A Framework for Composing Pervasive Applications

Oleg Davidyuk, Ivan Sanchez, Jon Imanol Duran, Jukka Riekki

HAL Id: inria-00400314

https://hal.inria.fr/inria-00400314

Submitted on 30 Jun 2009

A Framework for Composing Pervasive Applications

Oleg Davidyuk[†], Ivan Sanchez[†], Jon Imanol Duran[‡] and
Jukka Riekki[†]

[†]Dept. of Electrical and Information Engineering, University of Oulu,
Erkki Koiso-Kanttilan katu 3, Oulu, 90570, Finland

E-mail: {firstname.secondname}@ee.oulu.fi

[‡]University of Basque Country, School of Computer Engineering,
Campus de Leioa, Leioa, 48940, Spain

E-mail: jiduran001@ikasle.ehu.es

Abstract

In this paper, we describe our experiences in developing application allocation algorithms and a framework for composing pervasive applications. The framework supports applications which consist of a set of components residing on physically distributed devices. We will argue that such composite applications gain additional flexibility as they can adapt to the situation at hand. That is, their application components are dynamically allocated and deployed onto the networking nodes by allocation algorithms that satisfy the resource constraints and optimize the user specified criteria. This paper will present three implemented prototypes and a concrete example of a ubiquitous multimedia application which was implemented to test the feasibility of the composition framework. We will also discuss user feedback collected during initial user evaluation study.

1   Introduction

Pervasive computing focuses on applications which satisfy user needs without distracting the users in their daily activities. These applications are

1

characterized by their ability to adapt to different situations (contexts), for example, to changes in the user goal and variance in resource availability. Pervasive applications may engage multiple devices (nodes) simultaneously; this functionality can be achieved by composing the applications from multiple components residing on nodes providing computational, communication or other resources in the environment. Composite applications benefit from higher flexibility as their components can be allocated and then reallocated according to the situation at hand. In addition, composite applications are less prone to resource variation, since each component can be reallocated independently and the reallocation does not involve the entire application.

However, designing a system which supports the dynamic composition of pervasive applications is a complex engineering task. First of all, multiple functional and non-functional requirements need to be satisfied while allocating the application components. These requirements are imposed, for example, by application resource demands, application-specific restrictions and constraints set by the available computing nodes. Secondly, application allocation should be performed according to user specified criteria; hence, the system has to find optimal (or near-optimal) allocations. Thirdly, the system has to cope with highly dynamic environments as both resource availability and user goals vary. Thus, the assignment of application components to the resources has to be performed at run-time with minimal delay in the system's overall response time.

In this paper, we describe our experiences in developing prototypes for application composition and present three prototypes. The most recent prototype is a complete framework which has the necessary functionality for executing composite pervasive applications. It allocates the applications using algorithms which require a low computational load and are highly time-efficient. These algorithms are generic; that is, multiple user criteria and various functional (i.e., application-specific) and non-functional properties (i.e., related to the application's QoS) can be supported without redesigning the code. We present a concrete example of an application which was implemented to test performance and feasibility of our approach. We also present the initial results obtained from our user study.

A number of frameworks for supporting composite pervasive applications have recently been proposed by the research community (e.g.,

2

Aura [1], PCOM [2], COCOA [3], Gaia [4] and CTB [5]). The Aura framework supports the adaptation of task-based applications on several architectural levels and it uses a Knapsack problem solver to calculate the application allocations. The Aura project takes a framework independent approach to expressing application requirements. It focuses on user interfaces which are used to collect requirements from the user and to forward them to the problem solver. Our framework does not have this feature. The Pervasive Component System (PCOM) uses pluggable algorithms for application allocation. However, these algorithms focus on resolving recursive component dependencies and they are based on the greedy approach, which is highly inefficient in large-scale component systems, as we pointed out in our initial prototype (we will discuss this later in the third section of this paper). The Conversation-based Service Composition in Pervasive Computing Environments (COCOA) prototype includes ontology-based service discovery to enable semantic reasoning and a mechanism to enable QoS attribute matching of application and service descriptions. However, we are not aware of any complete implementations of the COCOA framework which are capable of executing applications. The Gaia is a middleware which supports composite application in smart spaces and uses a planning algorithm to calculate application allocations. But, it only supports applications with a proprietary architecture. The Composition Trust Binding (CTB) project focuses on security issues related to composite services which are out of scope of our work.

Our framework differs from these related works, as our main goal is to design a high-performance system for large scale composite applications. In addition, we aim is to minimize the time needed to calculate application allocations, also for very large applications. We are not aware of any other generic application allocation algorithms which are capable of supporting such a wide range of application areas, such as load-distribution and task-based computing, without redesigning the code.

Next, we will present the concept of our framework and explain step by step how it performs the application composition.

2   Application composition concept

Application composition increases the flexibility of application adaptation and also creates a number of interesting applications. As proposed

by Budford et al [5], the potential categories of application composition are virtual devices, multimodal interfaces, and load distribution. The virtual device approach (also known as resource sharing) assumes that aggregating functionality across many resource-limited devices extends capabilities of each single device used in the aggregation. For example, a portable video camera connected to a mobile phone will enable 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 nodes with the required computational capacities. This functionality is required, among others, in such application 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 pervasive devices. For example, a mobile phone's UI (input) can be used to control the playback of a video clip on a wall display (output). In the next section, we will present an application scenario where application composition has been used to build an example application.

The application composition concept assumes that applications are assembled from a set of components which may reside on physically distributed nodes. The applications are started and then adaptation takes place in three phases. First, the available nodes are discovered. Then, the components are allocated, i.e., a deployment plan which denotes how the application components are deployed onto the nodes is produced. And finally, in the last phase, the application components are deployed and configured.

Figure 1 shows the key components of the system and also illustrates the three phases of application composition. The Application Assembly component controls the application's lifecycle and also performs the application adaptation. The Resource Management module monitors the utilization of resources and performs application deployment. The Service Discovery component handles information about available nodes in the environment and provides a matchmaking functionality.

The first phase of the application composition starts when the user activates the application. In this phase, the application assembly searches for the available local nodes using the Service Discovery component. By
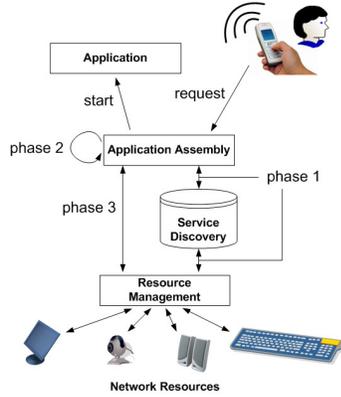
4

Figure 1: Application composition concept

local nodes we mean nodes in the close proximity to the user, for example in the same room. By remote nodes we mean nodes located physically further away from the user. Conventional protocols, such as Bluetooth or UPnP can be used to search for both local and remote nodes. The Service Discovery searches for nodes according to their functional and non-functional properties. The former denote a nodes' ability to provide certain services. These properties are descriptions which resemble interface statements in Java. The descriptions can also rely on ontologies to enable semantic search, as suggested by Ben Mokhtar et al [3]. Functional properties can be used to indicate the availability of specific resource (e.g., a file or a user profile) at a certain node. The non-functional properties describe device resource constraints, for example maximum available memory or computational resource capacity. Furthermore, these properties may be used to implement user access policies.

The applications are specified using descriptions similar to those used for nodes. It should be noted, that the application and node descriptions must contain same property types to enable property matching during the application allocation phase (the phase is explained later). The ap-

5

plication and node descriptions are stored and managed by the Service Discovery component. This component also enables matching between discovery requests (coming from the Application Assembly module) and the descriptions stored in the Service Discovery component.

In our concept, a node can host a single component or component groups, if such allocation does not exceed node's resource Besides, the functional properties of nodes and components have to be met. These two conditions are ensured during the application allocation phase which is performed by the Application Assembly. This phase is the most important step in the application's startup and execution stages. The goal of the phase is to produce a deployment plan. If no valid plan is found, the service discovery and allocation phases 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 negotiate lower resource demands with the user. In general, the deployment plans may be optimized, for example, according to user specified criteria, user needs and the situation in the environment. An optimal deployment plan 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 earlier deployment plans making them invalid or no longer optimal. In these cases, a reallocation of the application is performed.

Finding an optimal deployment plan which satisfies the numerous functional and resource requirements is a complex problem and solving it requires the utilization of allocation algorithms. These algorithms can be used to find an application structure (i.e., groups of the application components) which satisfies the node constraints after the application components are assigned to the chosen nodes In other words, the algorithms solve the application allocation problem. The algorithms operate 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 turn, formally defines the network resources and their properties (e.g., node resource constraints).

The application allocation problem is known to belong to the class

of NP-hard problems [6]. Additional challenges are caused by the fact that its search space does not contain information about the direction of the search, as we pointed out in [7]. In order to solve the application allocation problem, we apply the theory of evolutionary and genetic algorithms. Both types of algorithms are well suited for problems with the above mentioned characteristics, as discussed in [8]. We will discuss these algorithms in the next section.

After the deployment plan is found (i.e., in the third phase of the application composition), it is then realized by leasing the necessary nodes and deploying the application components onto them. This is achieved using a resource management schema as suggested in [9] . Resource management is out of scope of this paper and, hence, we do not discuss it further.

3 Developed prototypes

We have developed several prototypes of application allocation algorithms and an application where these algorithms are used. In this section, we will discuss the prototypes and the results achieved so far.

3.1 Prototype 1. The Straightforward Evolutionary Algorithm (SEA)

This initial prototype relies entirely on a classical schema described by Michalewicz and Fogel [8]. The algorithm is based on such evolutionary operators, as a tournament selection, one-point crossover and a multipoint mutation. The platform model constraints are enforced using a penalty function and a fixed objective function is used to evaluate the potential individuals. The SEA preserves population diversity by generating random individuals in the initializing phase and also by using a mutation operator after the population initialization.

We implemented the algorithm in Java and compared its performance with that of a greedy allocation schema on synthesized datasets (we used our proprietary solution to synthesize the models). The greedy schema produced solutions by iteratively choosing an application component with the highest resource demands and allocating it to the node with the biggest resource capacity. The greedy schema handled the application and node

constraints iteratively, one by one. The experiments were run on relatively small datasets, in the largest case 10 application components were allocated onto 26 nodes. This was due to the fact that both the SEA and the benchmark algorithm were time inefficient. Besides, in larger models the greedy algorithm was highly unstable and suffered from frequent failures (i.e., in the some case it found solutions in only 5 per cent of the algorithm executions). Although, the SEA outperformed the greedy algorithm in terms of robustness and quality of solutions, its computation times were of considerable magnitude (tens of minutes). The reason for this lies in the SEA's diversity preserving mechanism: the algorithm has to operate with large populations (usually 150-200 individuals) to avoid premature convergence. Another drawback was the large number of algorithm parameters which had to be tuned empirically through a series of experiments. Additional implementation details and a thorough analysis of the experiments can be found in [10].

3.2 Prototype 2. The Micro-Genetic Algorithm (MGA)

The goal of this prototype was to address the drawbacks of the SEA, and also to analyze the characteristics of the application allocation problem in more detail. In this prototype, we used a micro-genetic schema, originally proposed by Coello [11], which is known for its low computational load. The main difference between this schema and the algorithm developed in the first prototypes derives from their diversity preserving methods. Unlike the SEA, where the diversity mainly depends on the mutation operator, the MGA relies on external memory and population reshuffling techniques. This permits the usage of small populations with less than 10 individuals. The MGA is structured in two cycles, namely an external and an internal cycle. The first one controls the algorithm's memory, watches the stopping criteria, and restarts the internal cycle. The latter resembles a classical evolutionary algorithm in miniature, because the MGA only deals with a few individuals. In addition, the MGA has two crossover schemes and two mutation operators to enable the faster convergence of the algorithm. The MGA uses the same penalty and objective function as the SEA.

We implemented the MGA in Java and compared its performance with

the earlier results from the SEA. The experiments were conducted as with the first prototype. However, the largest model contained 16 application components which were allocated onto 32 nodes. Also, additional constraints, which affected the performance of the algorithms and especially their robustness, were introduced. The results showed that the MGA was faster and found better or same quality solutions than the SEA. In addition, the MGA failed less frequently than the SEA. We noticed that our model synthesizer produced datasets where the quality of the solutions found by both algorithms slightly oscillated, and therefore, we planned to use another tool to generate data for the next prototype.

The experiments also revealed that the search space in the application allocation problem does not contain information about the order in which the algorithm should sample the solutions. Also, the values of the objective function varied between neighboring solutions and they were independent from each other. A detailed analysis of the second prototype experiments was published in [12]. The MGA's computation time was already in the magnitude of a few minutes; however, the algorithm needs to be faster in order to be used in a real-time application scenario.

3.3 Prototype 3. Modified Evolutionary (EA) and Genetic Algorithms (GA)

The goal of this prototype was to achieve additional performance by exploiting features of the search problem discovered while testing prototype two. We also planned to use a third-party software router tool to synthesize the application and the network models. In addition, our goal was to make our algorithms generic (i.e., without tailoring their models to a certain application type), thus, our model could have any number of properties, and they could be supported as new objectives without redesigning the code.

Two algorithms, conceptually different from the previous two prototypes, were proposed, the evolutionary and the genetic algorithm. The evolutionary algorithm only relied on the mutation operator and iteratively mutated one individual (no population was used). The EA was expected to have a high search speed but at the expense of lowering the quality of the found solutions. The second algorithm, the GA, used standard genetic operators (i.e., tournament selection, uniform crossover

9

multipoint mutation and elitism) and cyclically evolved a population of individuals. The GA was expected to yield higher quality solutions, but at the expense of increasing computational times.

Another novelty in the designs was the evaluation schema used to compare the individuals. According to the schema, the algorithms first satisfy the platform constraints and when a valid solution is found (i.e., a solution which does not violate any constraint) the algorithms begin the optimization of the objective function. Thus, the application allocation problem is treated as a hybrid search problem, which simultaneously combines the features of both constraint satisfaction and optimization problems. This approach is necessary to decrease the complexity of the search problem and also to support many optimization objectives without redesigning the code. Further implementation details can be found in [7].

The main difference between the third and the second prototype is caused by the models used. That is, the third prototype operates with models in which additional properties and objectives can be added without redesigning the code. This means that the nodes and links in the platform and the application models are not associated with a fixed number of properties. We implemented these two algorithms in C++ and used the BRITE software router tool [13] to synthesize larger application and platform models. We evaluated the performance of the algorithms while increasing the number of model properties and model sizes. We did not compare its performance empirically with the previous prototype for two reasons: firstly, the models were different, i.e., the MGA only operated with a fixed number of properties and the aim of the third prototype was to test the behavior of the algorithms while increasing the number of properties. Another reason is that the third prototype was designed in C++, unlike the MGA, which was designed in Java.

The largest models we experimented with had 80 application components and 240 platform nodes. It should be noted, that in the related work the models were much smaller (e.g., 20 components in COCOA experiments [3]). We also tested the algorithms using models with 6 and 10 properties. The experiments demonstrated that the GA was the slowest in all the cases; however the quality of the solutions it found was 10-15 per cent better than of the ones found by the EA. Besides, the GA was more robust and its failure rate was almost two times smaller than the EA's rate. But, the EA's main advantage was in its exceptional search

speed which was on average 17 times faster than the GA's. Thus, we concluded that the utilization of these algorithms is justified in different situations: e.g., the GA can be used to find initial application allocations (as it produces higher quality solutions) and the EA can be used when the application has to be reallocated (as its search speed is fast). In addition, both algorithms experienced similar performance drops when the number of model properties was increased. The number of properties affected the complexity of the search problem making it more and more complex which consequently lead to longer computation times. The performance of the algorithms is acceptable for usage in real-time systems as their computation time was in the magnitude of milliseconds (in contrast to minutes in prototypes 1 and 2). A detailed analysis of the experiments can be found in [7].

3.4 User evaluation and feasibility studies

To evaluate the adequacy and feasibility of our concept, we tested the ability of our algorithm to allocate an application in a user experiment. We implemented a multimodal user interface application which controls a multimedia player on a large display from a mobile phone. The application played different media files which were streamed from a media server. The algorithm was used to allocate the application onto the available resources, e.g., media servers and computers connected to large displays. We implemented the application using the REACHES platform [14]. This system allows mobile devices to control remote services using the HTTP protocol. The REACHES platform offers service and resource discovering and deployment capabilities, and also provides the graphical interfaces needed to control the allocated resources. In the REACHES, the control events are sent from the mobile devices's UI, they are dispatched and forwarded to the specified service, which then performs the required command, using the allocated resources as needed.

We decided to choose the GA allocation algorithm because of its excellent stability characteristics. Although, it requires longer computational times with large models (e.g., on larger than 50 components and nodes), the performance was not a crucial parameter in our experiment as the models were fairly small. We integrated the allocation algorithm into the REACHES platform in order to allocate the application in the start-up

stage when the users choose a video file to watch.

The participants were 10 students and research staff from the Information Processing Laboratory of University of Oulu. They were asked to use the multimedia player application to watch a set of video clips of different quality. A total of 8 computers with wall screens and 3 media servers were used in the test; all of them had different capabilities (e.g., network connection, screen sizes or computational resource capacity). When a participant started the application, it was automatically allocated onto a media server and a computer connected to a wall display using the algorithm. After the testing, the users were interviewed and later they also filled in a questionnaire. We asked the users whether they felt comfortable using the system or whether they desired more control. All participants reported that they would need more over the algorithm's choices to feel comfortable. We also asked the users to evaluate usefulness of the approach and to suggest situations in which it can be used. All participants mentioned that the concept is useful especially in public places where a user is not familiar with the environment (suggestions were, e.g., an airport or a shopping mall). Overall, the feedback from the users was positive with only a few comments related to the disadvantages of the application scenario. Further details of the feasibility experiment and its analysis will be presented in upcoming publications.

In this paper, we presented the prototypes developed to date, however, our research continues and more results are expected. We will discuss our future plans in the next section.

4   Future work

In the following we present our plans regarding the application allocation concept and the directions of our research. The most critical issue is the performance and quality of the application allocation algorithm. It is important to further increase the search speed while preserving (or, ideally, increasing) the quality of the solutions found. We believe that the GA allocation schema is advantageous for this purpose and, therefore, we plan to modify its genetic operators in order to achieve additional performance. We presume that after the necessary modifications the GA may demonstrate performance matching that of the EA.

Another important issue which we intend to address in our future

work is new application areas of the application allocation concept. As we presented earlier in the paper, the concept has been applied to support applications with multimodal user interfaces. However, we believe that the concept (and also the algorithms) can be used to allocate components, for example, in web services and in grid computing domains.

An interesting aspect is to study human-related aspects of application composition. As the user tests revealed, a lot of effort still has to be put in to create a system which the users will trust. This means that we have to study how much control (over the system's actions) should be provided to users so that they feel comfortable and safe. In addition, it would be interesting to apply soft computing and fuzzy logic theory to take, for example, fidelity and quality constraints into account. These constraints are important in capturing user preferences which will permit users to adjust the application allocation according to their expectations. This will increase the system's overall usability and the user satisfaction.

5 Conclusions

In this study, we presented a framework which supports the dynamic composition and adaptation of pervasive applications. Our aim was to enable the seamless binding of pervasive resources to application components on the basis of the application's functional and non-functional requirements and the current context. Our framework utilizes resources in the vicinity of the user as well as in the network. The allocation of the applications is performed by algorithms based on the theory of evolutionary computing.

We presented three prototypes which focused on these algorithms. We implemented the framework and presented a concrete example of an application which was used to study feasibility and user experience. In the initial user experiments the approach was rated useful by the test subjects.

References

[1] J. Sousa 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, Vol. 36, 3, 2006. pp. 328-340.

[2] M. Handte, K. Herrmann, G. Schiele, C. Becker, *Supporting Pluggable Configuration Algorithms in PCOM.* In Proc. of Int. Workshop on Pervasive Computing and Communications, 5th Annual IEEE Int. Conference on Pervasive Computing, 2007. pp. 472–476.

[3] Ben Mokhtar, N. Georgantas, and V. Issarny, *COCOA: COnversation-based Service Composition in PervAsive Computing Environments with QoS Support.* Journal of Systems and Software, Vol. 80, 12, 2007.

[4] A. Ranganathan and R. H. Campbell, *Pervasive Autonomic Computing Based on Planning.* In Proc. of the IEEE Int. Conference on Autonomic Computing (ICAC 2004), New York, NY, US, 2004.

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

[6] T. Kichkaylo et al, *Constrained Component Deployment in Wide-Area Networks Using AI Planning Techniques.* In Proc. of Int. Parallel and Distributed Computing Symposium (IPDPS'03), Nice, France, 2003.

[7] O. Davidyuk, I. Selek, J. I. Duran and J. Riekki, *Algorithms for Composing Pervasive Applications* Int. Journal of Software Engineering and Its Applications, Vol. 2, No. 2, April, 2008. pp. 71-94.

[8] Z. Michalewicz and D. Fogel, *How to Solve It: Modern Heuristics* Berlin: Springer, 2000.

[9] M. Jurmu, S. Boring and J. Riekki, *ScreenSpot: Multidimensional Resource Discovery for Distributed Applications in Smart Spaces* In Proc. of the 5th Int. Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, Dublin, Ireland, 2008.

[10] O. Davidyuk, J. Ceberio and J. Riekki, *An algorithm for Task-based Application Composition* In Proc. 11th IASTED Int. Conference on Software Engineering and Applications (SEA'07), Cambridge, Massachusetts, USA, 2007. pp 465-472.

[11] C. Coello Coello and G. Pulido, *A Micro-Genetic Algorithm for Multiobjective Optimization.* Lecture Notes in Computer, Science, Volume 1993, 2001. pp 126-140.

[12] O. Davidyuk, I. Selek, J. Ceberio and J. Riekki, *Application of Micro-Genetic Algorithm for Task Based Computing* In Proc. 1st Int. Conference on Intelligent Pervasive Computing (IPC'07), Jeju Island, Korea, 2007. pp. 140-145.

[13] The Boston University Representative Internet Topology Generator (BRITE) http://www.cs.bu.edu/brite/, (Accessed in November 2008).

[14] I. Sanchez, M. Cortes, and J. Riekki, *Controlling Multimedia Players using NFC Enabled mobile phones* In Proc. of 6th Int. Conference on Mobile and Ubiquitous Multimedia (MUM'07), Oulu, Finland, 2007.