[RHC$^+$02] Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganat, Roy H. Campbell, and Klara Nahrstedt, *Gaia: A middleware infrastructure to enable active spaces*, IEEE Pervasive Computing (2002), 74–83.

[SdF03] Mithun Sheshagiri, Marie desJardins, and Tim Finin, *A planner for composing services described in daml-s*, Proceedings of ICAPS'03 Workshop on Planning for Web Services, June 2003.

[SG03] João Pedro Sousa and David Garlan, *Improving user-awareness by factoring it out of applications*, UbiSys'03 - System Support for Ubiquitous Computing Workshop, October 12 2003.

[SPAS03] Katia P. Sycara, Massimo Paolucci, Anupriya Ankolekar, and Naveen Srinivasan, *Automated discovery, interaction and composition of semantic web services.*, J. Web Sem. **1** (2003), no. 1, 27–46.

[SPB05] Rachid Saadi, Jean-Marc Pieson, and Lionel Brunie, *APC: Access pass certificate. distrust certification model for large access in pervasive environment*, IPCS'05, 11-14 July 2005.

[SPP$^+$03] Umar Saif, Hubert Pham, Justin Mazzola Paluska, Jason Waterman, Chris Terman, and Steve Ward, *A case for goal-oriented programming semantics*, UbiSys'03 - System Support for Ubiquitous Computing Workshop, October 12 2003.

[SvdAB$^+$03] Steffen Staab, Wil M. P. van der Aalst, V. Richard Benjamins, Amit P. Sheth, John A. Miller, Christoph Bussler, Alexander Maedche, Dieter Fensel, and Dennis Gannon, *Web services: Been there, done that?*, IEEE Intelligent Systems **18** (2003), no. 1, 72–85.

[Unt05] TC Unterhaltungselektronik, *Tvoon media center (online: http://www.tvoon.de/)*, 2005-02-15, last checked: 2005-06-15.

[VR04] Maja Vukovic and Peter Robinson, *Adaptive, planning-based, web service composition for context awareness*, Second International Conference on Pervasive Computing, April 2004.

[VRV05] Mathieu Vallée, Fano Ramparany, and Laurent Vercouter, *Composition flexible de services d'objets communicants*, UBIMOB 05, Mai 31 - June 3 2005.

[Wei91] Mark Weiser, *The Computer for the 21st Century*, Scientific American, September 1991, pp. 94–10.

[Wei96] ———, *Ubiquitous Computing (online: http://www.ubiq.com/hypertext/weiser/UbiHome.html)*, 1996-03-17, last checked: 2005-06-15.

Glossary[16]

Bluetooth An industrial specification for wireless personal area networks, range 10m–100m

BNF Backus-Naur Form, a metasyntax used to express context-free grammars: that is, a formal way to describe formal languages

Ethernet A frame-based computer networking technology for local area networks

HMI Human Machine Interface (User Interface), the aggregate of means by which people (the users) interact with a particular machine, device, computer program or other complex tool

IEEE 802.11b/g A synonym for *WLAN*

LIRIS Laboratoire d'InfoRmatique en Images et Systèmes d'information, Lyon Research Center for Images and Intelligent Information Systems

OWL-S Web Ontology Language, a markup language for publishing and sharing data using ontologies on the Internet

PDA Personal Digital Assistant, handheld devices that were originally designed as personal organizers, but became much more versatile over the years

PerSE Pervasive Service Environment, a middleware developed at *LIRIS* managing a complex computer system consisting of embedded services, cmp. chapter 3

Pervasive computing The pervasive computing integrates computation seamlessly into the environment, rather than having computers which are distinct objects

PsaQL Pervasive Service Action Query Language, a formal language to describe service requests in a pervasive service environment, developed in this work (cmp. chapter 4)

Service An application without *HMI* performing a specific task and accessible via an application programming interface

SQL Structured Query Language, the most popular computer language used to create, modify, and retrieve data from relational database management systems

Ubiquitous computing synonym for *pervasive computing*

Web Service A Web Service is a collection of protocols and standards used for exchanging data between applications using the Internet or another appropriate network

WiFi A promotional labeling of *WLAN*

WLAN Wireless LAN, a wireless local area network that uses radio waves as its carrier: the last link with the users is wireless, the backbone network usually uses cables

HTTP(S) HyperText Transfer Protocol (Secure), the primary method used to convey information on the World Wide Web.

XML Extensible Markup Language, a text-based general-purpose markup language to structure data semantically

[16]Created with the help of Wikipedia, the free encyclopedia (http://wikipedia.org)

A PerSE-Base Description Interface

To share the information about available services in the pervasive environment with other PerseBases, each PerseBase supports a set of commands resulting in specific XML-responses (cmp. chapter 5.2). In this annex, responses for each command are exemplary shown.

The *allObjects*-request reveals all known PerseBases from the demanded database:

allObjects.xml

```
<objects>
  <object>
    <name>Notebook</name>
    <guid>{ACFC602D-9A76-4cec-8D56-DBF517EB711C}</guid>
    <doc>services.xml; serviceDescriptionByGUID.xml?serviceGUID</doc>
    <addresses>
      <ipv4>172.20.0.9:8080</ipv4>
    </addresses>
    <description>Laptop used by A. M.</description>
    <keywords>Notebook; Laptop; Portable</keywords>
  </object>

  <object>
    <name>ClassroomServer</name>
    <guid>{f1da6754-e4ee-42d7-a631-d27753f08d86}</guid>
    <doc>services.xml; serviceDescriptionByGUID.xml?serviceGUID</doc>
    <addresses>
      <ipv4>172.20.0.72:8080</ipv4>
    </addresses>
    <description>This computer is fixed in classroom 101</description>
    <keywords>Classroom; Presentation; Projection</keywords>
  </object>

  <object>
    <name>pda_z</name>
    <guid>{015f9085-8e26-4d46-abfa-f274a4b69b1f}</guid>
    <doc>services.xml; serviceDescriptionByGUID.xml?serviceGUID</doc>
    <addresses>
      <ipv4>172.20.0.112:8080</ipv4>
      <ipv6>fe80:0000:0000:0000:020d:93ff:fe72:da6a:8080</ipv6>
      <dns>pda.personal.name:8080</dns>
    </addresses>
    <description>PDA of M. Z.</description>
    <keywords>Notes-Taker; Palm; PDA; Mobile</keywords>
  </object>
</objects>
```

The *services*-request returns information about local available services:

services.xml

```
<services>
  <service>
    <name>FileSystem</name>
    <guid>{EFBF165F-E665-4388-966B-83ED31085D77}</guid>
    <description>Access to filesystem</description>
    <keywords>FileSystem</keywords>
    <parameters>
      <parameter>filename</parameter>
    </parameters>
  </service>
  <service>
    <name>HTMLViewer</name>
    <guid>{2F2EA93A-081A-495d-8FB6-0EC80D86A182}</guid>
    <description>View HTML files</description>
    <keywords>View; HTML</keywords>
    <parameters>
      <parameter>URL</parameter>
    </parameters>
  </service>
</services>
```

For each service it is possible to get detailed information using the *serviceDescription*-request in combination with the service's GUID:

serviceDescriptionByGUID.xml?{2F2EA93A-081A-495d-8FB6-0EC80D86A182}

```
<service>
  <name>HTMLViewer</name>
  <guid>{2F2EA93A-081A-495d-8FB6-0EC80D86A182}</guid>
  <description>View HTML files</description>
  <keywords>View; HTML</keywords>
  <parameters>
    <parameter>URL</parameter>
  </parameters>
  <ports>
    <port>
      <guid>{9DE5409F-A3A3-453b-A103-1A060EC910D3}</guid>
      <name>channel1</name>
      <direction>in</direction>
      <accepted_data_types>text/ascii</accepted_data_types>
    </port>
  </ports>
</service>
```

B Sparse Flooding Algorithm

In chapter 7 the question of how to assure the best local availability of service descriptions was raised. One possibility, proposed in this appendix, is to distribute service information proactive to the machines where they are needed in the pervasive network. While this attempt can be considered as theoretical reasonable, unfortunately there was not enough time and resources to develop the idea further on and to verify its capability in a benchmark environment.

Introduction

A PerSE service network (ServiceNet) consists of two main types of entities: PerSE *Services*, that means a long running application as well as any other data treating unit, a data source like a sensor, a transforming algorithm like an adopted classical Web Service as well as data drains, a video projector for example. Each service is managed (together with other – maybe related – services) by a *PerseBase* (cmp. figure 2 on page 8). These core entities are interconnected (with a wired connection or with a wireless link like Bluetooth or WiFi) and therefore responsible for these *connections*, too. By using this network, each PerseBase is able to run the services offered by any other PerseBase in the network in responding to a user demand (action). To do this efficiently, the PerseBase is using a precalculated action graph which is based on its or the neighbor's action graph history and the known entity descriptions respectively.

The information about each entity (service, PerseBase) consist of a more or less statically description, the more dynamical context (managed by the service's PerseBase) and some *meta-information* (Entity-Identificator (EID), Entity-Type) combined on an *InformationCard* (This can be considered like an extended identity card of the entity). Each InformationCard is marked with a timestamp (to decide which the newest information set in doubts is) and a validity-time (used to help the management of InformationCards and to estimate the validity of action graphs based on this information set). Each PerseBase is storing InformationCards in a database called *InformationDeck* (an exemplary network is sketched in figure 10).

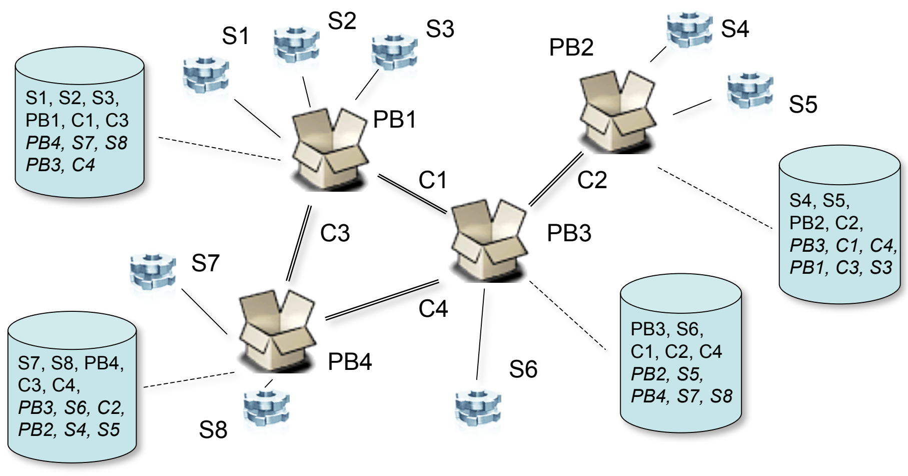
S1
S2
S3
PB2
S4
S5
PB1
S1, S2, S3, PB1, C1, C3 *PB4, S7, S8 PB3, C4*
C1
C2
PB3
C3
S4, S5, PB2, C2, *PB3, C1, C4, PB1, C3, S3*
S7
C4
PB4
S6
PB3, S6, C1, C2, C4 *PB2, S5, PB4, S7, S8*
S7, S8, PB4, C3, C4, *PB3, S6, C2, PB2, S4, S5*
S8

Figure 10: Description dispertion: Every PerseBase manages an InformationDeck storing the descriptions of the own and interesting alien entities.

Apart from this descriptional knowledge, each PerseBase has (after a sufficient amount of time) as well a kind of *"historical knowledge"* in the form of ancient used action graphs which maybe can be reused in future (if they are still valid). While the first steps in trying to react to an user action is the reuse of already existing action graphs, the creation of new, specific action graphs based on the current entities' InformationCards represents nevertheless an important part in PerSE's action treatment. This calculation is performed by the action handling PerseBase, so this base needs for the processes of service and base discovery a partial global knowledge of the ServiceNet, sufficient information about all the services, the base is "interested in". To assure that each PerseBase has the information needed for the action graph calculation, a kind of intelligent distribution algorithm is needed.

By designing an information distribution algorithm for a pervasive computing environment one has to keep in mind that the context of a service can change rapidly which means that the data stored on the service's information card changes and that this update has to be distributed to interested recipients as well. To increase the network's fault-tolerance and to avoid bottle-necks as well as single points of failure we have decided to provide in contradiction to classical attempts like Web Services (Service Registry, UDDI) or DAISGR/OGSA-DAI in computational grids a solution independent from a central server/directory. While the time for distributing information changes has to stay for the mentioned reasons in a pervasive environment as minimal as possible, we are also confronted with small bandwidth connections (like Bluetooth) or narrow-chested smart devices in some configurations demanding a minimization of the meta information transport and an optimization of the InformationDeck management.

The algorithm

To provide the possibility to calculate a new action graph in time, the PerseBase's InformationDeck does not only hold the InformationCards of the entities, the base is responsible for (that means its own InfoCard with information about connections to other PerseBases as well as the InfoCards of all connected services), but also the InformationCards from all entities it is interested in (services and the corresponding PerseBases). To obtain this knowledge, InfoCards can be exchanged among the bases sending Description Exchange Messages (DXMs) to each other. A Full DXM contains the entity's complete InformationCard as stored in the emitting PerseBase's deck, a Partial DXM, which is emitted after a record update in the InformationDeck, just contains the meta-information (EID, timestamp, previous timestamp (to assure integrity), and validity time) accompanied by the records which have changed. Combined with an algorithm of "Dull Discard" which means that each PerseBase just stores (and propagates) information about the services it is interested in we receive a sparse flooding algorithm (EID + Card Timestamp provide a unique DXM identifier needed to avoid loops). Reasons for discarding a DXM are widespread and depend on the implementation and configuration of the PerseBase, this can be reasons like "information to old", "service to far away", "service to slow", "service not matching another quality criteria" or "already enough similar/better services in the InformationDeck" for instance. This mechanism leads to a reduction of information in the network in function of distance and service's value while still maximizing the user's satisfaction. The behaviour is outlined in a pseudo-code style in algorithm 4.

A second way to initiate the emission of a DXM message is a DXM Request for all or specific entities stored in a neighbor's InformationDeck. These requests are needed for the initial creation of an InformationDeck, to obtain information about entities mentioned in an (historical) action graph, or to gain information about intermediate PerseBases and their connections. Another use case is the reaction to a received Partial DXM for an entity where no InformationCard can be found in the InformationDeck. The reaction to a Full DXM request is sketched in algorithm 4.

Stressing Neighbor Problem

The mechanism of Dull Discard in combination with Partial DXMs and DXM Request can lead to an interesting ping-pong effect, which we call the *Stressing Neighbor Problem*: A PerseBase (A) floods information about a service (S) to a neighbor PerseBase B which is assessing the service as "uninteresting" and discarding the DXM. Now S is adapting (e.g. the context of S is changing) and A is updating his InformationDeck followed by a Partial DXM emitted to its neighbors. B receives the message, is not able to determine the "level of interest" from the given values and therefore requesting a Full DXM from A. A has to response with a Full DXM just to help B to decide that it is still not interested in S and discards all information about S (cmp. figure 11).

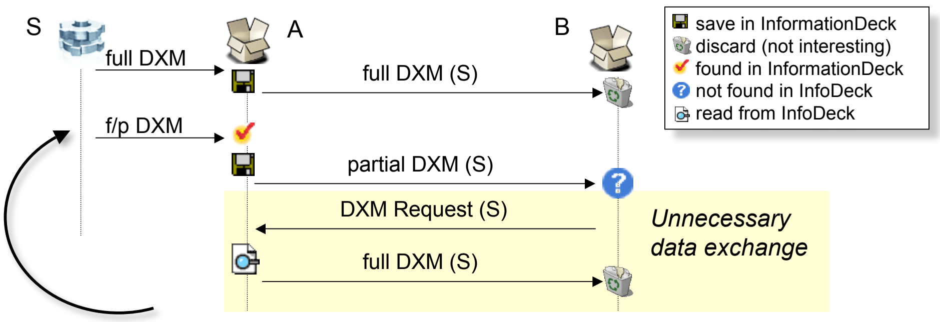

Figure 11: The Stressing Neighbor Problem: B requests a Full DXM of S from A in reaction to a received Partial DXM, just to discard this information afterwards.

In small networks with cheap connections containing no loop reactions this superfluous communication does not lead into big troubles and can easily be ignored. To handle the Stressing Neighbor Problem, a PerseBase can remember why it suppressed the information of which entity and just request a Full DXM when a record changed that justified this decision. But by this, the PerseBase stores a lot of rubbish it is not interested in and it needs a management to maintain this mountain of garbage (LFU strategies, validity time,...). Also transferring the problem to the neighbors (just send information to a neighbor when he did not discard the last DXM) is not solving it, but increasing the communication costs by introducing special deny messages and "I've changed my mind" notes.

Open questions

Apart from settle this question, there are still a lot of other challenges connected to the introduced sparse flooding algorithm which are leading to further research: Which exactly are the records on the InformationCards, what can be considered as "description", what as "context"? How to estimate the values of this context (measurements, still-alive-packets)? How to interpret these values and to compare services with each other? One of the main objects is to define rules for assessing a DXM as "interesting" and "not interesting" representing the heart of the sparse flooding algorithm. At the level of implementation, decisions have to be made on how to represent the information on

Algorithm 4 The main methods of the sparse flooding algorithm

```
action onMsgReceived(msg,emitter)  // Reception of a Full or Partial Descr. Exchange Message (DXM)
 1: if not(isEntityInDeck(msg.eid))  // No information about the entity in database
 2:     if not(isFullDXM(msg))
 3:         sendFullDXMRequest(emitter,msg.eid)
 4:         return  // End action and wait for new message
 5:     if not(isInteresting(msg))
 6:         return  // Discard message
 7:     storeEntityInDeck(msg)
 8:     for each neighbors as neighbor
 9:         if neighbor $\neq$ emitter
10:             sendDXM(neighbor,msg)  // Flood full message
11: else  // There is already a knowledge about the entity in database
12:     currentInfo = getEntityFromDeck(msg.eid)
13:     if msg.timestamp $\leqslant$ currentInfo.timestamp
14:         return  // Message to old
        // Make msg a complete InformationCard
15:     if not(isFullDXM(msg))
16:         msg = mergeInfos(currentInfo,msg)
        // Get the records which have changed
17:     infoDiff = getInfoChanges(currentInfo,msg)
18:     if isEmpty(infoDiff)
19:         return  // Nothing has changed
20:     if not(isInteresting(msg))
21:         removeEntityFromDeck(msg.eid)
22:         return  // Discard entity
23:     storeEntityInDeck(msg)
24:     for each neighbors as neighbor
25:         if neighbor $\neq$ emitter
26:             sendDXM(neighbor,infoDiff)  // Flood partial message
27: return

action onFullDXMRequestReceived(eid,questioner)  // Reception of a Full DXM request
 1: if eid $\neq$ null  // Send particular InformationCard
 2:     infoCard = getEntityFromDeck(eid)
 3:     sendDXM(questioner,infoCard)
 4: else  // Demand for all known entities
 5:     for each entitiesInDeck as infoCard
 6:         sendDXM(questioner,infoCard)
 7: return
```

the entity's InformationCard (hierarchically or "flat"? Provide generic templates to reduce transportation costs further?) as well as defining a structure to organize the InformationDeck in a way it can be easily accessed and updated, but is also prepared to support a fast action graph calculation. These considerations lead to open questions about the handling of physical limits which can likely be found when implementing PerseBases (together with their corresponding InformationDecks) on smart devices. Working in a pervasive and hereby inevitably vulnerable environment pose security questions, too: How to assure integrity of DXMs? How to assure authentication of DXMs? Is signing an appropriate way? But do we need different "level of trusts" then based on computational power of a PerseBase? At the moment, the sparse flooding algorithm seems to lead to a further improvement of the existing way of information distribution in PerSE, but the experiments to prove this superiority are not yet finished.

C The PlnGen Algorithm

The PlnGen algorithm generates efficiently Pseudo Local Network structures. Pseudo Local Networks normaly have several local and a few distant connections. Based on a number of random decisions and designed with the goal of creating complex structures with a non-complex algorithm, they neither reflect correctly natural computer nor Small-World networks. Nevertheless, these networks can be used to simulate on a very low level heterogenious, partially connected networks (like networks of pervasive entities) formed with local network-mediums like WLAN or Bluetooth. The algorithm has a run-time complexity of $O(n)$ as well as a memory complexity of $O(n)$.

The main idea of the algorithm is to distribute normally the network nodes on an area of fixed size and to benefit from this random distribution sequence. During the distribution of the nodes, connections with *backConnDistance* d predecessors are established. For each node, the *connectivity* c_i ($c_i < d$) will be fixed with a normal random generator. In the following consolidation phase, just the c_i shortest connections of each node will be preserved while the other $(b - c_i)$ connections will be eliminated. Due to the random sequence of node-distribution, long distance connections will be preserved while short distance connections nevertheless will be favored.

Algorithm 5 The PlnGen algorithm

```
input regionSize // The dimensions of the region, in which the network resides
input nodeCount // The number of nodes in the network
input minimalConnNumber // Minimal connections of each node
input standardDeviation // Standard deviation to estimate node connectivity
input backConnDistance // The backward connection distance
output nodes // Container for the PLN
 1: for i = 0; i < nodeCount; i++
      // Place the node in the region
 2:   nodes[i].coordinates = random_uniform_point(regionSize)
      // Create basic connections
 3:   for k = 0; i > 0 && k < min(backConnDistance,i); k++
 4:     j = i-k-1 // Select partner
        // Calculate distance
 5:     distance = get_distance(nodes[i].coordinates,nodes[j].coordinates)
        // Store connections
 6:     nodes[i].connections[j] = distance
 7:     nodes[j].connections[i] = distance
      // Define connectivity (randomly min+λ)
 8:   nodes[i].connectivity = minimalConnNumber+abs(random_normal(0,standardDeviation))
      // Consolidate connections
 9:   for each i in random_permute(0..nodeCount-1)
10:     partners = sort_by_distance(index(nodes[i].connections))
        // Delete the longest connections
11:     for k = partners.size-1; k > 0 && nodes[i].connections.size > nodes[i].connectivity; k--
12:       j = partners[k]
13:       if nodes[j].connections.size > nodes[j].connectivity // if partner is not already to less connected
14:         delete(nodes[i].connections[j])
15:         delete(nodes[j].connections[i])
```

A lot of algorithms to generate networks already exist, like for Small-World Networks presented by Kleinberg[17] as well as research is done, for example by Adamic[18]. Since the PlnGen algorithm does not try to create real Small-World Networks, it can meet a lower complexity of $O(n)$ then more strict approaches. A lot of references about the subject of network theory are collected by Cosma Shalizi.[19]

Some results of the PlnGen algorithm are presented in figure 12. An interactive demo can be found at `http://liris.wh4f.de/pln_gen_demo.html`.

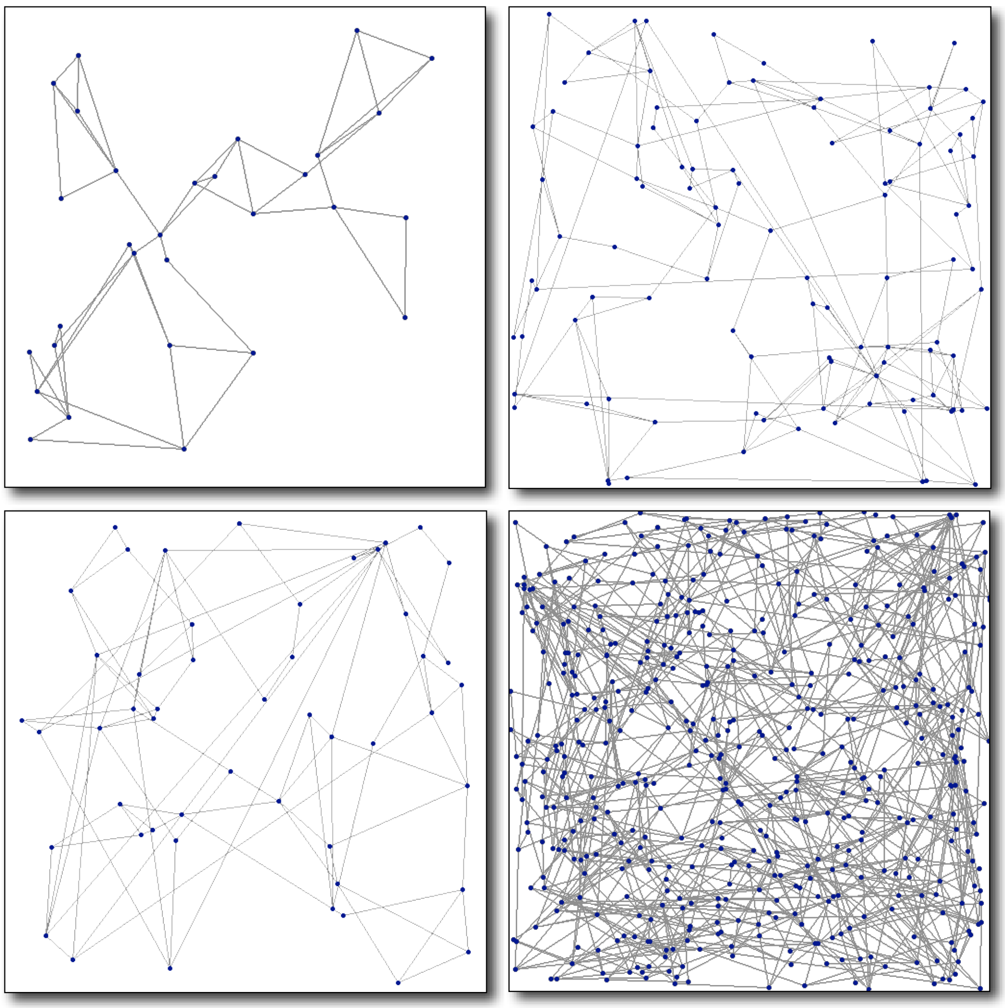

Figure 12: Pseudo Local Networks, generated with the PlnGen algorithm. The networks consists of 30, 50, 100 and 500 nodes respectively. The minimal connection number is always 2, connection number standard deviation 2.0 and backward connection distance 20. Each network area has a size of 500x500.

[17] `http://doi.acm.org/10.1145/335305.335325`

[18] `http://www.hpl.hp.com/research/idl/papers/smallworld/smallworldpaper.html`

[19] `http://cscs.umich.edu/~crshalizi/notebooks/complex-networks.html`