Application of Micro-Genetic Algorithm for Task Based Computing

Oleg Davidyuk, Istvan Selek, Josu Ceberio, Jukka J. R. Riekki

Application of Micro-Genetic Algorithm for Task Based Computing

Oleg Davidyuk[1], István Selek[2], Josu Ceberio[3] and Jukka Riekki[4]

[1,4] Dept. of Electrical and Information Engineering
P.O. Box 4500, University of Oulu, Finland, 90014
Tel: +358-08-553-2544, {firstname.secondname}@ee.oulu.fi

[2] Dept. of Informatics, Budapest University of Technology and Economics
H-1111 Budapest, Műegyetem rkp. 3
Tel: +36-1-463-1266, selek@rit.bme.hu

[3] School of Computer Engineering
University of Basque Country, San Sebastián 20018 Spain
Tel: +34-94-301-8000, jceberio001@ikasle.ehu.es

Abstract— **Pervasive computing calls for applications which are often composed from independent and distributed components using facilities from the environment. This paradigm has evolved into task based computing where the application composition relies on explicit user task descriptions. The composition of applications has to be performed at run-time as the environment is dynamic and heterogeneous due to e.g., mobility of the user. An algorithm that decides on a component set and allocates it onto hosts accordingly to user task preferences and the platform constraints plays a central role in the application composition process. In this paper we will describe an algorithm for task-based application allocation. The algorithm uses micro-genetic approach and is characterized by a very low computational load and good convergence properties. We will compare the performance and the scalability of our algorithm with a straightforward evolutionary algorithm. Besides, we will outline a system for task-based computing where our algorithm is used.**

I. INTRODUCTION

Ubiquitous and pervasive computing are interaction models where computation is integrated into an environment containing many heterogeneous computational devices. This heterogeneity is imposed by a vast diversity of communication protocols, interfaces, and computational platforms among the devices. In addition, resource availability is highly variable as the users are mobile [17, 18, 20].

Recently, this paradigm has evolved into task-based computing [12, 16, 21], where the user explicitly defines tasks which are then realized using services available in the environment. It is desirable that task assembly is supported at run-time as the situation and user needs are changing. That is, different resource combinations can be appropriate to realize even the same task or application. The planning of task assembly is yet one of the main challenges to be achieved by task-based computing. This planning problem has been considered recently by a number of component frameworks [14, 18, 21]. For example, the planning system in the Gaia framework [18] uses an abstract resource model and supports a broad variety of user goals. However, in Gaia the planner does not optimize the application's quality of service (QoS). In the Aura project [21] the application's QoS requirements are specified explicitly by the user and the planning system focuses on user task feasibility maximization, which is the abstract measure of "user happiness". The COCOA middleware [14] focuses on a semantic free service conversation but it does not include a planner for application allocation. Moreover, none of these related works consider dynamic application QoS or support resource management.

The application allocation algorithms are closely related to processor job allocation and load-balancing algorithms such as in [6, 10]. These methods operate on a processor task level and do not capture application requirements and user needs which are critical for optimizing the application's QoS. Our research has also been influenced by a partitioning bin-packing task considered by de Niz and Rajkumar [15]. In this task, a set of software components are packed into a minimum number of bins. However, Niz and Rajkumar address a planning problem of whether every software component can be further partitioned. DecAp allocation algorithms [11] focus on network partitioning problems and assume that every host can only access a part of the network. Our algorithm do not target this issue. Sekitei [7] and modified Sekitei [8] deploy components dynamically to reduce the computational load of the hosts, to satisfy QoS requirements, and to improve throughput. However, both these algorithms only optimize special kinds of applications in which components consume or produce data streams.

To give a concrete example where dynamic assembly and execution of user tasks can take place, we present the following scenario: *"John decides to see a movie and he needs to assemble an eMovie application which consists of three service components: local and remote user interfaces (UIs) and AV playback. 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 service synchronizes the audio and video streams received from the online movie trading service. AV playback can also compress streams to match the capabilities of the end-point rendering device. When John opens the application, the AV playback and the remote UI services are allocated to the available devices. The mobile device uses an application allocation algorithm to find an optimal allocation for these services. After this, the needed resources are allocated; the services are deployed and configured so that John can enjoy watching his movie. If no feasible allocation is found, the application components are configured to run in John's mobile device. When John is watching the movie the assembled eMovie application can be adapted (i.e. reallocated) to another display if John's context changes or the application starts to consume more resources than anticipated."

As demonstrated in the scenario, dynamic application assembly requires an algorithm to allocate the application components to the networking hosts. The algorithm has to optimize the allocation according to a given criterion such as the minimization of application hardware requirements, load-balancing, or the maximization of application QoS.

In this paper, we will introduce a new application allocation algorithm for task-based computing. The new schema is characterized by a very low computational load and good convergence properties. It relies on a micro-genetic algorithm and maximizes the application QoS within a given constraint set imposed by the environment and the user. We compare our algorithm with a straightforward evolutionary computing algorithm that we described in our earlier publication [2].

Besides, we will specify a list of external services to support task-based application assembly and will outline their functionality.

II. SYSTEM FOR TASK-BASED APPLICATION COMPOSITION

Task-based computing systems assume that user tasks are represented explicitly and they take the responsibility of mapping user tasks to the available network resources. Each user may have one or more tasks; however, only one task is active at a time. The user moves from one task to another by changing the currently active task. Each time a task is changed the system has to choose application components constructing the task and then allocate them according to user needs and the task quality attributes. This approach enables the adaptation to changing context and user preferences, but the application allocation problem has to be solved as the tasks have to be constructed dynamically. We will describe the application allocation problem already in the section 3.

The overview of the system is shown in Figure 1. [16]. Application Assembly is the central component in the system because it controls the application composition and decides on the allocation of applications accordingly to task properties and user needs.
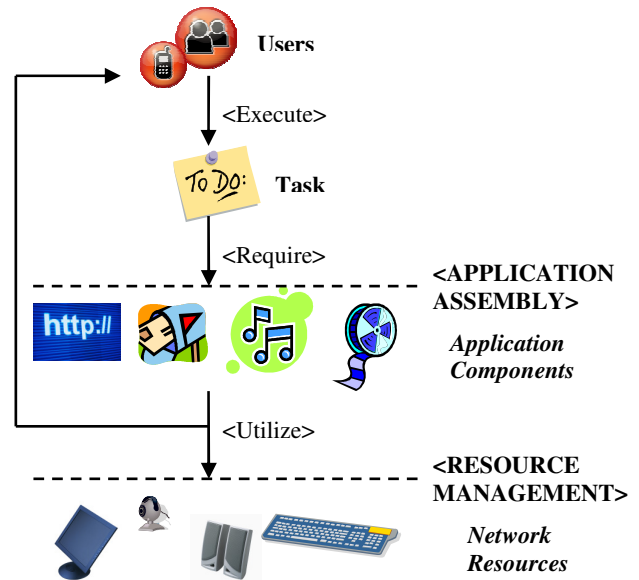

Fig. 1.   System overview.

The Application Assembly controls the entire task lifecycle from the activation moment to the completion. After the user has activated a task, the Application Assembly requests information about the environment from the Service Discovery component. The Service Discovery keeps a list of the application components, hosts and their properties and performs matchmaking. For example, Application Assembly can request The Service Discovery to find hosts in a specific area or find the application components required for the current task.

The Resource Management component monitors environment resources and controls resource access via leasing. It optimises the resource usage in presence of multiple users and simultaneous access requests, resolves access conflicts and handles validity of resource usage [5].

After the Application Assembly has performed an application allocation, it leases the required resources from the Resource Management. If the resources are currently not available, the Application Assembly may try to negotiate lower task QoS preferences and to reallocate the application once again.

III. TASK-BASED APPLICATION ALLOCATION PROBLEM

User tasks. Tasks are abstract descriptions that include user preferences and descriptions of the application components needed for the task. An application component is a software object which implements a specific functionality accessed via an interface. The application's component descriptions specify the validity constraints (e.g., computational, bandwidth, security, etc) which have to be fulfilled before the component is allocated. Optionally, user or the Application Assembly can tag an application component with an affinity constraint to restrict the component's allocation to a certain host. Affinity constrains are also needed if a certain state of the task requires access to specific material (e.g., a document or a user profile).
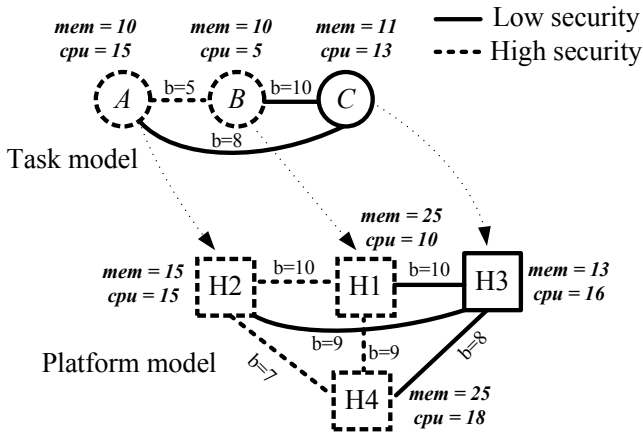
Fig. 2.   An example application allocation.

Tasks can be modelled as graphs in which nodes represent application components and links specify communication channels between the application components.

Computational hosts. Hosts are networking devices executing application components. Each host can allocate one or more application components if the resource constraints of the host are not violated. These constraints are, for example, maximum computation, memory and bandwidth resource capacities. The hosts can be presented as a platform model, which is a graph describing network topology.

The properties of both the platform and the task graphs are expressed by real values (e.g., bandwidth and memory capacities) and integer intervals (e.g., security levels).

The goal is to optimize the task graph structure (i.e. groups of application components and their connections) and assign groups according to the feasibility constraints and the task QoS requirements.

The solutions to the application allocation problem are evaluated using the objective function that has 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 by the allocation to the sum of the bandwidths required by the application component links. This value decreases when some of the components are allocated to the same host and hence the communication between the components does not require a network link.
- 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 affects the time needed for the actual deployment, and
- f_V is a standard statistical measurement of variance in processing capacity usage in the hosts, that is, the variance of free capacity in the hosts after the allocation of the components. Lower variance is better

as it balances the load so that the utilization of each host is within a desired range.

The application allocation problem was proved to be NP-complete in [7]. Further details of the application allocation problem can be found in [2].

An example of an application allocation, consisting of 3 application components and 4 hosts, is illustrated in Figure 2. Resource consumptions and capacities, such as memory ("mem") and CPU are shown next to the nodes. Bandwidth consumptions are depicted next to the links.

IV.   THE APPLICATION ALLOCATION ALGORITHM

A.   Straightforward Evolutionary Computing Algorithm

We present a straightforward evolutionary algorithm (SEA) for the application allocation problems which produces near-optimal solutions as seen in [2]. It uses a tournament selection mechanism, one-point crossover and a multipoint mutation operator. The constraints are enforced with a penalty function (Eq. 2). The population diversity in the SEA depends only on the mutation operator after the population is initialized. But, the mutation rate has to be set very low in order to allow the offspring to inherit characteristics from their parents. Hence, the SEA has to operate with a large population to provide a sufficient sample size and to avoid premature convergence which leads to long computational time.

B.   Micro-Genetic Allocation Algorithm

To address the performance drawbacks of the SEA we introduce a micro-genetic algorithm (MGA) [1, 9] that is characterized by a very low computational load. The algorithm uses an external memory and internal population with reinitialization. The internal population size is less than 10 individuals. An external memory is used as a source of the population diversity and to store the best individuals found earlier. The MGA works on the internal population until it reaches nominal convergence [4]. That is until the internal population contains the individuals with either identical or very similar genotype.

The candidate solutions (i.e. individuals) have a direct representation [19]. An example representation of an individual containing 6 application components and 3 hosts is shown in Figure 3. As the figure shows, the length of an individual is equal to the total number of application components in the task description. Thus, the number in the ith gene position denotes the host identity (id) which allocates the ith application component.
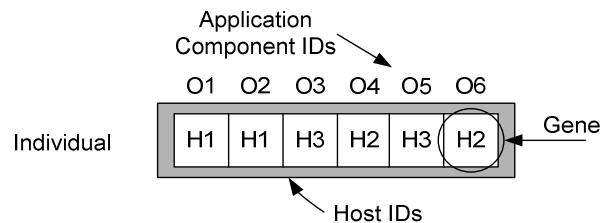

Fig. 3.   The representation of a candidate solution.

The flowchart of the algorithm is presented in Figure 5. During the initialization phase, the algorithm randomly generates a population which is used to fill the external memory. Then, the individuals are evaluated: the infeasible individuals (violating the constraints) are penalized and the fitness values are calculated for the feasible individuals in the population. The penalty function is defined as follows:

$$P = P_C + P_M + P_B + P_{SS} + P_{SL} \qquad (2)$$

where

- P_C and P_M specify a portion of memory and the computational resource capacities which are violated by the individual,
- P_B specifies the part of the bandwidth resource capacity which is violated by the individual,
- P_{SS} and P_{SL} define the number of hosts and network links that do not meet the security constraints.

At the beginning of each micro-GA cycle the algorithm picks half of the internal population with a certain probability from the external memory. Then, it randomly generates the second half to increase genetic polymorphism.

The algorithm uses standard genetic operators such as binary tournament selection, crossover, mutation and elitism at each iteration of the micro-GA cycle (see Figure 4). The elitism operator saves the individuals with the highest fitness in the internal memory regardless of individuals with higher fitness values existing in the external memory.

In the end of each micro-GA cycle the memory handler compares two of the best individuals from the current population with two of the worst individuals in the external memory. If the latter have smaller fitness values than the former ones, the memory handler replaces these individuals in the external memory with the individuals from the population.

The MGA handles individuals accordingly to their feasibility as follows:

Infeasible individuals. The fitness values of the infeasible individuals are set to the total number of the violations multiplied by negative one. Hence, the algorithm does not use the objective function to calculate fitness values. The crossover operator applies a standard one-point and uniform crossover schemas [3] both of which have 50% probability to be used.
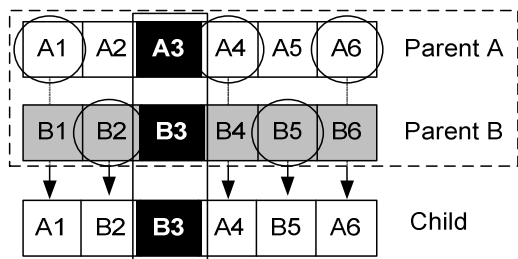

Fig. 4. The uniform crossover schema for the feasible individuals. The numbers denote gene positions.
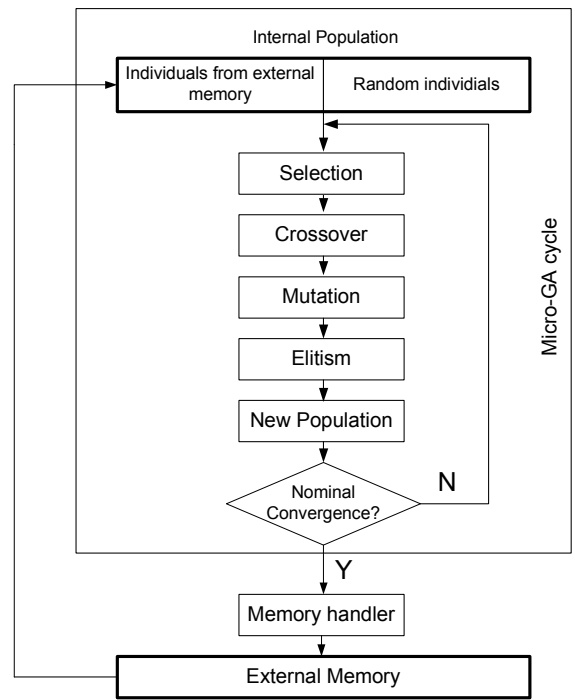

Fig. 5. The flowchart of micro-genetic algorithm.

The first crossover schema enforces faster convergence, and the second guarantees higher information exchange rate between the parents when the population reaches nominal convergence.

Feasible individuals. The fitness values of the feasible individuals are calculates as follows:

$$fitness = \frac{1}{F_{obj}} \qquad (3)$$

The crossover operator chooses from two uniform crossover schemes with 50% probability. The second crossover schema for the feasible individuals performs as shown in Figure 4. It assigns child genes as the following: the initial parent is randomly chosen and its first gene is copied to the child's first gene. The second gene is taken from the second parent. If both parents have the same gene value at the same gene position (these genes are marked as black boxes in the picture), this gene is copied to the child and the process starts once again from the next position. The crossover stops when the child has all the genes filled.

Besides general generic parameters, such as tournament size, mutation and crossover rates, the algorithm has three additional parameters: micro population size, external memory size, and micro-cycle size. The micro population size denotes the size of the internal population. The external memory size implies the size of the whole population. The micro-cycle size affects nominal convergence [4]. The algorithm converges slower if the external memory size is increased.
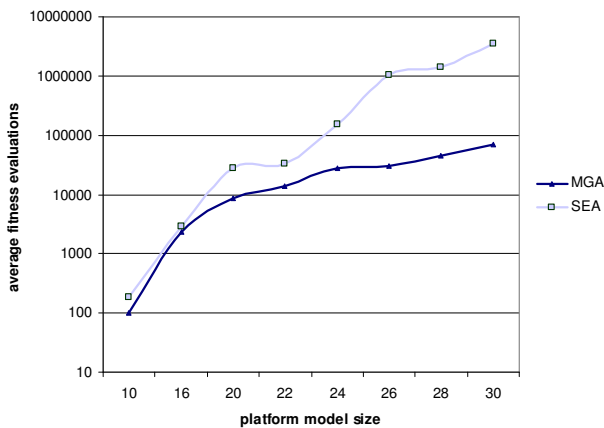
Fig. 6.   Average number of fitness evaluations while increasing graph model sizes.

Fig. 7.   Quality of solutions produced by MGA and SEA. Smaller values denote better quality.

V. EXPERIMENTS

We have implemented a micro-genetic algorithm in Java and evaluated its performance in comparison to the SEA. The implementation and the evaluation results of the SEA are given in [2]. The aim of these experiments was to compare the performance, the quality of the yielded solutions and the robustness of the algorithms. We tested the algorithms on the graph models where the platform graph was always twice bigger than the task graph. We used an average number of fitness evaluations as a metric of algorithm performance.

The algorithms were run until they find a first valid solution. In case of a failure, when one of the algorithms did not find a valid solution, the graphs models were synthesized and the algorithms were run once again. Figure 6 demonstrates how scalability is affected by the number of hosts in the platform and size of the tasks. The resulting curves (graphed on a logarithmic scale) show that MGA drastically outperformed SEA because the MGA only needs a few fitness evaluations due to its population reshuffling and memorization mechanisms. The parameter values of the MGA used in the experiments are shown in the Table 1.

TABLE 1. MGA PARAMETERS.

Crossover probability	1
Mutation probability	0.3
Tournament size	2
Micro-cycle size	2
Micro population size	6
External memory size	100

The quality of the produced solutions was evaluated accordingly to the values of the objective function used by the algorithms (see Eq. 1.) The algorithms were run until they reached 30 000 fitness evaluations without average population fitness improvement. As expected, MGA yielded solutions with the same or better fitness values than SEA (presented in Figure 7). The oscillation in the quality of solutions for different graph pairs most probably results from the randomness of the synthesized graph models.
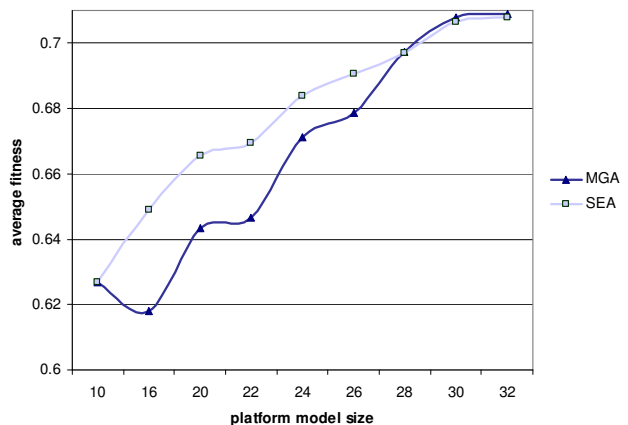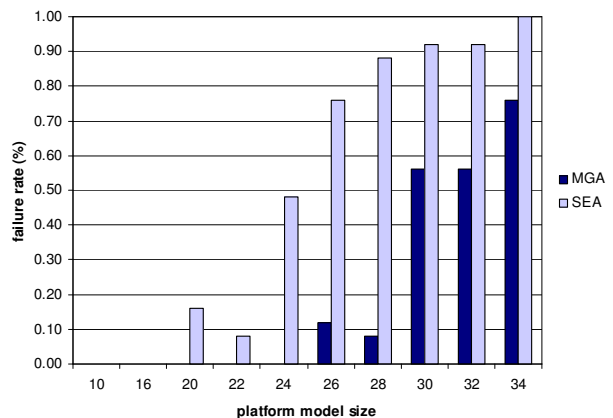

Fig. 8.   MGA and SEA failure diagram.

We compared robustness of the algorithms in the third experiment (Figure 8). We set an additional limit of 200 000 fitness evaluations to make algorithm failure more likely. The algorithms iterated until they found a valid solution or until they reached the fitness evaluation limit. We observed that SEA failed in more cases than MGA. However, the failure ratios may encounter more false positives, meaning that a solution exists, but the algorithms cannot find it within a defined number of fitness evaluations. We observed that failure ratios increase as the platform size increases. This may due to the fact that only a part of the expanding search space was explored.

VI. DISCUSSION AND FUTURE WORK

The task based computing paradigm supports usage of applications composed of multiple distributed components allocated in the network. Our goal is to provide dynamic user task composition to facilitate the adaptation to changing context and user needs. However, this requires a scalable way of solving application allocation problems that are known to be NP-complete. An allocation algorithm plays a key role in the composition of the tasks: it maximizes user task QoS, balances the load among the hosts and meets the platform constraints. The initial design of the allocation algorithm was

based on a straightforward evolutionary algorithm that was presented in [2]. However, SEA suffers from slow convergence and requires the usage of large population sizes which has a negative effect on scalability and performance. In this paper, we presented a micro-genetic algorithm for application allocation which addresses the aforementioned drawbacks. We implemented the algorithm and evaluated it against SEA. The experiments on synthesized models demonstrate that MGA is more scalable, robust and outperforms SEA but at the expense of greater implementation complexity. In addition, the results show that MGA decreases computational time without affecting the quality of solutions. This is achieved by introducing a reshuffling (micro cycle) procedure that randomly generates new individuals when convergence happens. Therefore, MGA only needs a small population size and therefore it does fewer fitness evaluations than SEA.

We are planning to integrate MGA as a part of a system for task based computing applications. However, the current resource model only considers a few resource types, such as memory and bandwidth. Therefore, we will design a QoS model for the application allocation problem, which supports generic resource types and considers proximity as suggested in [16] and takes cost of reconfiguration [21] and other factors relevant to user context into account.

VII. REFERENCES

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

[2] O. Davidyuk, J. Riekki and J. Ceberio, "An application allocation algorithm for pervasive environments," *In Proc. IADIS International Conference on Intelligent Systems and Agents (ISA'07)*, Lisbon, Portugal, 3-5 July 2007.

[3] A. Eiben and J. Smith. **"**Introduction to Evolutionary Computing," *Springer-Verlag*. ISBN 3 540 40184 9, 2003.

[4] D. Goldberg, "Sizing Populations for Serial and Parallel Genetic Algorithms," In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, California, Morgan Kaufmann Publishers, pp 70-79, 1989.

[5] M. Jurmu et al, "Lease-based resource management in smart spaces," *In Proc. Workshops of the 5th Annual IEEE International Conference on Pervasive Computing and Communications*, WiP Session, White Plains, NY, pp. 622-625, 2007.

[6] A. Kejariwal and A. Nicolau, "An Efficient Load Balancing Scheme for Grid-based High Performance Scientific Computing," *In Proceedings of the 4th International Symposium on Parallel and Distributed Computing (ISPDC'05)*, CA, USA, pp. 217- 225, 2005.

[7] 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.

[8] 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.

[9] K. Krishnakumar, "Micro-genetic algorithms for stationary and non-stationary function optimization," In *SPIE Proceedings: Intelligent Control and Adaptive Systems*, pp 289-296, 1989.

[10] Y. Li and Z. Lan, "A novel workload migration scheme for heterogeneous distributed computing," *The IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05)*, pp. 1055 – 1062, 2005.

[11] S. Malek et al, "A Decentralized Redeployment Algorithm for Improving the Availability of Distributed Systems" *In Proceedings of 3rd International Working Conference on Component Deployment (CD'05)*, Grenoble, France, 2005.

[12] R. Masuoka et al, "Task Computing – The Semantic Web Meets Pervasive Computing," *Proc. of 2nd International Semantic Web Conference (ISWC2003)*, Sanibel Island, Florida, USA, pp. 866-881, 2003.

[13] Z. Michalewicz and D. Fogel, "How to solve it: modern heuristics," *Berlin: Springer*, 2000.

[14] S. Mokhtar et al, "COCOA: COnversation-based Service COmposition in PervAsive Computing Environments," *In Proceedings of the IEEE International Conference on Pervasive Services (ICPS'06)*. Lyon, 2006.

[15] 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.

[16] M. Perttunen, M. Jurmu and J. Riekki, "A QoS Model for Task-Based Service Composition," *4th International Workshop on Managing Ubiquitous Communications and Services (MUCS 2007)*, Munich, Germany, 25 May, 2007.

[17] A. Ranganathan and R. Campbell, "Autonomic Pervasive Computing Based on Planning," *Proceedings of the First International Conference on Autonomic Computing (ICAC'04)*, New York, NY, USA, pp. 80-87, 2004.

[18] M. Roman et al, "A middleware infrastructure for active spaces," *Pervasive Computing, IEEE*. Vol. 1, 4, pp. 74-83, 2004.

[19] F. Rothlauf, "Representations for Genetic and Evolutionary Algorithms," *Springer-Verlag Berlin Heidelberg*, ISBN-13 978 3 540 25059 3, 2006.

[20] M. Satyanarayanan, "Pervasive computing: vision and challenges," *IEEE Pervasive Communication*, Vol. 8, 4, pp. 10-17, 2001.

[21] 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.