Algorithms for Composing Pervasive Applications

Oleg Davidyuk, Istvan Selek, Jon Imanol Duran, Jukka J. R. Riekki

International Journal of Software Engineering and Its Applications
Vol. x, No. x, xxxxx, 2008

Algorithms for Composing Pervasive Applications

Oleg Davidyuk[1], István Selek[2], Jon Imanol Duran[3] and Jukka Riekki[1]

[1]*University of Oulu, Finland*
Department of Electrical and Information Engineering,
{oleg.davidyuk, jukka.riekki}@ee.oulu.fi

[2]*University of Oulu, Finland*
Department of Process and Environmental Engineering,
istvan.selek@oulu.fi

[3]*University of Basque Country, Spain*
School of Computer Engineering,
jiduran001@ikasle.ehu.es

Abstract

Pervasive computing environments enable the composition of applications from components allocated across different devices. The applications have to be composed at run-time, to cope with changes in context and resource availability in the environment. In addition, this functionality has to be automated in order to minimize user involvement in application management. We propose two new algorithms which are capable of dynamic allocation of application components to multiple networking devices. These algorithms optimize the selection of the networking devices and the structure of composite applications according to a given criteria, such as minimizing hardware requirements, maximizing the application QoS or other criteria specified by the user. The algorithms are based on generic models. This allows the approach to be used in multiple application domains. We analyze the performance of these algorithms in a simulated environment and suggest a system that utilizes our algorithms for pervasive application composition.

1. Introduction

The pervasive computing paradigm assumes that computation is performed on networking devices surrounding the user [1, 2]. Therefore, pervasive applications can be composed or assembled from components which physically reside across multiple nodes. Composite applications provide a number of advantages which are not available for monolithic applications. For example, composite applications are able to aggregate functionality across several resource-limited devices, thus sharing their resources and providing so-called virtual device capabilities. The application components can be used to build multimodal user interfaces by combining and controlling inputs and outputs from multiple pervasive devices. Moreover, composite applications enable the distribution of the computational load across multiple devices, which is necessary in computationally demanding application domains such as content-based retrieval, information fusion and semantic search.

However, implementing a solution for the dynamic composition of pervasive applications is not an easy task. The users are mobile and therefore the binding of the application components to the location-based network resources, such as wall projectors, cameras and

interactive displays, has to be realized at run-time. Furthermore, because the situation and the user needs can change, it might be appropriate to choose different sets of resources even for the realization of the same application. These challenges require automated mechanisms that take care of application adaptation and lifecycle management. The mechanisms need to take non-functional requirements into account when modelling the application behavior and the limitations of the available resources. Furthermore, the planning of application composition has to be abstracted: it should not be tailored to certain types of applications with specific properties. Finally, the optimal allocation of resources needs to be determined for cases in which there are multiple ways to compose the same application across networking devices.

Application composition research can be divided into three categories that is the study of frameworks supporting the composition functionality of generic pervasive services, systems for task-based computing, and dynamic component-based systems. The systems in the first group mainly focus on context-awareness and syntactic heterogeneity (e.g., USON [3], Galaxy [4], PCOM [5] to name a few). The second group specifically deals with applications which are abstracted as user tasks. These frameworks allow the user to feed task descriptions directly into the system, thus addressing issues related to designing user interfaces for specifying tasks and user preferences, task representation models and ontologies (e.g., COCOA [6], Aura [7], Gaia [2]). The last group focuses on component-based systems which react to resource variations and adapt their behavior to the set of available resources in the environment. These systems are, for example, Sekitei [8, 9], TimeWeaver [10], and DecAp [11].

Most of the research concentrates on selecting resources according to a given goal [6, 7, 12], whereas our work assumes that the application structure has to be adapted too. Although, a similar functionality is provided by the planning algorithm in Gaia [13], this only supports applications with a proprietary architecture.

In this paper, we present a system architecture for dynamic application composition. We introduce two planning algorithms based on evolutionary and genetic computing. The planning algorithms allocate components onto networking hosts according to a given optimization criterion such as minimizing the application hardware requirements, load-balancing or maximizing the application's quality of service (QoS). The proposed models and algorithms are generic and they are not restricted to a certain type of application domain, this facilitates the extensive utilization of our approach. We analyze the performance and quality of these algorithms in a simulated environment. This paper also includes a related work survey which, to our best knowledge, for the first time provides an aggregate account of topics related to application composition.

The rest of this article is structured as follows. In the next section we survey related work addressing application composition in dynamic component-based systems, pervasive computing and task-based computing. Section 3 presents the application allocation problem. Section 4 describes the application allocation algorithms. Section 5 presents an example application scenario and Section 6 describes a simulation based method of evaluating algorithm performance. And finally, Section 7 discusses the limitations of the model and addresses areas of future work.

2. Related work

Related work can be divided into three categories: frameworks supporting the composition functionality of generic pervasive services, systems for task-based computing, and dynamic component-based systems.

2.1. Frameworks for pervasive service composition

Pervasive computing systems assume that the computation is performed in the environment surrounding the user. Such environments are called smart spaces because they are context-aware and aim to minimize user distraction, thus performing actions on behalf of the user and even make decisions in some cases. For example, Gaia OS [2] is a middleware operating system which provides the following functionalities for applications in smart spaces: mobility, adaptation, context-awareness, and dynamic binding. Gaia enables composite applications which are bound automatically or by the user to access the resources of the smart space. These applications can be reconfigured during the execution as required by the user or the reconfiguration may be triggered if the context changes.

There are two application prototypes, namely ARIS [14] and Active Space [15], based on Gaia OS and they are, to some extent, related to service composition. ARIS is an interactive window manager which supports the relocation of application windows across different shared pervasive devices. The ARIS prototype focuses on human computer interaction techniques and therefore, it addresses problems related to representing relocation tasks and designing interfaces for the manual relocation of application windows.

The Active Space prototype enables the utilization of programming behavior in smart spaces, for example, it uses so-called application bridges which are sets of static interaction rules describing how changes in the context affect the execution of other applications. The prototype also supports composite applications which control a smart space. Although, our work focuses on providing a dynamic application composition functionality, we do not aim at defining smart space behavior. Moreover, the ARIS and Active Space prototypes do not use a planning approach in application composition.

The Event Heap framework [16] has been developed to enable the communication of multi-user collaborative applications for smart spaces. It focuses on the seamless integration of legacy applications with large composite services where multiple users control multiple network resources simultaneously. These resources are static host machines which are remotely controlled by events sent from mobile client devices. The event dispatching is decentralized also enabling many-to-many connectivity within the smart space. The downside of this ad-hoc approach is that it does not use planning in application composition because the network resources used in Event Heap's applications are always static. Our planning algorithm assumes that the environment changes over time; therefore, the reconfiguration of composed applications might be explicitly required by the user or by the changed context (e.g., if a resource becomes unavailable). Thus, our approach permits the applications to use different network resources during the execution lifetime.

The objective of CoDaMoS project (Context-Driven Adaptation of Mobile Services) [17] is to enable the automated composition of pervasive services by tailoring them according to user preferences in a specific context or the capabilities of specific devices. This approach uses a centralized algorithm entirely based on OWL and Protégé reasoning tools. The algorithm is based on backtracking as it tries to find a minimal composition of component instances targeted at a certain terminal client device and it cuts down on the user preferences if no suitable solution is found. Thus, the algorithm only focuses on resource constraint satisfaction and it does not optimize either the structure of the composite services or the set of network resources needed. We propose a hybrid approach which is capable of simultaneous resource constraint satisfaction and optimization based on customizable criteria.

Personal Router [18] is a cognitive personal agent for the selection of wireless access points. The agent supports mobile users who roam between different service providers and service zones by making selection choices on the basis of user preferences. The selection algorithm learns the user preferences from price-quality tradeoff in different contexts. Although, the framework is related to service selection in pervasive environments, it is not specifically created for composite applications. The framework supposes that the applications are monolithic and always deployed onto the user's terminal device. Also, it assumes that service properties are limited to network speed and two types of monetary costs.

Another framework, the IST-Context [12] is a platform for the automatic creation, delivery and management of context-aware services. The central idea of the platform is the context matching engine, which is responsible for performing the decision-making regarding the selection of the appropriate context sources which answer the context requests coming from the users. The engine evaluates the available context sources of a specific service based on the estimated fidelity and the response time of the context sources. It also considers the profiles of the selected services. The framework uses a context selection algorithm which determines the optimal combination of the context sources focusing on the costs related to retrieving context information in relation to, for example, the required response time, refresh rate and accuracy. The IST-Context framework differs from our approach in that it models context services as monolithic entities connected to one or more context sources. We consider it too restrictive, as monolithic services do not allow adaptation at application level which, in our view, is an essential functionality in enabling pervasive applications. For example, a composite service may be deployed across multiple devices in a resource-limited environment, which would not be possible in the case of the IST-Context.

The Composition Trust Binding (CTB) project [19] assures the trustworthiness of software components by enabling explicit representation of trust requirements between components participating in a service composition. The CTB is formulated as a set of rules which define the collection of allowable component instances for implementing a specific service or for processing specific content. Although, we do not target security issues in application composition, our algorithms enable the integration of the CTB and similar approaches via the explicit definition of affinity constraints. These constraints only permit the allocation of certain components from a service composition onto authorized nodes. We will explain the affinity constraints in more detail in section 3**.**

The Context-Aware Service Enabling (CASE) platform [20] proposes a solution for the dynamic adaptation of composite context-aware services. The platform consists of a composition service and a discovery service. The composition service locates the service components using the discovery service which obtains information from a context agent. The context agents provide access to the entity's context (e.g., people, places and things). The composition service is based on semantic matching and OWL-S. It also uses a similarity function to evaluate the constructed composite services and to measure the achieved functionality taking some non-functional constraints into account. However, the CASE platform does not provide any solutions for QoS-enabled optimization of composite services. In addition, we are not aware of any implementation of this approach.

The Ubiquitous Service Oriented Network (USON) architecture [3] focuses on the provision of services in ubiquitous computing environments. Service composition in

USON is limited to elementary attribute matching in the XML templates and the use of parameter resolution via the distributed dictionary engine, a proprietary solution which resembles the Semantic Web.

The Galaxy framework [4] concentrates on the semantic-free integration of devices in pervasive environments and it supports composite services which control and cooperate with the devices. Galaxy assumes that the composite services are described in an environment-independent manner by the service vendors or the end-users who also share service templates with each other. The composition of services is performed by a generic (not-type based) service lookup, such as "find a service which displays a list of songs acquired from a network" and it is restricted to XML template matching.

The objective of the Pervasive Component System (PCOM) [5] is to automate the configuration and runtime adaptation of component-based applications using a set of pluggable assembling algorithms. These algorithms compute a valid application configuration based on the functional properties required by component interfaces. The system supports composite applications with recursive dependencies which force the assembling algorithms to resolve configurations for only one component at a time thus adopting the greedy approach. While sufficient in some cases, the greedy approach has a number of drawbacks, as has been already proved in [9] and later in [21]. Firstly, it does not guarantee to find a valid solution when one exists. Secondly, the greedy approach fails to achieve optimality in terms of minimizing the overall resource usage or other criteria in the presence of multiple constraints. Besides, PCOM assembling algorithms only consider the functional properties of the applications, thus limiting the configuration of the assembly to elementary property matching.

2.2. Frameworks for application composition in task-based computing

Pervasive computing has recently evolved into the task-based computing paradigm, which supposes that users directly provide the system with descriptions of their tasks. The tasks are abstractly described in terms of required functionality and, in some cases, they include non-functional parameters such as fidelity and quality of service (QoS) constraints. The system then dynamically realizes the user tasks by binding them to the available network resources.

Multiple research efforts have addressed this problem. For example, the COCOA framework [6] supports the dynamic composition of user tasks and it also contains semantic language for specifying services and tasks, service discovery mechanism enhanced with semantic reasoning and QoS attribute matching. COCOA models user tasks as service conversations (or service workflows) which are dynamically integrated with the available service instances by a conversation-based matching algorithm. The algorithm is capable of QoS attribute matching, but it only focuses on QoS constraint satisfaction and does not optimize the selection of the available resources. Although, the workflow approach enables the detailed description of application functionality (in terms of required capabilities) it does not capture, for example, application network behavior. In other words, the COCOA's matching algorithm does not take the non-functional properties of the application links into account.

Another system, Gaia OS, introduced a STRIPS planning algorithm [13] for task-based computing, which takes abstract user goal description and the user's current context into account. The algorithm finds a sequence of actions which lead to the best realization of the

user task. This sequence of actions is later executed by the Gaia application framework, which ensures that none of the action executions fail because of resource unavailability. However, Gaia's planning algorithm is too restrictive; it only allows planning for Gaia applications which are based on proprietary Model-View-Controller architecture. Our planning algorithm differs from Gaia's planner as it supports framework independent applications and does not restrict the environment in a set of predicates. That is, we assume that the user's goal is achieved by deploying all the application components onto available network resources regardless of the deployment order. So, our algorithms produce deployment plans, which only specify the assignment of each application component to a network resource according to certain criteria.

The Aura project [7] calls for the adaptation of task-based applications on several levels and focuses on selecting and controlling applications so as to minimize the distraction of the user. The Aura project defines a vocabulary for expressing framework independent requirements and fidelity constraints, a set of graphical user interfaces for collecting user preferences, and an algorithm which performs the automatic configuration of the user tasks. The algorithm is based on the Knapsack problem solver and aims to maximize user task feasibility in a specific context which is the abstract measure of "user happiness". In our work, we do not consider how the user provides a description of his tasks and preferences to the system. But, our algorithm is capable of finding solutions according to the aforementioned task feasibility function. That is, the algorithm uses a customizable objective function which can also include additional components, such as criteria defined by the user. More details about this objective function can be found in the next section.

Perttunen et al [22] has proposed a model of QoS-based service composition which is integrated into a resource management schema. In their system, the composition of services is validated via context-based criteria. The criteria define the context in which the composition is enabled and the criteria also include proximity and temporal user properties. In their model, a separate similarity function is used for each property type in addition to the functions used for static and dynamic QoS determination or service composition candidates. However, their system does not use any planning algorithms to optimize the criteria.

2.3. Application composition in dynamic component-based systems

A number of algorithms have been designed for application allocation in dynamic component-based frameworks. Niz and Rajkumar suggested the TimeWeaver system [10] which aims to minimize the hardware requirements in real-time embedded systems. According to their approach, each application has functional and non-functional properties and they can be further partitioned into two or more pieces. Then the system will automatically insert communication code at the partitioned points and assign the pieces to available processors using a bin-packing algorithm. In contrast, our algorithm does not permit the further partitioning of application components, as we believe that partitioning of components increases the consumption of network resources and will cause the degradation of the application response time.

DecAp [11] and Anke et al [23] are similar planning approaches which both, focus on finding deployment plans for component-based frameworks in relation to resource consumptions and application availability. Both algorithms calculate application

availability in terms of the success probability of request execution within a given interval of time. Also, these algorithms consider networks where connections are not permanently available and where the handling of handoffs is necessary in the application layer (e.g., in wireless networks). Thus, the algorithms introduce a notion of "partial visibility" which means that each network host has a "domain" or a set of all the hosts of which it is aware. In our scheme, we assume that each host is accessible and believe that the handling of network partitioning can be delegated to the Service Discovery. Also, our models are generic, so the algorithms are capable of finding solutions in the presence of multiple constraints and resource properties. In contrast, the DecAp [11] algorithm only considers two non-functional properties (memory and the frequency of communication) and the system suggested by Anke et al [23] only considers three (memory, CPU capacity and communication bitrate).

Sekitei [8] and modified Sekitei [9] are planning algorithms which focus on QoS constraint satisfaction and the improvement of throughput while also aiming at reducing the overall computation load of the hosts. The deployment plans found by the algorithms also fulfill the deployment goals specified by the user, such as the availability of a certain application component on a specified host. However, these algorithms optimize the application allocation by injecting additional components along the communication paths. We consider this functionality to be framework-dependent and therefore we do not target it in our work.

3. Modeling the Application Allocation Problem

This section provides a conceptual overview of the application allocation problem and outlines an architecture for a system utilizing the application allocation algorithms. The core of the system consists of the Application Assembly, the Resource Management and the Service Discovery. These components control the applications' lifecycle, monitor the utilization of the network resources and perform dynamic application adaptation when necessary. The Application Assembly uses planning algorithms to find an optimal deployment plan which specifies how the application components are allocated onto the network resources. A deployment plan may become infeasible during the application execution due to resource unavailability or changes in user preferences. Therefore, the Application Assembly has to reallocate the executing application at run-time if needed.

The Service Discovery is a ubiquitous database which manages declarative descriptions of applications and network resources. Its main function is to find matches between discovery requests and the descriptions stored in the database. The Service Discovery may use ontologies to enable semantic matching as suggested, for example, in [6].

The responsibility of the Resource Management is to monitor and control the environment and also to trigger application adaptation, if some of the application components start to consume more resources than expected. The overview of the system is shown in Figure 1.

The application allocation problem is defined by an application and a platform model. The application allocation algorithm uses these two models as an input and computes a deployment plan, which determines the mapping of the application components onto the resources from the platform model. Then, the deployment plan is executed by the Resource Management which performs the actual deployment of the components. The Resource Management is also responsible for triggering the adaptation of the application, if some of the previously used resources are not available anymore.

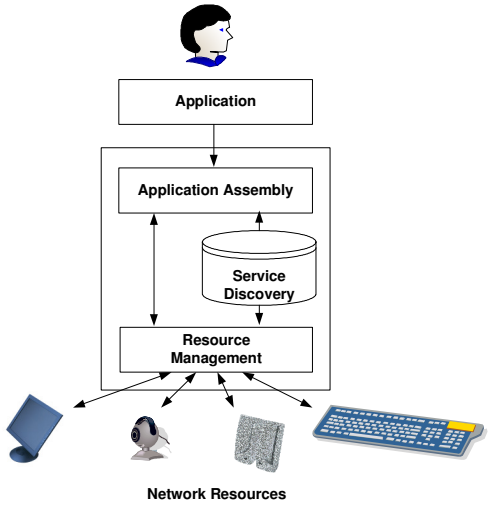

Figure 1. Conceptual overview of the system's architecture.

In such cases, the Application Assembly retrieves information about resources available at that moment from the Service Discovery and then produces a new deployment plan. The Resource Management then redeploys the application components according to the new plan and the application execution continues.

3.1. The Application Model

The application model is formally described as a connected graph that captures the application's topology. Each node of this graph represents an application component, and communication between two components is modeled by using graph links. The application components may communicate using, for example, request/reply or publish/subscribe mechanisms. An example application model, containing 4 nodes, 3 links and 5 properties (memory, cpu and bandwidth resource capacities, the link and node security properties) is shown at Figure 2.

Each element of the application model can have multiple properties which specify a non-functional component or link requirements, such as computational resources, up or downlink channel capacities and memory demand. Although, a greater number of properties guarantees a more detailed description of application resource behavior, it also increases the computational load and the memory consumption of the algorithm. As has been demonstrated in [24] that 5 properties is enough to capture the most important requirements of any system.

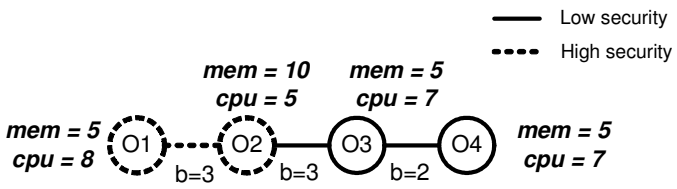

Figure 2. An example application model.

Each property can be expressed by a float, an integer or a Boolean value. The values of the application properties may be set by measuring the performance of the application under different workloads and by then recording the data onto resource profiles. For example, a two bucket profile [25] can be used to describe the properties of application network behavior. It is important to notice that if the values are not fed into the models accurately, the Application Assembly will produce less feasible deployment plans. The task of estimating resource requirements is important for the correct functioning of the application allocation algorithms, but it is beyond the scope of the paper.

The affinity constraints restrict the deployment of a component by specifying a set of allowed hosts for that component. These constraints are also necessary in cases where the user explicitly requests the use of certain resources for an application. For example, while watching a movie, the user may touch an RFID tag associated with a wall display. This action is a manual request to adapt the application component that is responsible for rendering the video. In this case, the wall display must be used. The affinity constraints are also helpful if an application component requires access to specific material (e.g., a file or a user profile) that is only available at a certain node.

3.2. Platform model

The platform model represents the application execution environment. That is, the graph's topology defines the network topology, while the graph nodes are computational hosts which are available for application allocation. We assume that each host is able to communicate with all other hosts in the network, thus the platform model is always a fully connected graph (see an example platform model in Figure 3.) This is a simplification, a more complete approach would include the decentralized networks where the algorithm (or its parts) does not know the current state of the whole system. For example, each host may only be aware of hosts which belong to the same domain (as suggested by Malek et al [11] and Anke et al [23]). This approach allows the allocation algorithms to handle network resources which are not centrally accessible. We delegate this functionality to the Service Discovery, which may use, for example, a distributed service discovery protocol to address this issue. However, it is out of the scope of our work.

Each host in the platform model can allocate a group of application components, if the resource constraints of the host are not violated. These constraints determine, for example, the maximum memory or computation resource capacities.

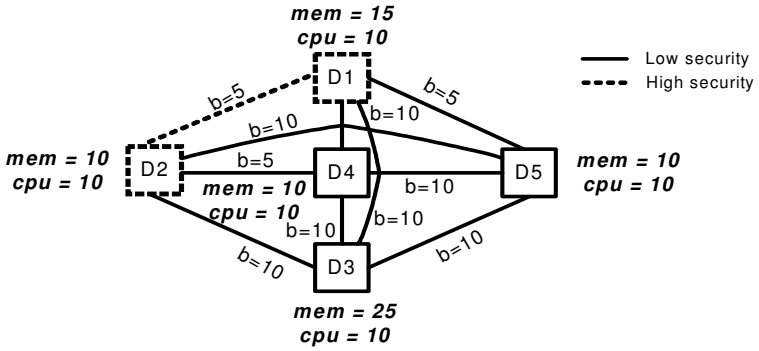

Figure 3. An example platform model.

The platform model properties can have float, integer or Boolean variable types. Both, application and platform models must have an identical number of properties which are matched by the algorithm.

3.3. Generic objective function

The aim of the allocation algorithms is to find an application structure (i.e. groups of application components), which satisfies the platform model constraints after assigning the application component groups to the platform nodes.

In general, the optimization goals for application allocation may vary, therefore the algorithms optimize a generic objective function which supports new objectives without redesigning the algorithm's code. Currently, our algorithms support the following objective function which needs to be minimized:

$$F_{obj} = f_B + f_D + f_V \qquad (1)$$

where

- f_B is the ratio of the network link bandwidth used in the allocation to the sum of the bandwidth required by all the component links in the application. This objective minimizes the overall network traffic by allocating communicating components onto the same host.
- f_D is the ratio of the number of hosts used in the allocation to the total number of application components in the task. This characteristic minimizes the time needed for the actual deployment of components, and
- f_V is a standard statistical measurement of variance in the processing of capacity usage in the hosts, that is, the variance of free capacity in the hosts after the allocation of the components. This objective balances the server load, so that the utilization of each host is within a desired range.

4. Application allocation algorithms

4.1. The solution representation and the three-phase validation schema

We present two application allocation algorithms based on generic application and platform models. These algorithms are Evolutionary Algorithm, (EA) and Genetic Algorithm (GA). The EA is based on mutation and it only uses one candidate solution (i.e., an individual or a possible solution) for searching. The GA (it originates from our earlier work [26]) which is based on mutation and crossover operators and it evolves a population (i.e. a group of candidate solutions) for searching. We develop the EA as we expected it to create a better balance between speed and quality in searching.

The efficiency and the speed of the heuristic search algorithms greatly depends on the algorithm's design, which is mainly affected by two factors: (i) the proper representation of the solutions, as it decreases the difficulty of the search problem; this has been shown by Liepins and Vose [27]; and (ii) the implementation and efficiency of the search operators and the fitness function.

The main objective in selecting the representation for the candidate solutions was to preserve the simplicity and the memory efficiency of the algorithms. The candidate solutions

have a direct representation [28]. An example candidate solution containing 6 application components and 3 hosts is shown in Figure 4. As the example shows, the length of the candidate solution (i.e., individual) is equal to the total number of components in the application. The positions in the solution representation are called genes and their values are gene values. The number in the i^{th} gene position denotes the host identity (id) which allocates the i^{th} component.

The application allocation problem becomes uncorrelated regardless of the objective function after the direct candidate solution representation is applied. This type of problem is challenging for search algorithms, as explained by Jones and Forrest [29]. In this type of problem, the objective function values of the neighboring individuals are different and independent from each other and the structure of the search space contains no information about the order in which the solutions should be sampled. So, while evaluating individuals, the algorithm cannot make any assumptions on the distance to the optimal solution. In addition, in our previous work [26] we discovered that increasing the problem size leads to longer computation times and higher failure ratios of the algorithms. To address these issues we implemented an approach which assumes that infeasible solutions have superiority over feasible ones [30].

According to this approach, the algorithms only handle infeasible solutions until they at least produce one feasible solution. That is, the algorithms do not calculate the values of the objective function and they perform constraint satisfaction only. The objective function is optimized later, when the first feasible solution is found. The idea behind this is to keep the algorithm design simple and efficient. Besides, this approach is also necessary for supporting new objectives (e.g., specified by the user) without touching the algorithm's design. To enable the aforementioned approach we use a three-phase evaluation schema which performs as follows.

Each individual is assigned with a bit string called a validation vector which indicates the feasibility of the individual (see Figure 4.). The length of this vector is equal to the length of the individual. Before the evaluation begins all the validation vector is initialized that is all the bits are set to zero (this means that the individual is feasible). The validation vector plays a key role in our design because it decreases the complexity of the problem and improves the performance of the mutation and the crossover operators by making them guided.

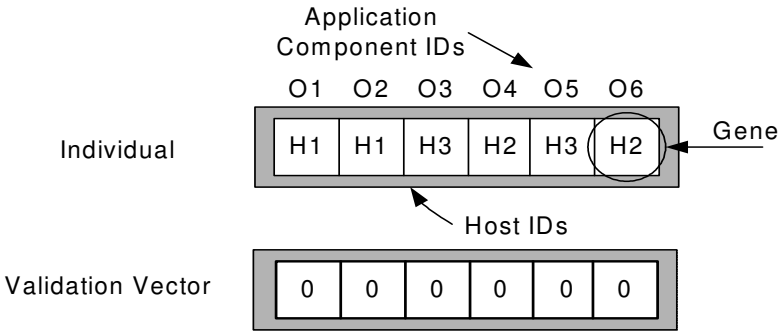

Figure 4. The representation of a candidate solution (or an individual) and the corresponding validation vector which does not indicate violations.

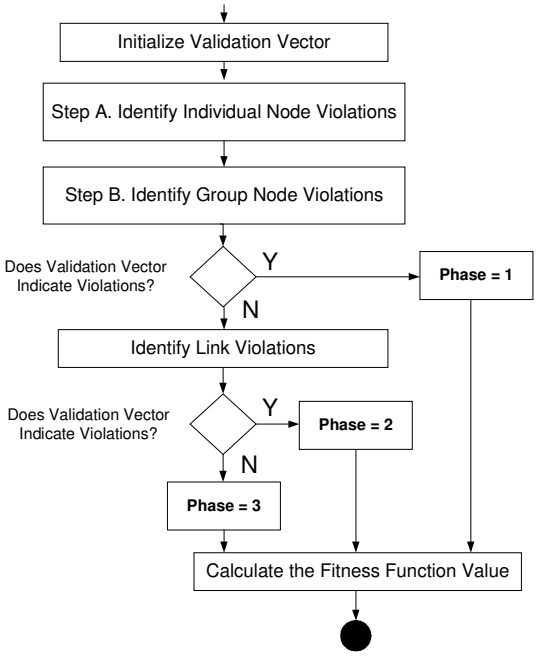

Figure 5. The three-phase evaluation schema.

The evaluation of the candidate solutions always starts with phase one and the process then proceeds towards phase three as follows (see Figure 5).

Phase 1. In this phase, the algorithm only checks if the solution satisfies the node constraints. This operation is performed in two steps.

- **Step A.** In this step, the algorithm checks whether any of the application components of the candidate solution violate node constraints when allocated. If a component does violate them, the algorithm changes the bit value representing the component in the validation vector to '1'. If no violation is detected, the value is zero. At the end of this step, the algorithm has detected all the single violations which take place after a component is allocated to a certain node. The group violations are considered in the next step.

- **Step B.** In this step, the algorithm checks whether groups of components from the candidate solution violate any node constraint. That is, the algorithm allocates components from each group to the determined host one by one and if a component causes its group to violate a constraint, the algorithm assigns the value '1' to the validation vector position which denotes the identity of the component. If the validation vector does not indicate any violations by the end of the step (i.e., all the values are zeros), the algorithm proceeds to the phase 2. Otherwise, the validation of the individual ends and the algorithm calculates the individual's fitness function value (Eq. 2).

Phase 2. Here, the algorithm only considers the link-related constraints. It allocates the component links across the platform links and then checks whether the component links violate the platform link constraints. When a link violates a platform link constraint, the

algorithm changes the validation vector in the following way. It randomly chooses one of the application components connected by the violating link and changes its bit value in the validation vector to '1'. If the validation vector indicates one or many violations by the end of the phase, the algorithm stops the evaluation and calculates the fitness function value (see Eq. 2). When no violations are detected by the end of this phase, the algorithm proceeds to phase 3.

Phase 3. In this last phase, the algorithm calculates the fitness function value according to Eq. 2. This phase is essential to performing the optimization of the objective function.

$$fitness = \begin{cases} -4 - \dfrac{I}{A} & \text{if calculated in phase 1} \\ -2 - \dfrac{I}{A} & \text{if calculated in phase 2} \\ -F_{obj} & \text{if calculated in phase 3} \end{cases} \qquad (2)$$

where

- I is the number of the components violating one or more constraints.
- A is the total number of components in the application.
- F_{obj} is the objective function value.

The fitness function (shown in Eq. 2) is based on the so-called clustering method. This means that the algorithm determines the phase of an individual (first, second or third) according to its fitness function value. So, the fitness function values always fall into one of the following intervals: [-5; -4] or [-3; -2] or [-1; 0]. The first interval denotes individuals from the first feasibility phase, the second interval from the second phase and, finally, the third interval denotes the feasible individuals.

The presented three-phase evaluation schema uses a hybrid approach, where constraint satisfaction and optimization tasks are performed separately. We found this approach convenient as it allows the use separate algorithm configuration in each phase, for example, in the optimization phase the sizes of populations and the number of the crossover points can be decreased. Besides, this approach does not use any penalty function to distinguish feasible and infeasible solutions, thus it also saves computation time.

4.2. The genetic allocation algorithm

The flowchart of the genetic allocation algorithm is presented in Figure 6. The algorithm starts with a randomly generated population and it evaluates the individuals using the three-phase schema explained earlier. Then, the population cyclically evolves using standard genetic operators such as tournament selection, crossover, mutation and elitism.

The algorithm uses the uniform crossover scheme (presented in Figure 7) which guarantees high information exchange rate between parent individuals. The child genes are assigned according to the values of the parents' validation vectors. In the case of an identical value, the gene is always copied from the second parent (parent B in Figure 7.).

The algorithm applies the crossover operator to individuals which belong to the same validation phase. The algorithm achieves this by sorting (ordering) the population. The phase of an individual is determined by its fitness value (see Eq. 2).

The mutation schema used by the genetic algorithm depends on the validation phase of the individual and the percentage value of violations in the validation vector. The feasible individuals (i.e., at the third phase) always mutate by copying a random gene into another randomly chosen gene.

The individuals in the first phase mutate using a multipoint operator. The operator checks the validation vector corresponding to the individual and mutates all the genes whose bit values indicate a violation (that is, contain '1').

The second phase individuals mutate according to the percentage value of violations in their validation vectors. That is, when a vector has 30 percent or fewer bits indicating a violation, the individual mutates its genes with 0.5 probability, otherwise the probability of mutation is set to 0.2. These probability values were defined empirically to minimize the randomness caused by the operator.

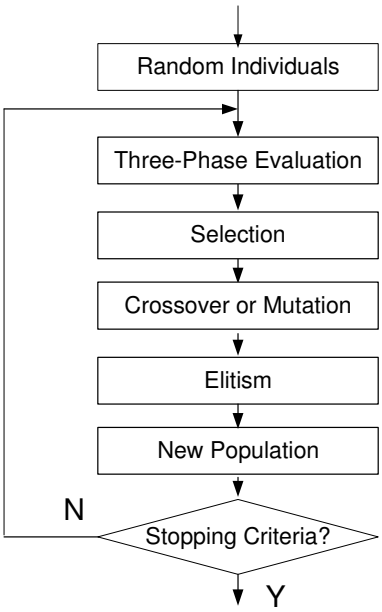

Figure 6. The flowchart of the genetic algorithm.

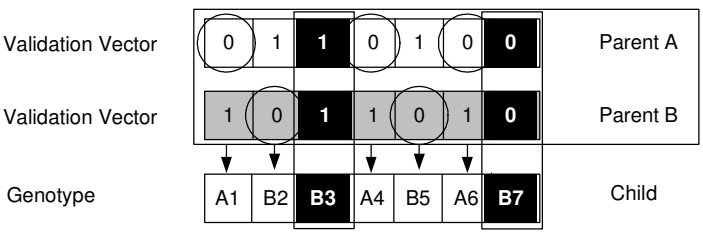

Figure 7. The uniform crossover schema.

The algorithm stops when a maximum number of individual evaluations or a limit of generations with no fitness improvement is reached.

The genetic algorithm only has two parameters: the population size which is set to the length of the individuals and the tournament size which is set to 2. These values were defined empirically during the initial testing of the algorithm.

4.3. Evolutionary-based allocation algorithm

This algorithm has a simple design and it is entirely based on mutation operators. The mutation probability values were determined during the initial testing (see section 4.2.) The flowchart of the evolutionary allocation algorithm is shown in Figure 8.

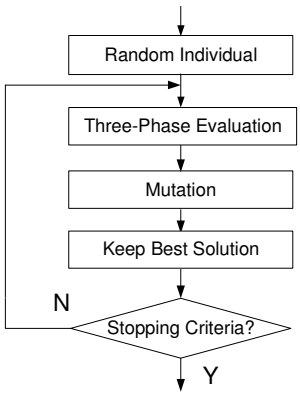

Figure 8. The flowchart of the evolutionary-based allocation algorithm.

5. Example Application Scenario

To give a concrete example how our algorithms are used, we present the following scenario (which we have been originally suggested in [26]):

> "John decides to see a movie and he needs to assemble an eMovie application which consists of four application components: local and remote user interfaces (UIs), AV playback and a media trading service. When John watches the movie on the embedded screen of his mobile device, the device allocates the local UI and AV playback components. However, if John watches the movie on a larger external display, his mobile device only allocates the local UI component. In this case, John's device is used as a remote control unit to control the remote UI component which uses the large display. The AV playback component synchronizes the audio and video streams received from the online media trading service. AV playback can also compress the streams to match the capabilities of the end-point rendering device. When John starts the application, the AV playback and the remote UI components are allocated to the available devices. The mobile device uses the application allocation algorithm to find an optimal allocation for these components. After this, the needed resources are allocated and leased; the components are deployed and configured so that John can enjoy watching his movie."

This application consists of 4 components, that is, the local UI, the remote UI, the AV playback and the media trading service (MTS). We use four properties to capture the application requirements: memory (mem), CPU, bandwidth (b) and screen size.

We suppose that the platform model used in the scenario consist of 6 network resources which are the user's mobile terminal (MT), the public monitors (M1 and M2), the servers (S1 and S2) and the media server (MS). Two devices, MT and M1 are connected via a wireless link, and the others are connected via a fixed network. As the media server is the only server providing streaming functionality, an affinity constraint is used to place the MTS to the correct server. The Media Server does not allow the allocation of other components, therefore its resource capacities are set to zero. Since the affinity constraint dominates the resource requirements, the MTS's requirements are set to zero. As the user controls the application via the Local UI, we also constrain the allocation of the Local UI component onto the user's mobile terminal using another affinity constraint.

The structure of the eMovie application and the available network resources are shown, in Figures 9 and 10, respectively. Although, we depicted the application structure as a graph, in practice, the application components are specified using XML descriptions which are retrieved from the Service Discovery.

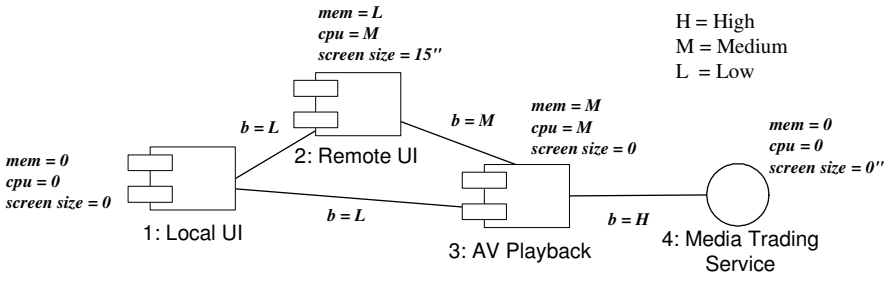

Figure 9. The structure of the eMovie application.

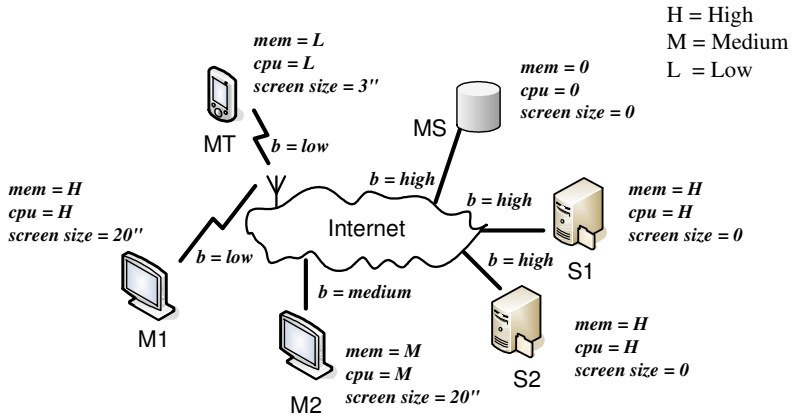

Figure 10. An example of the platform model.

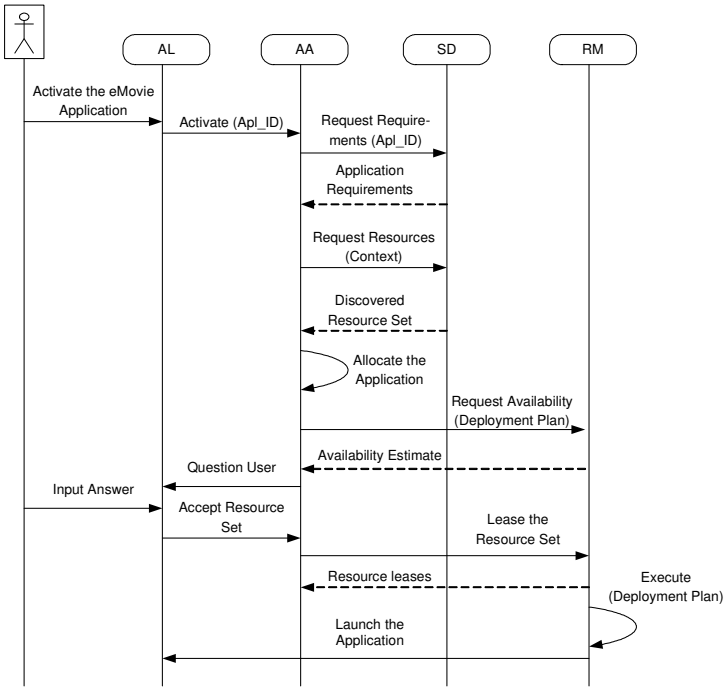

Figure 11. The eMovie application starting process.

Next, we explain how the Application Assembly allocates the eMovie application components onto the network resources. The process begins when the user requests to start the eMovie application via the Application Launcher's (AL) user interface (see Figure 11). The Application Launcher resides in the mobile terminal and it is responsible for application deployment. IT forwards the request to the Application Assembly (AA) which then retrieves the application requirements from the Service Discovery (SD). The Application Assembly queries the Service Discovery for the suitable network resources as well. It may include user preferences and context in the query; for example, it might request resources from a certain area.

Then, the Application Assembly performs the application allocation. If a valid allocation is found, the Application Assembly requests the availability of the resources listed in the deployment plan (which specifies the application allocation) from the Resource Management which, in turn, determines the availability estimates of the requested resources. The Application Assembly requests a confirmation from the user via the Application Launcher UI. That is, the user is asked to accept the estimated time when he can start to use the application. If the user accepts the estimated delay, the Application Assembly leases the resources from the Resource Management which then also executes the deployment plan, i.e., it places the application components into the chosen servers and terminals. Finally, the application is started and the control is transferred to the Local UI.

In our example, we assume that the Application Assembly uses the evolutionary allocation algorithm (presented in section 4.3.). The Application Assembly performs the allocation of the eMovie application as follows.

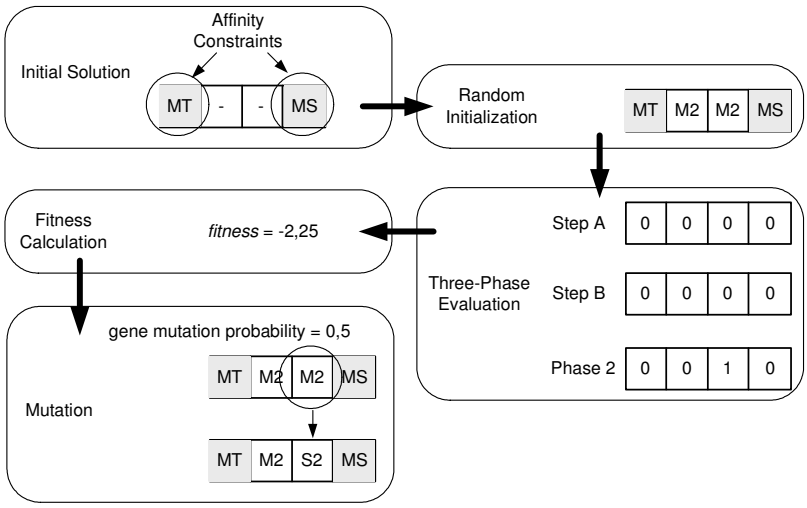

Figure 12. The candidate solution evolution.

The evolutionary algorithm starts with the initialization phase (see Figure 8) in which one candidate solution is generated at random. The evolution of the candidate solution during one cycle of the algorithm is shown in Figure 12. The algorithm evaluates the solution using the three-phase evaluation schema (explained in section 4.1.). The evaluation process ends in the second phase, because the AV playback (the component number 3) violates the M2's bandwidth resource constraint. Therefore, the resulting validation vector contains "1" in the position of the AV playback. After the evaluation, the algorithm calculates the value of the fitness function according to the phase of the candidate solution. Then, it applies the mutation operator to the gene which contains the link violation. The gene is found with the help of the validation vector. The algorithm finishes the cycle and returns to the three-phase evaluation step after saving the best solution. The algorithm stops when it reaches the maximum number of cycles.

6. Experiments

We have implemented the genetic and the evolutionary algorithms in C++ and compared their performance in simulated models. The comparison metrics included computation time, average fitness value and failure ratio. The failure ratio was calculated as the percentage of experiments in which the algorithm fails to find a solution out of the total number of experiments. To make sure that all the experiments finished within reasonable time, we limited the number of fitness evaluations to 1 million. This means that the evolutionary algorithm performs 1 million search cycles, whist in the case of the genetic algorithm the number of cycles is different because of the nomadic population sizes. The population size in the genetic algorithm was set to the application size.

The measurements reported in this section were taken with an AMD Opteron 270 dual core machine, running Red Hat 4.1.2 Linux. It should be noticed that we only used one processor core to execute the algorithms.

All the application and platform models were generated using the Boston University network topology tool BRITE [31]. An example platform model generated by BRITE is

shown at Figure 13. The BRITE tool was also used to synthesize the property values of the models. We generated the application and the platform models in 15 iterations and increased the application model size by 5 nodes in each subsequent iteration. The size of the platform model was always set to the application model size multiplied by three. That is, in the last iteration, the platform model size was 240 nodes. Although, we do not find this scenario with an application consisting of such a large number of components realistic, it gave us insight into the scalability limits of the algorithms. In contrast, algorithms were tested using smaller applications in related work (e.g., 20 components in COCOA [6] and 3 components in Sekitei [8]). One of our aims was to analyze how the number of model properties affects the overall performance of the algorithms. That is, for each combination of the application/platform model sizes two different model sets were synthesized, the first set contained models with 6 properties (marked as EA6 and GA6 on the graphs), and the second with 10 properties (marked as EA10 and GA10 on the graphs).

The experiments were performed as follows. Both algorithms were run until they either found a first valid solution or the maximum number of fitness evaluations was reached. If one of the algorithms failed, and another one found a solution, the graph models were resynthesized and both the algorithms were restarted. All measurements reported in this section were recorded over 100 algorithm executions.

In the first experiment, we measured the computational time and calculated the average individual fitness values. Figure 14 demonstrates how the scalability of the algorithm is affected by the number of nodes in the platform and the application models.

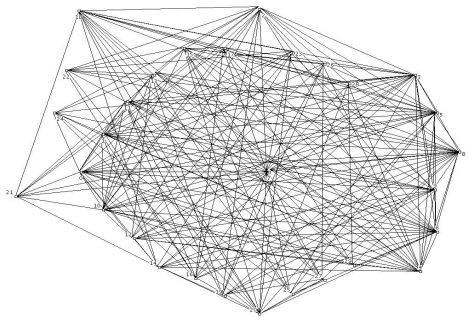

Figure 13. An example platform model graph (30 nodes), generated by the BRITE tool.

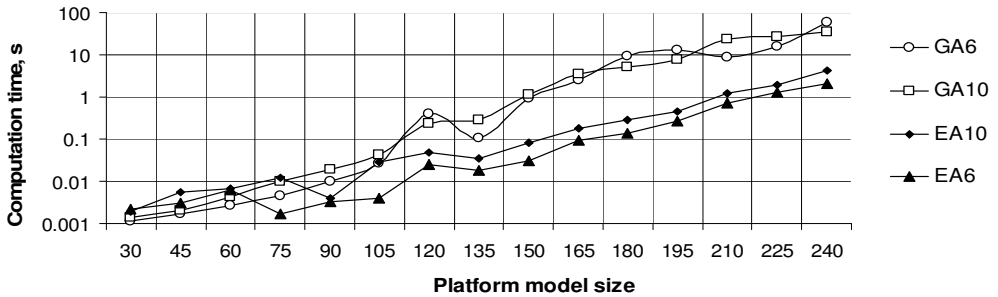

Figure 14. Computational overhead of different algorithms (graphed on a logarithmic scale). The platform model size was set to the application model size multiplied by three.

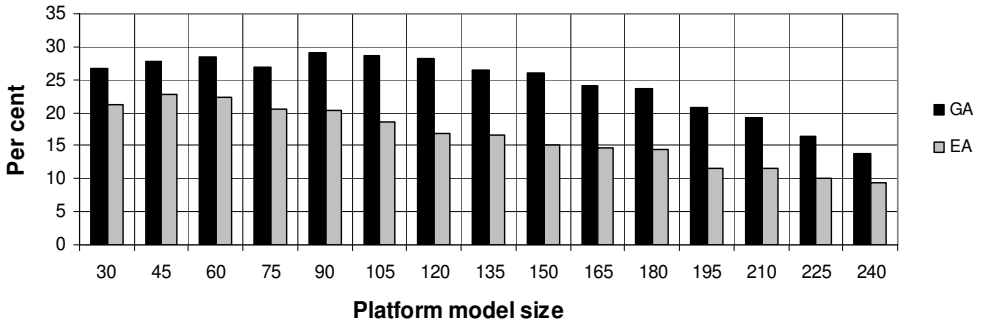

Figure 15. Quality comparison of the algorithms.

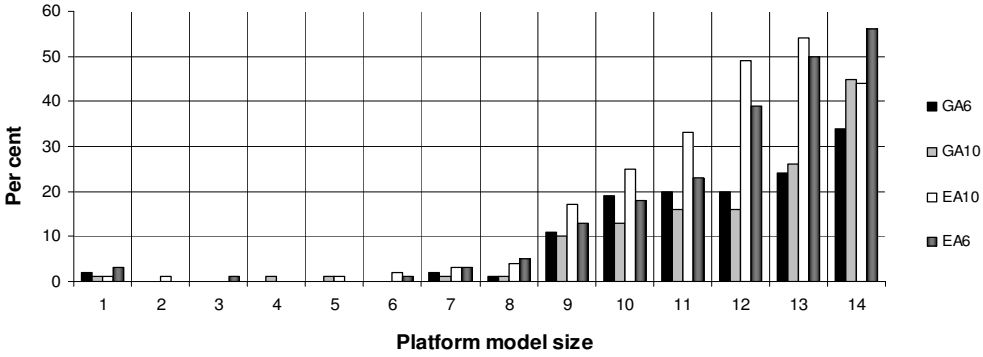

Figure 16. The failure ratios of different algorithms.

As the graphs show (Figure 14.), the genetic algorithm requires longer computational times. This is most likely due to the sorting of the mating pool (i.e., those individuals from the population that will have children). The evolutionary algorithm has a smaller overhead (in some cases over ten times faster than the GA) due to its simplicity. Also, it is possible to see that increasing the number of model properties caused an additional computation overhead with both algorithms.

The quality of the algorithms was estimated according to the improvement of the objective function during the optimization phase. The improvement of the objective function is the difference between the fitness values of the first found feasible individual and the solution after the optimization. Both algorithms were run on the models with 10 properties. As shown in Figure 15, the genetic algorithm always found better solutions than the evolutionary algorithm. This can be explained by the fact that the evolutionary algorithm only uses a mutation operator, which does not allow the exchange of information between the candidate solutions over the course of the optimization. Also, the graphs show that increasing the application and platform sizes leads to lower quality of the solutions found. A probable reason for this is that the algorithms only examine a fixed number of solutions within an expanding search space.

The robustness of the algorithms was evaluated in the last experiment (Figure 16). As was expected, the evolutionary algorithm experienced the highest failure ratios and the genetic

algorithm failed the least. The results also show that the number of failures increases with the model size. This is probably due to the growing number of constraints the algorithms have to satisfy.

7. Discussion and Future work

In this study, we presented an automated system for application composition. The system uses an application allocation algorithm to adapt the applications to changing context and user needs. We suggested two algorithms, the genetic (GA) and the evolutionary algorithm (EA). The goal of these algorithms was to satisfy the constraints imposed by the platform and application models and to optimize the objective function which can also include additional objectives (e.g., those given by the user). Our models are generic, and thus they are not tailored to a certain application type, Therefore, our work can be utilized in different application domains, including pervasive and task-based computing.

The implementation of the application allocation algorithms was based on genetic and evolutionary computing. The algorithms used a novel three-phase evaluation schema and a fitness function. This schema has a number of advantages, for example, it separates the optimization and the constraint satisfaction phases of the algorithms, and it does not require any penalty mechanism to handle feasible and infeasible individuals. The genetic operators were used in the aforementioned schema, which increased their efficiency. We also used a fitness clustering method to interpret the validity of the individuals according to their fitness values.

We analyzed the performance, quality and robustness of our algorithms on a synthesized platform and application models. The experiments demonstrated, that (i) the computational overhead of both algorithms increases when models with a greater number of properties are used; (ii) the GA on average finds 10-15% better solutions than the EA, but at the cost of longer computational times; (iii) the EA produces smaller computation overhead than the GA, but it is more likely to fail when a solution actually exists.

The results show that usage of these algorithms is justified in different scenarios. Thus, we plan to integrate both algorithms into our framework for application composition. This way, the system will decide whether it can tolerate computational overhead or whether it requires accuracy at the expense of longer computational times. For example, the framework may use the GA (which yields high quality solutions) to find an initial application allocation. When adaptation is required, the EA is used to reallocate the initial configuration, because it only causes a small computational overhead even with larger datasets. Affinity constraints can also be used to tag the components which do not require adaptation, thus further increasing the searching speed of the algorithm.

Our future work will focus on designing methods to increase the performance of the application allocation algorithms so that it will be possible to use them, for example, in real-time optimization tasks. In addition, we plan to expand our resource models with a new constraint type which permits partial constraint violations. This approach is promising because the algorithm will be better able to deal with user preferences, such as quality or fidelity requirements.

8. Acknowledgements

This work has been funded by the National Technology Agency of Finland (TEKES) and Academy of Finland. The first author would like to thank Jouni Markkula and Marko Jurmu for their valuable comments regarding the article.

9. References

1. M. Satyanarayanan, “Pervasive computing: vision and challenges,” IEEE Pervasive Communication, Vol. 8, 4, 2001, pp. 10-17.
2. M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R.H. Campbell, and K. Nahrstedt, “A middleware infrastructure for active spaces.”, Pervasive Computing, IEEE, Vol. 1, 4, 2002. pp. 74-83.
3. M. Takemoto, T. Oh-ishi, T. Iwata, Y. Yamato, Y. Tanaka, K. Shinno, S. Tokumoto, and N. Shimamoto, "A service-composition and service-emergence framework for ubiquitous-computing environments," In Proc. of Int. Symposium on Applications and the Internet Workshops, (SAINT 2004) pp. 313-318, 26-30 Jan. 2004.
4. J. Nakazawa, J. Yura and H. Tokuda, "Galaxy: a service shaping approach for addressing the hidden service problem," In Proc. of the Second IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, pp. 35-39, 11-12 May, 2004.
5. M. Handte, K. Herrmann, G. Schiele, C. Becker, “Supporting Pluggable Configuration Algorithms in PCOM” In Proc. of International Workshop on Pervasive Computing and Communications, Fifth Annual IEEE International Conference on Pervasive Computing, pp. 472 – 476, 19-23 March, 2007.
6. S. 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.
7. 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, pp. 328-340, 2006.
8. T. Kichkaylo and V. Karamcheti, “Optimal resource-aware deployment planning for component-based distributed applications,” In Proceedings of 13th IEEE International Symposium on High Performance Distributed Computing, pp. 150 – 159, 2004.
9. T. Kichkaylo et al “Constrained Component Deployment in Wide-Area Networks Using AI Planning Techniques,” In Proceedings of International Parallel and Distributed Computing Symposium (IPDPS’03), Nice, France, 2003.
10. D. de Niz and R. Rajkumar, “Partitioning Bin-Packing Algorithms for Distributed Real-Time Systems,” International Journal of Embedded Systems. Special Issue on Design and Verification of Real-Time Embedded Software, 2005.
11. S. Malek et al, “A Decentralized Redeployment Algorithm for Improving the Availability of Distributed Systems”, In Proc. of 3rd International Working Conference on Component Deployment (CD‘05), Grenoble, France, 2005.
12. M. Chantzara, M. Anagnostou, E. Sykas, "Designing a quality-aware discovery mechanism for acquiring context information," In Proc. of 20th International Conference on Advanced Information Networking and Applications (AINA’06), vol.1, 6 pp.-, 18-20 April 2006.
13. A. Ranganathan and R. H. Campbell, “Pervasive Autonomic Computing Based on Planning”, In Proc. of the IEEE International Conference on Autonomic Computing (ICAC 2004), New York, NY, US, May 17-18, 2004.
14. J.T. Biehl, and B.P. Bailey, “ARIS: an interface for application relocation in an interactive space”, In Proceedings of Graphics interface 2004 (London, Ontario, Canada, May 17 - 19, 2004). ACM International Conference Proceeding Series, vol. 62. Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, Ontario, pp.107-116, 2004.
15. M. Roman, B. Ziebart, and R.H. Campbell, "Dynamic application composition: customizing the behavior of an active space," In Proc. of the First IEEE International Conference on Pervasive Computing and Communications, (PerCom 2003), pp. 169-176, 23-26 March, 2003.

16. B. Johanson, A. Fox, P. Hanrahan and T. Winograd, “The Event Heap: An Enabling Infrastructure for Interactive Workspaces” CS Tech Report CS-2000-02, http://graphics.stanford.edu/papers/eheap/eheap_paper_final.pdf, (Accessed in February, 2008).
17. D. Preuveneers, and Y. Berbers, “Automated context-driven composition of pervasive services to alleviate non-functional concerns”, In Proc. of the ICPS'05 International Workshop on Software Aspects of Context (Kouadri, G.M. and Brezillon, P., eds.), vol 150, CEUR Workshop Proceedings, pp. 1-8, 2005.
18. G. Lee, P. Faratin, S. Bauer, J. Wroclawski, "A user-guided cognitive agent for network service selection in pervasive computing environments," In Proc. of the Second IEEE Annual Conference on Pervasive Computing and Communications, (PerCom’04), pp. 219-228, 14-17 March, 2004.
19. J. Buford, R. Kumar, and G. Perkins, "Composition trust bindings in pervasive computing service composition," In Proc. of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops, (PerCom Workshops 2006), pp. 6 pp.-, 13-17 March, 2006.
20. C. Hesselman, A. Tokmakoff, P. Pawar, S. Iacob, “Discovery and Composition of Services for Context-Aware Systems” In Proc. of the 1st IEEE European Conference on Smart Sensing and Context, Enschede, The Netherlands, October 2006.
21. O. Davidyuk, J. Ceberio, and J. Riekki, An Algorithm for Task-based Application Composition, In Proc. of the 11th IASTED International Conference on Software Engineering and Applications (SEA07), Cambridge, Massachusetts, USA, November 19-21, 2007.
22. M. Perttunen, M. Jurmu and J. Riekki, “A QoS Model for Task-Based Service Composition,” In Proc. of the 4th International Workshop on Managing Ubiquitous Communications and Services (MUCS 2007), Munich, Germany, 25 May, 2007.
23. J. Anke, B. Wolf, G. Hackenbroich, K. Kabitzsch. “A Planning Method for Component Placement in Smart Item Environments using Heuristic Search”. In the Proc. of the 7th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2007), 05-08 June 2007, Paphos, Cyprus.
24. H., van Dijk, K., Langendoen, H., Sips, "ARC: a bottom-up approach to negotiated QoS," In Proc. of the Third IEEE Workshop on Mobile Computing Systems and Applications, pp.128-137, 2000.
25. X. Yang, "Designing traffic profiles for bursty Internet traffic," Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE , vol.3, no., pp. 2149-2154 vol.3, 17-21 Nov. 2002
26. O. Davidyuk, I. Selek, J. Ceberio and J. Riekki, “Application of Micro-Genetic Algorithm for Task Based Computing” In Proceedings of 1st International Conference on Intelligent Pervasive Computing (IPC-07), Jeju Island, Korea, October, 2007, pp. 140-145.
27. G.E. Liepins, and M. D. Vose, “Representational issues in genetic optimization”, Journal of Experimental and Theoretical Artificial Intelligence, vol. 2, pp. 101-11, 1990.
28. F. Rothlauf, “Representations for Genetic and Evolutionary Algorithms”, Springer: Berlin, Heidelberg New York, 2006. ISBN-13 978 3 540 25059 3.
29. T. Jones, and S. Forrest, “Fitness distance correlation as a measure of problem difficulty for genetic algorithms”, In Proc. of the Sixth International Conference on Genetic Algorithms, San Francisco, CA. Morgan Kaufmann. pp. 184–192, 1995.
30. D. Powell, and M.M. Skolnick, “Using Genetic Algorithms in Engineering Design Optimization with Non-Linear Constraints”, In Proc. of the 5th international Conference on Genetic Algorithms S. Forrest, Ed. Morgan Kaufmann Publishers, San Francisco, CA, pp. 424-431, 1993.
31. The Boston University Representative Internet Topology Generator (BRITE) http://www.cs.bu.edu/brite/, (Accessed in February 2008).

Authors

Oleg Davidyuk is a PhD student at the Department of Electrical and Information Engineering in the University of Oulu. His research interests include application of evolutionary algorithms, pervasive computing and middleware. He received a MSc degree in information technology from Lappeenranta University of Technology, Finland, in 2004. Contact him at the Dept. of Electrical and Information Eng., PO Box 4500, FIN-90014, University of Oulu, Finland, oleg.davidyuk@ee.oulu.fi.

István Selek received his MSc degree in applied mechanics and machine design from Budapest University of Technology and Economics, Hungary, in 2003. He is currently working towards his PhD degree at the Systems Engineering Laboratory (SEL) in University of Oulu, Finland. His research focuses on the application and theoretical models of heuristics (mainly genetic algorithms). His current interests are centered on the industrial applications of soft computing methods.

Jon Imanol Duran is writing his Master Thesis at the Department of Electrical and Information Engineering in the University of Oulu. His research interests are pervasive computing, mobile computing and applications. He has been studying computer engineering at the University of the Basque Country. Contact him at the Dept. of Electrical and Information Eng., PO Box 4500, FIN-90014, Univ. of Oulu, Finland, jiduran001@ikasle.ehu.es.

Jukka Riekki is a professor of software architectures for embedded systems at the Department of Electrical and Information Engineering in the University of Oulu. His main research interest is context-aware systems serving people in their everyday environments. He received his Doctor Of Technology degree from the University of Oulu. Contact him at the Dept. of Electrical and Information Engeering, PO Box 4500, FIN-90014, University of Oulu, Finland, jukka.riekki@ee.oulu.fi.