Towards User-Oriented Application Composition

Ivan Milara Sanchez, Oleg Davidyuk, Jukka J. R. Riekki

Towards User-Oriented Application Composition

Ivan Sánchez, Oleg Davidyuk and Jukka Riekki
Dept. of Electrical and Information Engineering and Infotech Oulu,
P.O. Box 4500, University of Oulu, 90014,
Oulu, Finland
firstname.secondname@ee.oulu.fi

***Abstract*—This paper presents an autonomic system for composing ubiquitous applications at run-time. The applications are composed according to the user preferences collected via a physical user interface. This interface allows users to specify preferences by simple actions of touching with their mobile terminals icons in the environment, instead of explicitly selecting resources and dealing with their properties. In this paper, we present a system prototype and an example multimedia application. We also evaluate the performance of the prototype and the allocation algorithm which is used to compose applications.**

***Keywords*-Ubiquitous application composition, service composition, physical user interface, allocation algorithm;**

I. Introduction

Application composition is a concept that emphasizes building applications from a number of components which are physically allocated on multiple computational devices available in a ubiquitous smart space. This concept increases the flexibility of application adaptation and also enables a number of useful scenarios.

In addition to ubiquitous applications, the potential scenarios of application composition include virtual devices, multimodal interfaces, and load distribution [4]. The virtual device approach (also known as resource sharing) assumes that aggregating functionality across many resource-limited devices increases the capacity of each single device used in the aggregation. E.g., a portable video camera connected to a mobile phone enables the mobile phone to access data stored in the camera's memory and to send video messages. Application composition can also be used to achieve load distribution among personal devices by decomposing applications and allocating their heavy-weight components onto the nodes with the required computational capacities. This functionality is required, among others, in such domains as content-based retrieval, information fusion, and semantic search. In addition, application composition enables the construction of multimodal user interfaces by combining and controlling inputs and outputs from various ubiquitous devices.

Recently, application composition has become an important domain also for autonomic systems. Such systems require the users to specify a desired application only, which is then realized as needed in an autonomous manner. However, this functionality requires that the autonomic system is capable of discovering resources, comparing resource properties and, selection of an optimal application allocation (i.e. a combination of resources and application components). Besides, such an autonomic system has to be able to adapt composed applications when the context or the user goals change. Depending on the available resources, even the same application can be appropriately realized using different resource combinations. Autonomic systems are essential in smart spaces offering a great amount of services and devices, because users may become too overloaded with manual controlling and configuring applications, and thus, they may even stop using them.

Several systems for automatic application composition have been proposed. For example, Sousa et al. [6] presented a system for self-adaptation of composite applications on several architectural levels. Their approach is to specify the applications as abstract tasks, which are subject to the QoS requirements defined by the end-users. Sousa's framework calculates the optimal resource allocation and reallocation for each task using a Knapsack problem solver. Sousa's research also focuses on expressing application requirements using a set of user interfaces for collecting requirements which are then forwarded to the problem solver. In their most recent work [7], these authors presented an architectural style called uDesign, which focuses on applying end-user programming techniques to the assembly of personalized applications used for monitoring and controlling smart-spaces.

Another example of an application composition system is the COCOA middleware [12]. This middleware supports the composition of task-based applications which are modeled using workflows and QoS properties. The COCOA middleware is based on a service discovery mechanism that can handle semantic heterogeneity; in addition, it utilizes an ontology-based matching algorithm with QoS support for application composition.

Similar solution, the Galaxy [5] framework, focuses on the interoperability of devices in ubiquitous environments. Galaxy supports composite services which control and cooperate with ubiquitous devices. Galaxy, as well as COCOA,

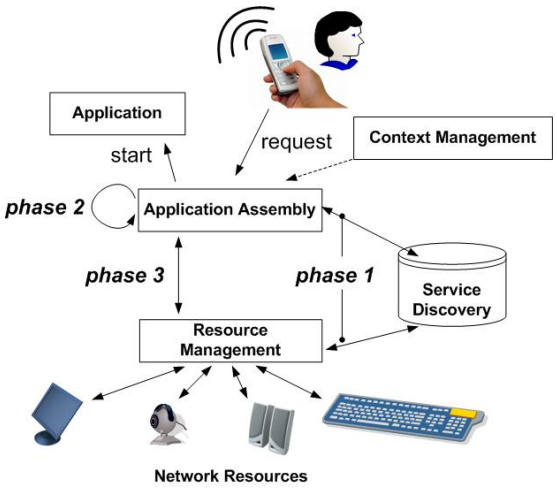

Figure 1. The Application Composition Concept.

targets semantic-free service composition issues. The Pervasive Component System (PCOM) [9] offers an infrastructure for multiple application composition algorithms as certain application scenarios may require changing the algorithm at runtime.

All aforementioned solutions target allocation of only one application component at a time, while our approach deals with multi-component allocation and optimization of the structure of the whole application according to context and user needs.

In this paper, we present a prototype of an application composition system, an example application, and results from the performance tests. In our prototype, application composition is carried out in two phases. In the first phase, a user specifies her/his preferences by simply touching icons attached to physical resources in the environment. The user touches the icons using a mobile phone, hence the phone is used as a physical tool to interact with the environment (i.e. the phone is the part of a physical user interface). In the second phase, the allocation algorithm [11] composes an application fulfilling the requirements set for the application (including the requirement to use the resources selected by the user). The REACHeS framework [3] is used to integrate the physical user interface, the allocation algorithm, and the rest of the required components into the complete system. We also conducted a user evaluation to study the feasibility of our system. The results of the user evaluation are presented in [2].

II. The Application Composition Concept

The application composition concept assumes that applications are built from a set of components which may reside on physically distributed nodes. Figure 1 shows the key components of our system. The Context Management component handles the user context and triggers the application adaptation when needed. The Application Assembly controls the application's lifecycle and also performs the application adaptation by composition. The Resource Management monitors the utilization of resources and performs application deployment. The Service Discovery handles information about nodes available in the environment and provides the matchmaking functionality. The applications are started and then the composition process takes place in three phases as follows:

The first phase. This phase of the application composition process is started either explicitly, when the user activates the application, or implicitly, after the Context Management component triggers composition. In the former case the composition is triggered by the user through a physical user interface. That is, the user requests the composition by touching with her/his mobile terminal a corresponding icon in the environment. Then, the Application Assembly searches for the available local nodes (that is, nodes in the user's close proximity, e.g. in the same room), or remote nodes (that is, nodes located physically far away from the user) through the Service Discovery. Conventional protocols, such as Bluetooth or UPnP, can be used. Application Assembly searches the nodes according to their functional and non-functional (i.e. resource) properties. The functional properties of a node denote its ability to provide certain services. Functional properties are descriptions and resemble interface statements in Java. They can also rely on ontologies to enable semantic search, as suggested by Mokhtar et al. [12]. Functional properties can be also used to indicate the availability of specific resource (e.g. a file or a user profile) at a certain node. The non-functional properties, in their turn, mainly denote device resource constraints, such as the maximum available memory or the computational resource capacity. Furthermore, these properties may be used to implement user access policies.

The second phase. We assume that a node can host one or more application components as long as the node's resource capacities are not exceeded. Besides, the functional properties of nodes and components have to be met. These two conditions are ensured during the second phase of the application composition process which is performed by the Application Assembly. This phase is the most important step in the application's startup and the execution stages. The goal of the phase is to produce a valid application composition. If no valid composition is found, the service discovery (i.e. the first phase) and the application allocation (i.e. the second phase) are iteratively repeated. It is possible that some applications might not be allocated due to their high resource demands or specific functional constraints which cannot be met by the nodes in the environment. In such cases, the Application Assembly may use another application profile with lower resource demands or downsize the resource demands through interaction with the user.

The third phase. After the valid application composi-

tion is found, it is then realized by leasing the necessary computational resources, deploying the components onto them and configuring the application. This can be achieved using a resource management schema, as suggested in [10]. However, the resource management is out of scope of this paper.

A. The Application Allocation Algorithm

Our approach focuses on optimizing the application compositions according to user preferences. An optimal application composition can minimize bandwidth consumption, balance load among the nodes, and meet the various resource requirements imposed by the application components. As the optimization goals and resource availability may vary during the application execution, they sometimes affect application compositions built earlier, making them invalid or no longer optimal. In these cases, a reallocation of the application is performed.

Finding an optimal application composition which satisfies the numerous functional and resource requirements is a complex problem and solving it requires the utilization of an allocation algorithm. Such an algorithm can be used to find an appropriate application composition which satisfies the node constraints after the application components are assigned to the chosen nodes. In other words, the algorithm solves the application allocation problem.

Our application allocation algorithm [11] is part of the Application Assembly component in Figure 1. This algorithm operates with two models, an application and a platform model. The application model defines the structure and the properties (e.g. application resource requirements) of the application. The platform model, in its turn, formally defines the network resources and their properties (e.g. node resource constraints). Both models are represented in the form of graphs, where nodes represent application components or network nodes and links model communication links (between application components or network nodes). Each element of the models can have multiple properties which specify non-functional requirements like computational resources, up or downlink channel capacities, and memory demand. Each property can be expressed by a float, an integer, or a Boolean value. The values of application properties may be set by measuring the performance of the application under different workloads and recording this data onto application resource profiles. The values of platform model properties may be measured using various resource and network monitoring tools. The task of capturing the resource requirement values is essential for the correct functioning of the allocation algorithm, but is out of the scope of our paper.

Our application allocation algorithm supports special kind of user preferences, which we call affinity constrains. These constraints restrict the deployment of an application component by specifying a set of permitted nodes for that component. These constraints have multiple purposes. For example, they can be used to represent the user trust preferences, similar to the Component Trust Binding prototype [4]. Thus, the user can specify a set of trusted nodes for each application component, and then, the algorithm will allocate the components onto the trusted nodes only. These affinity constraints enable the so-called 'visibility', a notion introduced by Malek et al. [13], which means that each network node has a 'domain' or a set of all the nodes it is aware of. The visibility is an important feature for ubiquitous environments where the handling of handoffs is necessary in the application layer (e.g. in wireless networks). Besides, the affinity constraints enable reliable application composition like in the prototype developed by Del Prete et al. [8]. To achieve this, our platform model has to have additional properties specifying reliability of each network node. The allocation algorithm will then maximize the reliability of the whole composed application. The affinity constraints are also helpful if an application component requires access to some specific resource (e.g. a file) that is only available at a certain node. In our prototype, the user creates the affinity constraints by touching with the mobile phone the physical resources that he would like to use with the application.

In order to solve the application allocation problem at runtime, the Application Assembly uses a genetic algorithm that relies on mutation and crossover operators and iteratively evolves multiple solutions. The algorithm optimizes a generic objective function that supports multiple objectives without redesigning the algorithm's code. The algorithm also supports generic application and platform models which can be easily modified to add and remove additional properties, thus the number of properties used in our models is not fixed.

B. The REACHES Framework

We realized the application composition concept using the REACHES framework [3] which enables the utilization of a mobile terminal's UI to control a wide range of ubiquitous applications. These applications can be composed dynamically from service components which process asynchronous events generated by users. The REACHES clients are resource-restricted mobile terminals, thus, all computationally-heavy tasks (including the generation of UIs and the application allocation) are performed on the REACHES server. The REACHES architecture (presented in Figure 2) is centralized and consists of four components: Remote Control, User Interface Gateway, System Display Control, and Service Components.

The Remote Control consists of a physical UI (a set of RFID tags) placed in the user's local environment and of a mobile terminal equipped with an RFID reader. The RFID tags contain commands which are triggered when the tags are read using the terminal. We use these tags in our prototype in order to trigger an application to start, utilize

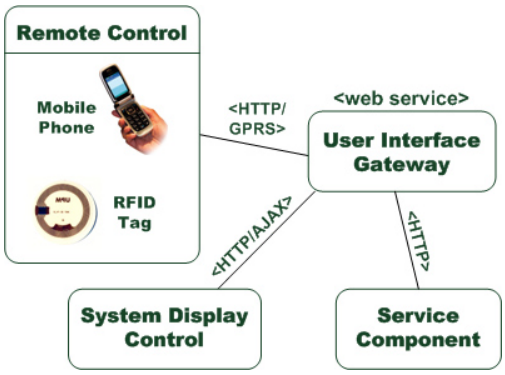

Figure 2. The REACHES Framework's Architecture.

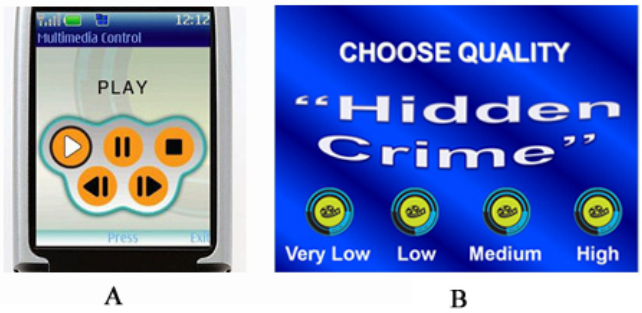

Figure 3. The remote controller GUI (A) and the control panel for choosing the quality (B).

a certain resource with the application (e.g. a certain wall display), start the adaptation of the multimedia content, and set the user preferences.

The User Interface Gateway (UIG) connects users and service components. Its main purpose is to synchronize different subsystems by processing events from the Remote Control and dispatching them to appropriate service components. In addition, the User Interface Gateway performs the service discovery functions: it registers and unregisters services, establishes connections between them, and provides the matchmaking functionality.

The System Display Control (SDC) is intended for multimedia applications. It connects external resources (e.g. displays) to the REACHES server. After the UIG registers a resource, the browser, which is hosted on a computer connected to the resource, loads scripts (REACHES client) that enable communication between the SDC and the browser. The REACHES architecture does not require deploying any other software on the resource side.

When an application is started, the system assigns one or more services and resources to it. Then, the service components send events to the resource via the SDC, which dispatches each event to the corresponding browser to perform the requested update to the user interface.

The Service Components perform application specific functions and are allocated onto remote computation nodes. The Service Components communicate with the other elements of the REACHES architecture via the UIG.

The REACHES framework implements most of the functionality shown in Figure 1. The user provides his/her preferences and triggers the application composition by touching icons in the environment (corresponds to the Context Management). The Application Assembly is performed by the allocation algorithms executed by the UIG. The UIG is aware of all resources and services available for certain context (corresponds to the Service Discovery). The SDC deploys application onto the resources (corresponds to the Resource Management).

C. The Ubiquitous Multimedia Player Application

We integrated a multimedia player application into the REACHES to demonstrate features of the framework. The application is based on Flash and supports various content including audio, video, images and flash multimedia files. The player allows rendering of multimedia files, supports streaming, and accepts dynamic playlists.

The application controller UI is shown in Figure 3 (A). The users controls the player using the mobile phone's UI as follows. The Service Component (SC) manages the multimedia content. and receives commands (generated by the user) from the Remote Control. These commands are interpreted and dispatched to the external display which embeds the multimedia player and executes the commands with the action required. The URL specifying details of the playlist file can be stored as a parameter in a RFID tag. This data is delivered to the SC when the application is started. The RFID-based control panel for the application is shown in Figure 3 (B).

III. Experimental Evaluation

We used the REACHES application framework to create a prototype of a system for application composition. Firstly, we embedded the application allocation algorithm in the framework and, tested the performance of our allocation algorithm on synthesized datasets. Our aim was to use the algorithm for real-time application composition; therefore, the computational load has to be low even when handling large application and platform models. We also tested whether the algorithm was executed correctly. The introduction of the affinity constraints made this point even more interesting. Secondly, we measured the multimedia application startup latency caused by the REACHES framework. Here, our goal was to analyze factors contributing to latency while increasing the number of the real ubiquitous resources in an environment.

The allocation algorithm was implemented in C++ and tested using the Boston University Representative Internet Topology Generator (BRITE) tool [1]. The application and

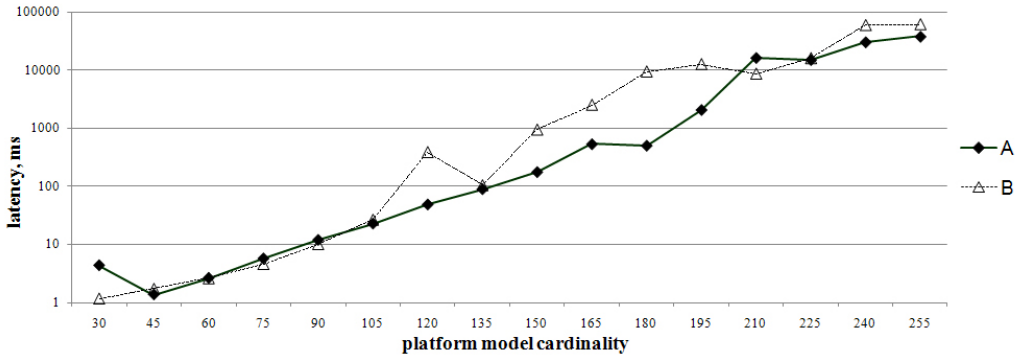

Figure 4. The allocation algorithm performance test. The curve A denotes the case when affinity constraints were present and the curve B corresponds to the original algorithm (without affinity constraints).

the platform models contains six different QoS properties, the values of which are randomly generated. We also included affinity constraints in these models as follows: each application node α from the application model A was associated with a set of nodes S from the platform model P. Nodes from the set $S_i \subset P$ were permitted to host the component α_i. However, the affinity constraint of each platform node $\beta \notin S_i$ did not permit β to host the application α_i even if β satisfied all the other constraints. Elements of these sets $S_i \subset P$ were chosen randomly, and cardinality of S_i was set to 90 per cent of cardinality of S.

We performed the experiments under Red Hat Linux 4.1.2, on an AMD Opteron dual core 270 [2GHz] PC with 8 GB memory, but only one processor core was used to execute the algorithms. Figure 4 (plotted on a logarithmic scale) shows how the performance of the allocation algorithm with the affinity constraints present (curve A) and the original algorithm (curve B) is affected by the cardinality of the models.

The figure shows that the cardinality of the platform model was gradually increased from 30 to 255 nodes, by 5 nodes in each subsequent iteration in of the experiment. The platform model's cardinality was always computed as $|P| = |A| \cdot 3$, where A and P correspond to application and platform models.

The reported measurements were recorded over 100 runs. The resulting curves (see Figure 4) show that the latency of the algorithm depends on the application (and the network) complexity. However, the presence of affinity constraints had an insignificant impact on the performance of the algorithm, which can be explained by the fact that affinity constraints actually reduce the search space, i.e. the algorithm goes through fewer solutions.

Although the latency is high for the large application sizes (e.g. 200 components), this is not a severe problem, as it is reasonable to assume that the real-world application sizes will not exceed 30 components, based on the current application estimates. As the test revealed, the algorithm spent only 12 milliseconds to process this scenario. Thus, we found the performance of our algorithm suitable for real-time tasks: we believe that the latency of the algorithm will be dwarfed by the latencies introduced, for example, by the service discovery process.

Next, we evaluated the performance of the whole system. The goal of the experiment was to study the performance of the REACHES server and also to identify possible bottlenecks. We implemented the REACHES using Java servlets which were ran in Tomcat 5.5. The mobile terminals used were Nokia 6131 NFC phones, equipped with RFID (NFC compliant) readers. The application in the mobile terminal controlled the remote network resources using a J2ME MIDlet client.

The network resources consisted of PCs connected to the REACHES server. Each PC ran the REACHES client, which executes server requests locally. The clients communicated with the server using the HTTP protocol. The allocation algorithm was integrated into the REACHES server to start and adapt the applications. We measured the latencies of the REACHES server as follows:

- *Service Discovery latency* is the time delay spent for acquiring available resource descriptions from the local database.
- *Algorithm Execution latency* is recorded from the moment when the server starts the algorithm execution until the algorithm returns an optimal solution. This latency also includes the time overhead used to create application and platform model files and the time delay spent to associate the found solutions with the resources in the database.
- *Other processes latency* includes the request processing time delays caused by the server and the communication latencies between resources and the server.
- *Overall latency* includes all the aforementioned overheads and is measured as the time interval from the moment the server received a request from the user to start an application until the resources are acquired

and the application is started. However, we omitted the communication latency between the application client in the mobile terminal and the server because its values fluctuated a lot. This was due to high variability in the quality of the GPRS connection during the test.

The latencies were measured during the presence of an increasing number of ubiquitous network resources. Figure 5 summarizes the experimental results. Each latency value was recorded in over 30 measurements.

As the graphs show (Figure 5), the *Algorithm Execution latency* occupies approximately 85 per cent of the *Overall latency*, thus dominating the other processes. The *Service Discovery latency*, which has the second largest values, occupies from 5 to 19 per cent of the *Overall latency*.

It should be noted, that in this test, the *Algorithm Execution latency* also included the time delay spent for parsing the data received from the server and transforming this data into the format understood by the algorithm. On the other hand, we left this delay out when we tested the algorithm as a stand-alone tool earlie (see Figure 4). Moreover, in the second test, an additional delay was caused by the fact that the REACHES, which was implemented in Java, used a separate C++ process to access the running algorithm.

Also, we noticed that the *Service Discovery latency* was slightly longer when a larger number of network resources were present in the environment. Although the service discovery performance was acceptable in this experiment, we believe that it may cause bottlenecks when, e.g., more than 100 resources are present. The *Overall latency* was 3 seconds when 30 resources were available in the environment, and this value is acceptable to the users as we revealed in the user experience tests with our prototype [2].

IV. Conclusions and Future Work

The autonomic system for application composition presented in this paper is intended for everyday use in ubiquitous environments. Our system composes and adapts applications accordingly to user preferences and user-provided criteria. Thus, the system minimizes the users' cognitive load and increases usability of ubiquitous environments, as the users do not explicitly need to select resources and also deal with their properties. To address this, we designed an allocation algorithm which is capable of finding optimal application compositions in the real environment and for different applications. The problem of finding optimal application compositions belongs to the class of NP-hard problems [14] and is exacerbated by the fact that the problem's search space does not contain information about the direction of the search, as we revealed in [11].

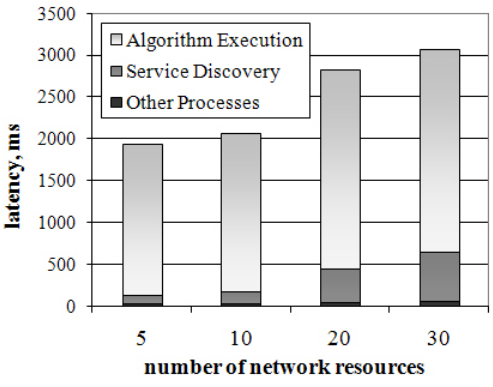

Figure 5. The REACHES Server Performance.

We tested performance of the allocation algorithm in two setups: (i) on synthesized datasets as a stand-alone tool, and (ii) with the real multimedia application when the algorithm was integrated as a part of the REACHES framework. The algorithm demonstrated performance which we found eligible for real-time application composition.

Our future work addresses several issues. At first, we are enhancing autonomy of the REACHES in order to incorporate additional context elements to fully support the user's mobility and traversing of application sessions in adjusting ubiquitous environments. We believe that the role of mobile resources is increasing, and applications are going to rely more and more on these resources in the future.

The second issue addresses the functionality of the algorithm which has to be extended in order to take into account mobile resources. This requires dealing with the real-time resource availability and also predicting the reliability of application compositions. For that reason, we need to include an additional component to the REACHES which is capable of estimating, according to existing behavioral patterns, for how long different resources in the environment stay available. Thus, the allocation algorithm will be able to evaluate different application compositions according to their reliability. The predictor can be designed, as suggested by Del Prete and Carpa [8].

Acknowledgment

This work has been funded by Academy of Finland, Finnish Graduate School in Electronics, Telecommunications and Automations (GETA) and Nokia Foundation. The authors would like to thank Jon Imanol Duran for helping out with the testing and Jiehan Zhou for his valuable comments regarding the paper.

References

[1] BRITE. The boston university representative internet topology generator, http://www.cs.bu.edu/brite/. sept 2009.

[2] Oleg Davidyuk, Ivan Sánchez, Jon Imanol Duran, and Jukka Riekki. Autonomic composition of ubiquitous multimedia applications in reaches. In *Proc. of the 7th Int. ACM Conf. on Mobile and Ubiquitous Multimedia (MUM'08)*, pages 105–108. ACM, 2008.

[3] Sánchez I., Cortés M., and J. Riekki. Controlling multimedia players using nfc enabled mobile phones. In *Proc. of 6th Int. Conf. on Mobile and Ubiquitous Multimedia (MUM'07)*, pages 290–294, Oulu, Finland, December 2007.

[4] Buford J., Kumar R., and G. Perkins. Composition trust bindings in pervasive computing service composition. In *Proc. of the 4th Annual IEEE Int. Conf. on Pervasive Computing and Communications Workshops, (PerCom Workshops 2006)*, march 2006.

[5] Nakazawa J., Yura J., and H. Tokuda. Galaxy: a service shaping approach for addressing the hidden service problem. In *Proc. of the 2nd IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems*, pages 35–39, may 2004.

[6] Sousa J. et al. Task-based adaptation for ubiquitous computing. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, Special Issue on Engineering Autonomic Systems*, 36(3):328–340, 2006.

[7] Sousa J.P., Schmerl B., Poladian V., and A. Brodsky. udesign: End-user design applied to monitoring and control applications for smart spaces. In *Proc. of the Working IEEE/IFIP Conf. on Software Architecture*, pages 71–80, Vancouver, Canada, 2008. IEEE Computer Society.

[8] Del Prete L. and L. Capra. Reliable discovery and selection of composite services in mobile environments. In *Proc. of the 12th IEEE Int. Enterprise Computing Conf. (EDOC08)*, Munich, Germany, sept 2008.

[9] Handte M., Herrmann K., Schiele G., and C. Becker. Supporting pluggable configuration algorithms in pcom. In *Proc. of Int. Workshop on Pervasive Computing and Communications, 5th Annual IEEE Int. Conf. on Pervasive Computing*, pages 472–476, march 2007.

[10] Jurmu M., Boring S., and J. Riekki. Screenspot: Multidimensional resource discovery for distributed applications in smart spaces. In *5th Int. Conf. on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous08)*, Dublin, Ireland, july 2008.

[11] Davidyuk O., Selek I., Duran J.-I., and J. Riekki. Algorithms for composing pervasive applications. *Int. Journal of Software Engineering and Its Applications*, 2(2):71–94, 2008.

[12] Ben Mokhtar S., Georgantas N., and V. Issarny. Cocoa: Conversation-based service composition in pervasive computing environments with qos support. *Journal of Systems and Software*, 80(12), 2007.

[13] Malek S. et al. A decentralized redeployment algorithm for improving the availability of distributed systems. In *Proc. of 3rd Int. Conf. on Component Deployment (CD05)*, Grenoble, France, 2005.

[14] Kichkaylo T. et al. Constrained component deployment in wide-area networks using ai planning techniques. In *Proc. of Int. Parallel and Distributed Computing Symp. (IPDPS03)*, Nice, France, 2003.