Expressing and Interpreting User Intention in Pervasive Service Environments

Pascal Bihler§ Vasile-Marian Scuturici, Lionel Brunie‡
§Computer Science Department, Yale University,
51 Prospect St., New Haven, CT 06520, USA
bihler@cs.yale.edu
‡Laboratoire LIRIS - UMR 5205, INSA de Lyon,
7 avenue Jean Capelle, 69621 Villeurbanne cedex
France
{marian.scuturici,lionel.brunie}@insa-lyon.fr

Journal of Digital
Information Management

ABSTRACT: *The introduction of pervasive computing environments enforce new ways of human-machine-interaction. The welldefined interaction interfaces will make place for other, more intuitive ways of interaction. In a pervasive service environment, the system middleware should take care of capturing the user's expression of an action intention, solving ambiguousness in this expression, and executing the final pervasive action This article introduces the Pervasive Service Action Query Language (PsaQL), a language to formalize the description of a user intention using composed pervasive services. It presents the next steps of intention treatment in a pervasive service environment: A mathematical model is given, which helps to express the algorithms performing translation of the user intention into an executable action. To implement such algorithms, a suitable object- oriented model representing actions is introduced. In the scope of PERSE, a pervasive service environment developed by our research group, general evaluation metrics for such algorithms are identified, a prototype has been developed and first benchmark results are presented in this article.*

Categories and Subject Descriptors
H.2.3 [Languages]**;** Query languages **H.1.2** [User/Machine Systems]: **H.5.1** [Multimedia Information System]

General Terms
Pervasive service environment, Human-machine interaction

Keywords: Pervasive Service Action Query Language (PsaQL), Pervasive Computing, Object- oriented model, Algorithms in pervasive computing

Received 10 Sep. 2005; Reviewed and accepted 27 Oct. 2005

1. Introduction

Pervasive Computing is going to become everyday reality for a large part of our society. A network of omnipresent, highly embedded computing machines 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/her[1] 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.

Computing devices will move from reactive actions to proactive ones. 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.

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". These two first aspects touch the two ways of deriving the action intention, either from an explicit user directive 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 the 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, a pervasive service environment platform called PERSE has been developed. This article presents the first step towards interpreting a user intention and describes in a formal way a corresponding action using service composition. In addition, the article shows how this approach can be included into a pervasive service environment middleware. As a proof-of-concept, the proposed algorithms have been integrated in the PERSE environment and benchmark test have been performed.

The rest of this article is organized as follows: The challenges and constraints of a pervasive service environment are presented by introducing the main aspects of PERSE in Sect. 2. In the following Sect. 3, the process of handling the user intention in a pervasive service environment is presented. We introduce PsaQL, 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. After this a formal description of the translation process between the user's and the machine's interpretation, along with corresponding algorithms are described. This is followed by some benchmarks guidelines and prototype results in Sect. 5, leading to a short overview of related work, a conclusion, and a set of open questions.

2. PERSE: A Pervasive Service Environment

In this section we introduce in an informal way the key concepts of this article by presenting the pervasive service environment PERSE developed by the Lyon research group. This allows us to introduce in an informal way the key concepts of this article that will be formally defined in the next sections.

Pervasive service environments support the interaction of independent services 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*.

PERSE models such a system and offers the interfaces needed to use the managed services without worrying about the limits of the pervasive environment. The system consists of inter- connected bases and services: Each *service* is managed by a baseapplication called *PerseBase*, corresponding to a device included in the pervasive environment. The PerseBase is responsible for managing the services it proposes and is capable to construct and start complete actions. 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. 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 this partial action and the knowledge it has about the available services in the network. This process is sketched in Fig. 1 and can be easily understood with the following example:

A person enters a room and wants to display some presentation about his vacation from his personal notebook. Today, he has to connect his computer to the local network and copy the presentation-file to the local server connected to the video projector, or he can disconnect the projector from the local machine and connect it to his own mobile computer, adjust the screen settings and so forth. A

[1] In the following, we will just use the male form, but the female form is intentionally included.

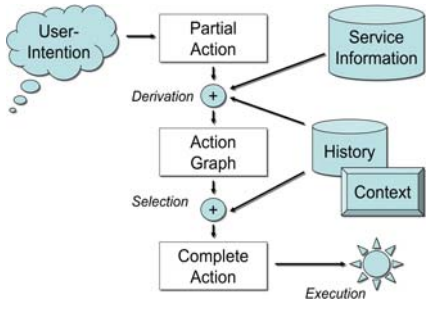

Figure 1. A user intention is transformed into a complete action by selecting the optimal action from all possible complete actions (action graph) matching the user intention, using the knowledge about available services, the context information and the execution history.

PERSE-enabled, smart classroom, does not provide the ability to disconnect 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 communicates with the local resources via Bluetooth or WLAN interprets the user command as a partial action. Using the information about the local available services, context information and the action history, the PerseBase constructs a graph of all possible (and reasonable) service combinations, each representing a complete action matching the given partial action. From these combinations, an algorithm (see Sect. 4) 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.

3. 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 that the system should execute seems very clear for him. Unfortunately, this action is not complete that is it lacks some information to be executable.

It is the challenge for the embedded user-intention-interpretation algorithm to create, based on this little information from the user, a complete action that suits best the user intention, maximizing his level of satisfaction with the system. The first step is to introduce a formal language, representing in a welldefined way the action as given by the user. 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). PsaQL plays a similar role in the PERSEenabled pervasive environment as SQL [Date 1986] does for accessing relational database management systems. PsaQL also looks similar to SQL on the first glance (BNF notation):

Language Definition 1: PsaQL

```
<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', '^', '$',
                              '(', ')', ' [', ']', '.', '+', '*', '?', ...>}
```

Our example (see Sect. 2), expressed in PsaQL:

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

In this case 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 use the "LIKE"-statement (currently treated as regular expression).

Based on this description of a partial action, an action graph of all possible solutions is created and finally, a complete action is selected (see Fig. 1), which can be executed by the pervasive environment. 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.

4. From User Intention to User Satisfaction

4.1. Translating a Partial Action into a Complete Action

We present in this section an algorithm to translate a partial action into a complete action (see Fig. 1).

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 for instance 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

With some mathematical formalization, we can define step two and three of this algorithm:[2]

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 =⊥.*[3] *A* partial action *p, the formal expression of a user intention, can be modeled as a list of 2-tuples:*

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

Definition 2 (Service Graph) *Let E = {ϵ =(b,s) | b⊆B, s⊆S(b)} be the set of all available services in a pervasive service environment and I(E) ⊆ E X E the set of all possible service interactions within this environment.*[4] *Then we can define a service graph in a part ℰ of the pervasive service environment as*

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

The set $\tilde{A}_E$ *of all valid service graphs in E is defined as:*

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

Definition 3 (Connected Service Graph) *A service graph g=(ϵ ℐ);ℐ⊆ℐ (ϵ) is called* connected *iff*

$$\forall\, \epsilon_o, \epsilon_n \in \mathcal{E}, \epsilon_o \neq \epsilon_n : (\epsilon_o, \epsilon_n) \in \mathcal{I} \vee$$
$$(\exists \epsilon_1, \ldots, \epsilon_n - 1 : (\epsilon i, \epsilon i + 1) \in \mathcal{I} ; (i = 0,1,\ldots, n-1))$$

Definition 4 (Solution) *A graph* $g_p E(\epsilon, \mathcal{I}) \subseteq \mathcal{I} \subseteq \mathcal{I}(\epsilon)$ *is called* solution *for a given partial action p in a pervasive service environment* E *iff*

$$g_p^E \in \Gamma_E$$
$$g_p^E \text{ is connected}$$
$$\forall e = (b, s) \in p\ \exists \epsilon = (\beta, \sigma) \in \mathcal{E} \text{ with}$$
$$\begin{cases} \beta = b & \text{if } s = \perp, \\ \sigma = s & \text{if } b = \perp, \\ (\beta = b) \wedge (\sigma = s) & \text{otherwise} \end{cases}$$

[2] To simplify the model, we do not consider attributes in the following section, they can be easily added lately.

[3] ⊥ represents "undefined".

[4] The service-interoperability is not studied here. We assume that (ϵ1,ϵ2)∈ I (E) = ((ϵ1,ϵ 2)∈ EX E) ∧(ϵ1 *is interoperable with* ϵ 2)

Definition 5 (Action Graph) *Let $\mathcal{S}^E_p$ be the set of all solutions for p in a pervasive service environment E. An* action graph $A^E_p \subseteq \tilde{A}$ *of a partial action p in the pervasive service environment E is a connected service graph containing all solutions for p:*

$$A^E_{p=}\Big(\bigcup_{(\mathcal{E},\mathcal{I})\in \mathcal{S}^E_p} \mathcal{E} \quad \bigcup_{(\mathcal{E},\mathcal{I})\in \mathcal{S}^E_p} \mathcal{I}\Big)$$

Definition 6 (Complete Action) *Let C(g,) be the function calculating the cost of a solution g when it is executed in a context .*

Then the complete action c_p *for a given partial action p in a service environment E is defined as:*

$$C(c^E_p,\gamma) = \min\{C(g^E_p,\gamma) \mid g^E_p \in \mathcal{S}^E_p\}$$

Let *H(p,E,γ)* 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 can define an exhaustive algorithm (Algorithm 1), transforming a user intention *p* into a complete action c^E_p.[6] Starting with an action graph (representing a solution for *p*), this algorithm removes step by step the connections (see line 11 of algorithm 1). 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 global cost is selected as solution of the algorithm.

Algorithm 1 Exhaustive Translation Algorithm

```
1:  c = H(p,E,γ) // try to find a solution in history
2:  if c ≠ ⊥
3:    c_p^E = c
4:  else
5:    fifo A, fifo S
6:    push(A, A_p^E)
7:    while A ≠ ⊥
8:      (ℰ,ℐ) = g = shift(A) // take a service graph from the beginning of the list
9:      is_leaf = true
10:     for each ι ∈ I
11:       ℐ' = ℐ - {ι} // remove connection ι from ℐ
12:       ℰ' = ∪_{(ι1,ι2)∈ ℐ'} [(ε1,ε2)]} // keep connected elements
13:       if g' = (ℰ',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' ∈ S // find minimal solution
20:       if C(g',γ) < C(g,γ)
21:         g = g'
22:   c_p^E = g
```

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. It is not applicable in larger pervasive service environments, so we developed an heuristic approach, solving the query in polynomial time (Algorithm 2) using the same definitions as above. The approach bases on the following coloring of the nodes, defining a partition of *E*:[7]

- 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 $\in W$: : all other nodes

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 (Alg. 2):

First we calculate using the Shortest Path Algorithm of Dijkstra [Manber 1989, p. 204] for each red node *R* the distance to all black nodes *B* by using a modified cost function $C_l((R, B), \geq)$. 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 [Manber 1989, p. 208] we extract from *(E,I(E))* the graph containing all black nodes as a complete action (see Fig. 2).

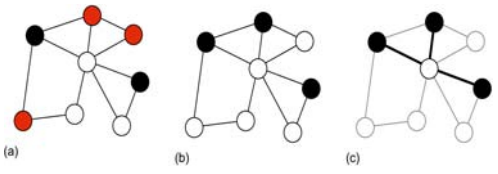

Figure 2: In this example, the partial action *p* contains three elements. Two of them define exactly a corresponding (black) node in the action graph and one matches to three different (red) nodes (a). From the red nodes, the one with the nearest distance to a black node is selected and colored black, the others are colored white (b), and the Minimum Cost Spanning Tree containing all black nodes is calculated (c).

Algorithm 2 Heuristic Translation Algorithm

```
1:  c = H(p,E,γ) // try to find a solution in history
2:  if c ≠ ⊥
3:    c_p^E = c
4:  else
5:    list F // black nodes
6:    list R // red nodes
7:    for each ε = (β,σ) ∈ E
8:      if ∃ e = (b,s) ∈ p: (b = β) ∧ (s = σ)
9:        push(F, ε)
10:     else if ∃ e = (b,s) ∈ p: ((b = β) ∧ s = ⊥) ∨ ((s = σ) ∧ b = ⊥)
11:       push(R, ε)
12:   array D // calculate the minimal distance
13:   for each ε_f ∈ F
14:     for each ε_r ∈ R
15:       d = length(dijkstra((E,I(E)), ε_f, ε_r))
16:       if d < D[ε_r]
17:         D[ε_r] = d
      // elements of partial action covered only by red nodes:
18:   p' = {e = (b,s) ∈ p | (b = ⊥) ∨ (s = ⊥) ∧ (¬∃ ε_f = (β,σ) ∈ F : (b' = β) ∨ (s' = σ))}
19:   sort(R,D) // sort red nodes by distance
20:   for each ε_r = (β,σ) ∈ R
21:     if ∃e = (b,s) ∈ p': (b = β) ∨ (s = σ)
22:       push(F, ε_r) // color the node black
          // removed satisfied parts from the partial action:
23:       p' = p' - {e' = (b',s') ∈ p' | (b' = β) ∨ (s' = σ)}
24: c_E = MCST((E,I(E)),F) // calculate Min. Cost Spanning Tree on black nodes
```

4.2. Complete Action Representation

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.

To describe a complete action, we use the object model sketched in Fig. 3. The utilised classess are:

- *Address* - Combination of a type (e.g. "ipv4") and a valid address-string (e. g. "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 *ervice*. 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*)

5. In some rare cases (when $\forall (b,s \in p: b = 1)$ it can happen, that *Ap E* is not unique. If so, algorithm 1 (see below) has to be executed for each $(A^E_p\ E)$i.

6. A^E_p can be directly derived from *(E,I(E))*.

7 Each node wears just one color, a hierarchy is given as: *Black* > *Red* > *White*, at least one black node is required.

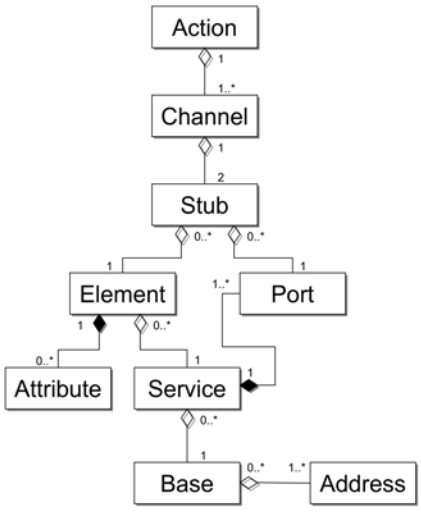

Figure 3. Class diagram for the internal representation of action graphs and complete actions.

- *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 *Channel*

5. Implementation and Evaluation Issues

5.1. Service Description Management

The objects which form an action graph are created based on a database connected to the system performing the translation process.[8] 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. We have designed a PerseBase-API supporting the following requests:

- *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.

The results of this interface requests are transmitted using specific XML schemas. The interoperability of services is determined using the `accepted_data_types` information of a service description.

5.2. Evaluation Metrics

To achieve the goal of seamless integration of the pervasive computing system into the user's everyday life environment, 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 that 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 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 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 service descriptions* 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 section 5.3).
- *Pervasive Computing Constraints* as an *accountable usage of CPU and memory resources*.

5.3. 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 4.1. 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. The connecting network has been created using the PLNGen algorithm, developed for this testcase and modeling pervasive networks.[9] 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.

The results of our first test runs (see Fig. 4) 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.

6. Related Work

The idea to execute pervasive applications in a "user-aware" way has been worked out for instance in [Sousa 2003]. Similar as PERSE does, El-Kathib et al. design in [El-Khatib 2004] 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, where he doesnot present a formal language to define the adaptation requests. Other user-oriented systems for managing pervasive environments have been developed, for instance Gaia [Román 2002] or Aura [Garlan 2002]. Gaia proposes a programming language to construct executable tasks based on the interoperability of services, whereasAura tries to avoid any interaction with the user and does not present a model for user intention.

[8] Normally, this will be a PerseBase.

[9] see http://liris.wh4f.de/pln_gen_algo.html

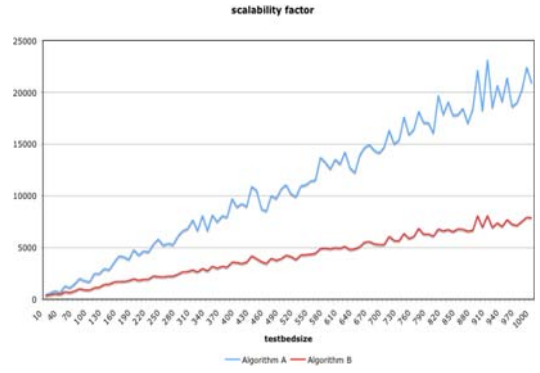

Figure 4. The progression of the scalability factor as introduced in the text when varying the testbed size.

The Ninja Environment [Gribble 2001] introduced the idea of composing services on distributed adaptation paths, whereas [Buchholz 2003] added path optimization considerations for content adaptation. In [Vallee 2005], M. Vallée et al. introduce a system for dynamic service composition in intelligent environments: they are following a related way as PERSE does, 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.

Another widely explored field of research inspiring the development of this work are Semantic Web Services, as presented for instance by [McIlraith 2001]. Ontologies as defined by OWL-S [McGuinness 2004] 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 [Staab 2003] among others.

A pervasive environment will be characterized by the availability of application context information [Ma 2005]. PERSE will use the contextual information to build an appropriate action graph and to select the best solution as complete action. Earlier approaches introducing context based adaptation into pervasive environments are presented in [Ranganathan 2003] and [Mostefaoui 2003].

7. Conclusion and Open Issues

This article has presented a strategy and a methodology to take the user intention into account when composing service-based actionsin a pervasive service environment. It introduced PsaQL, a language to express user intention in a pervasive service environment. We outlined an exhaustive algorithm extending this partial action to an executable graph of services (complete action) using the service descriptions, the context information and the execution history.

We presented an object-oriented model of complete actions. Finally, some metrics supposed to model the user satisfaction and environment scalability has been worked out and performance tests have been applied to a prototype implementation.

In future, 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 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. As well, security will be important in further PERSE-development. The current presumption that all services can be used by everyone cannot be hold in real world application, a system ofauthentication and authorization based on roles and access control lists will be implemented.

References

1. Buchholz, Sven., Buchholz, T(2003). Adaptive content networking. ISICT '03, 2. : Proceedings of the 1st international symposium on Information and communication technologies, p. 213–219. Trinity College Dublin.

2. Date, C. J (1986). A guide to the SQL standard. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

3. El-Khatib, Khalil, G. v. Bochmann . A. E. Saddik (2004). A Distributed Content Adaptation framework for Content Distribution Networks: http://beethoven.site.uottawa.ca/dsrg/PublicDocuments/Publications/ElKh04c.pdf last visited: 2005-06-15.

4. Garlan, David, Siewiorek, D., Smailagic., A, Steenkiste, P (2002). Project Aura: Toward Distraction-Free Pervasive Computing. IEEE Pervasive Computing, 1(2) 22–31.

5. Gribble, Steven D., M. Welsh, R. von Behren, E. A. Brewer, D. Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R. H. Katz, Z. M. Mao, S. Ross, B. Zhao . R. C. Holte (2001). The Ninja architecture for robust Internetscale systems and services. . IEEE Intelligent Systems, 35(4):473–497. Elsevier North-Holland, Inc., New York, NY, USA.

6. Ma, Jianhua, L. T. Yang, B. O. Apduhan, R. Hunag, L. Barolli . M. Takizawa (2005). 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, 53–68. Troubador Publishing Ltd.

7. Manber, Udi (1989). Introduction to Algorithms: A Creative Approach. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

8. McGuinness, Deborah L., van Harmelen, F (2004). OWL Web Ontology Language Overview. http://www.w3.org/TR/2004 /REC-owl-features-20040210/ last visited: 2005-06-15

9. McIlraith, Sheila A., Son, T. C., H. Zeng, H (2001). Semantic Web Services. *IEEE Intelligent Systems,* 16 (2) 56–53. IEEE Educational Activities Department.

10. Mostéfaoui, Soraya Kouadri (2003). A Context-Based Services and Discovery and Composition Framework for Wireless Environments. *In*: Proceedins of the 3rd IASTED International Conference on Wireless and Optical Communications (WOC'2003), p. 637–642, Banff, Canada.

11. Ranganathan, Anand., Campbell, R.H (2003). *An infrastructure for context-awareness based on first order logic*. Personal and Ubiquitous Computing, 7(6) 353–364. Springer-Verlag London Ltd.

12. Román, Manuel., Hess, C. Cerqueira, R., Ranganat, A., Campbell, R.H., Nahrstedt, K (2002). *Gaia: A Middleware Infrastructure to Enable Active Spaces*. *IEEE Pervasive Computing*, 74–83.

13. Sousa, João Pedro., Garlan, D (2003). *Improving User-Awareness by Factoring it Out of Applications*. UbiSys'03 - System Support for Ubiquitous Computing Workshop.

14. Staab, Steffen., van der Aalst, W. M. P., Benjamins, V.R., Sheth, A.P., Miller, J.A., Bussler, C., Maedche, A., Fensel. D., Gannon, D (2003). *Web Services: Been There, Done That? IEEE Intelligent Systems,* 18(1):72–85.

15. Vallée, Mathieu., Ramparany, F., Vercouter, L (2005). *Composition Flexible de Services d'Objets Communicants*. UBIMOB 05, Grenoble, France.

Pascal Bihler's research in Computer Science is currently focused on e-learning, networking, and challenges in ubiquitous computing, he has been an author and coauthor of various research papers at national and international conferences. For his undergraduate studies, he attended the Friedericiana University of Karlsruhe, Germany. He continued his studies at the National Institute of Applied Sciences (INSA)of Lyon, France, where he received a Master of Science as well as his engineering degree. At the moment, he is visiting the Department of Computer Science at Yale University in New Haven, Connecticut, where he works as an assistant researcher with the group of Prof. Yang Richard Yang. Beginning this summer, he will continue his research career at the Rhenish Friedrich-Wilhelms-University of Bonn, Germany, working towards a PhD-degree under the supervision of Prof. Armin B. Cremers.