Genetic Algorithm-Based Optimization of Service Composition and Deployment

Yves Vanrompay Peter Rigole Yolande Berbers

{Yves.Vanrompay, Peter.Rigole, Yolande.Berbers}@cs.kuleuven.be
Department of Computer Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A, 3001 Heverlee
Belgium

ABSTRACT

Services running on mobile systems must be able to adapt themselves to changing user needs and availability of resources. We propose to use Genetic Algorithms to search for the best service variant in the current context. The chosen service composition is then deployed on a set of available nodes in an optimal way. We illustrate that Genetic Algorithms provide a scalable and self-organizing solution to service composition and deployment. We argue that the approach meets some main requirements demanded by services running on mobile systems. A motivating scenario is presented in which a distributed server allows users to share content and run applications in mobile ad-hoc networks.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; I.2.8 [**Computing Methodologies**]: Artificial Intelligence—*Problem Solving, Control Methods, and Search*

General Terms

Algorithms, Performance

Keywords

Service Composition, Service Deployment, Genetic Algorithms

1. INTRODUCTION

Mobile systems that can adapt themselves to changing user needs and availability of resources must be aware of their context by sensing and reacting to context information. An architecture-centric approach can be followed where architecture models of the service are represented at runtime to allow a middleware to reason about and control adaptation [5]. In the event of a context change, the middleware selects the most appropriate service composition that matches the user needs and current available resources. The scalability of this approach is a challenge because the evaluation of all possible service compositions can lead to a combinatorial explosion of solutions. In large service compositions, an exhaustive search could be very time consuming. There exist several heuristics that provide feasible and near optimal solutions in acceptable time. We propose the use of Genetic Algorithms (GAs) to efficiently search for a near optimal service composition. GAs scale well and are suitable to handle generic QoS attributes. We envision a system consisting of several nodes on which a composite service can be deployed in a distributed manner. The goal is to deploy the composite service onto a set of connected nodes in a way that the allocation meets the given Quality of Service (QoS) constraints and minimizes the communication cost between the nodes. GAs are also used to solve this mapping problem.

This paper is organized as follows. The next section gives an overview of the requirements we derived for adaptive mobile systems. Also, a real-life scenario motivating the need for resource-aware distributed service composition and deployment is presented. Then, this approach is compared to related work in Section 3. We describe the application of GAs to component composition and their deployment in section 4. An evaluation with respect to the given requirements and scenario is presented in Section 5. Finally, we draw some conclusions and elaborate on ongoing and future work.

2. REQUIREMENTS AND SCENARIO

The focus of our approach is the development of a middleware system that supports the development and deployment of context-aware adaptive services and applications. With the emphasis on adaptation and deployment, we have identified the following requirements:

- Artificial Intelligence (AI) learning mechanisms must allow the system to self-organize its service deployment configuration in a valid and efficient manner. [R01]
- The planning process should develop ways of dealing with the scalability of the number of variants that can be involved in an adaptation. [R02]
- Service deployment should take into account the limited resources on targeted mobile devices. [R03]
- Service deployment should consider limited or costly bandwidth situations and should support mobile agent mechanisms to offload costly tasks. [R04]

SIPE'08, July 7, 2008, Sorrento, Italy.

- The planning process should distribute the adaptation reasoning among the available nodes to address scalability of the number of nodes, users and networks that can be involved in an adaptation. [R05]
- Models used at run-time should include QoS properties of both software and hardware components. [R06]

In the following, we describe a scenario showing how mobile, context-aware adaptive applications need distributed service deployment. The scenario also motivates the requirements we discussed above.
We envision a distributed platform for sharing content, gaming, chat, etc. formed by devices joining a federation in an ad-hoc manner. Users perceive an *InstantSocial* cloud as an ordinary site offering services. However, these services are scattered across nearby devices and integrated by a distributed application server running on top of our middleware. The server consists of a dynamic composition of services like an ontology service, a video service, a presentation layer and so on. This composition is distributed and replicated which means the distributed server is a combination of different implementations on nearby devices forming an ad-hoc network. Users get notified of topics they may be interested in or multiplayer games that can be joined. Our approach can be used to adapt and configure the ad-hoc server infrastructure needed for *InstantSocial*. Services can be deployed, redeployed or migrated across the different available devices taking into account the resource availability of each individual device. Users can also specify a maximum amount of resources they want to reserve for the distributed services running on their device. While devices come and leave and the ad-hoc network evolves, services can be dynamically redeployed to ensure the QoS for the users, balance the use of resources and minimize user distraction. Whenever fixed infrastructure is available, this can be used as a backup and for deployment of services that use a lot of resources. A way of using *InstantSocial* is on metro trips where instant sites could appear out of nothing just by travelers having their devices turned on. Other usages include conferences, concerts, festivals and sports events.

3. RELATED WORK

Given that there may exist more than one web service that provides identical functionality but different QoS, Canfora et al. [2] highlight the need to determine which services participate in a required composite service (web application). A Genetic Algorithm approach to the QoS-aware service composition problem is proposed. When compared to widely-used linear integer programming methods, the proposed genetic algorithm deals with non-linear functions while it scales well with an increase in the number of tasks. A randomized heuristic based on general principles of genetic search strategy was described by Chockalingam et al. [4] to solve the mapping problem, in which parallel tasks are assigned to a multiprocessor to minimize the execution time. Each candidate solution in the population is an integer string where the ordinal value of any element of the string represents a task identity while its cardinal value represents a processor identity to which the process has been mapped. The work done by Cao et al. [3] is a recent development on the use of GA to the service selection problem in the context of web service composition. A service selection model defines a business process as a composition of many service agents. Each service agent corresponds to a set of multiple web services that perform a specific task, and are provided by different service providers. Furthermore, a service composition agent is employed and is responsible for selecting an appropriate service component for all service agents such that the resulting business process is the optimized solution satisfying the specified Quality of Service criteria. Despite this web service selection model, a GA-based service composition module is used to formulate the selection problem based on the nature of an application and an encoding mechanism to accommodate the genetic algorithm in the context of the selection problem. Research presented by Zhang et al. [8] elaborates on a specific GA to achieve QoS-aware web services selection, but does not treat distributed service deployment. Likewise Gao et al. [6] have developed a tree-coded GA to support service composition. Project Cubik [7] focuses on the distribution and deployment of software components in dynamic pervasive networks. While based on the Fractal component model, the deployment of components takes into account the heterogeneity of the hosts and of the links between them. It also copes with the possibility of disconnections and fragmentation of the network. Constraints put on host resources and instances co-location can be specified.

4. APPLICATION OF GA TO SERVICE COMPOSITION AND DEPLOYMENT

This section elaborates on the use of GAs for service composition and deployment. First we explain how genetic algorithms work. Then, the mechanism to achieve service variability is presented, after which we go into detail on how to compose and deploy services with GAs.

4.1 Genetic algorithms

The use of Genetic Algorithms is a popular search method in the area of multi-objective optimization. Having its roots in natural genetic systems, the basic principle of survival of the fittest is fundamental to the idea of GA. To achieve this, GAs work with a set of genomes called a population. A genome in this setting is a candidate solution that evolves over several generations (iterations). In its basic form, a genome is a string of binary values and each genome is associated to a fitness value based on a fitness function that indicates how good it is when compared to other solutions in the population. The fitness value of an individual solution is also an indication of its chances of survival and reproduction in the next generation. During each generation of a GA, genomes in the population undergo a standard set of operations (selection, crossover, mutation, and evaluation). These operations are central to GAs since they exploit the historical information of the current population to evolve into the next generation with some possible improvement. Beginning with a selection operation, members of the population are selected in pairs where each pair is considered for a crossover operation that generates new genomes (also called offspring) using a randomly generated cut-off point and swapping of binary values between the original pair of genomes. Further on, the offspring is placed back into the population replacing the weaker ones. After the crossover, each genome in the population is mutated with some probability that may change a genome into a new one. Finally, an evaluation operation is performed to update the fitness

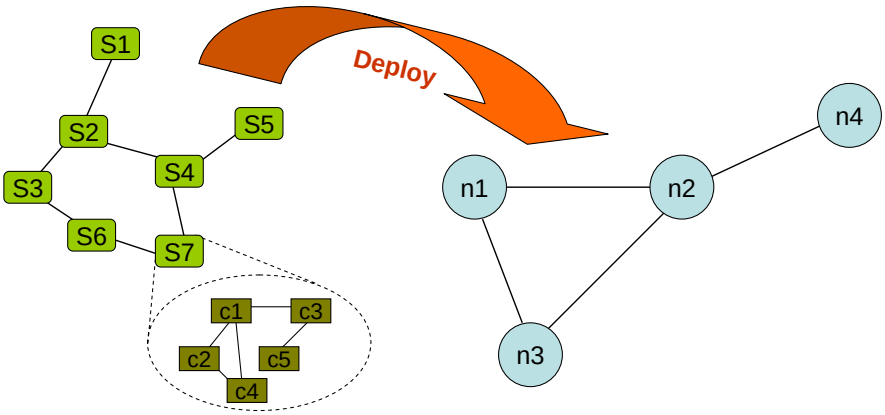

Figure 1: Service composition and deployment

values of the genomes of the population. This process is repeated until the population meets the termination criteria defined by the user.

4.2 Service variability

To develop adaptive mobile systems, an architectural approach can be followed, in which an architecture model describes the variability of the service. We assume a service is built using component types. These component types are variation points in the architecture that specify alternative or optional building blocks for the service, i.e. alternative implementations of a component type. At runtime, a middleware can automatically derive a service variant that is optimal in the given context. Variability is thus achieved by having different component implementations whose behavior conforms to the type. Alternative component implementations are annotated with properties that specify the services offered and needed by a component. System resources, such as memory and bandwidth that are needed by the component implementation can be specified. Property predictor functions predict the properties of a component implementation in a given context. When a context change occurs and adaptation is necessary, the variant must be selected that best fits the current context. Utility functions are used that assign a scalar value to service variants as a function of the properties of the service in the given context.
Using GAs, we would like to select a suitable service composition and deploy that composition onto connected nodes. The solution to this problem consists of two steps. First, we decide which concrete services can be allocated on each node, given the abstract services of which the composition is built. Second, concrete services are effectively distributed onto the nodes, taking into account communication costs and the feasibility of the deployment. Referring to figure 1, the set of abstract services S1 to S7 is called the composite service. Each abstract service is associated with a number of concrete services. These concrete services are implemented as component compositions. Figure 1 shows one possible component composition for S7, consisting of component implementations C1 to C5. Our goal is to deploy a near-optimal service composition on the set of nodes N1 to N4.

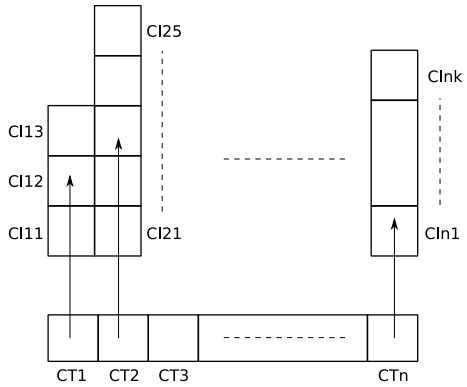

Figure 2: Genome encoding of service composition

4.3 Service Composition

First, a service selection step is performed at each node. The composite service consists of a set of abstract services and each abstract service is associated with a number of concrete services. These concrete services are alternative component compositions implementing the abstract service. On each node j, the optimal composition for each abstract service is calculated. The fitness of the genome $F(g_{ij})$ is equal to $F(S_{ij})$, the fitness of the concrete service for abstract service i on node j.

$$F(g_{ij}) = F(S_{ij}) \tag{1}$$

In this step, the communication cost is not taken into account. The solution needs to be encoded with a suitable genome. We present the genome by an integer array with the number of elements equal to the total number of Component Types (CT) that compose the service (Figure 2). Each element contains an index to the array of alternative Component Implementations (CI) that exist for the given component type.

Parents are selected using a linear search through a roulette wheel with slots weighted in proportion to genome fitness values. The crossover operator is the standard two-point crossover, and the mutation operator randomly selects a

component type and replaces the corresponding component implementation with another one. Component types for which only one component implementation is available are not considered for mutation. The problem can be modelled using a fitness function and constraints:

$$F(g_{ij}) = Q(g_{ij}) - D(g_{ij}) \quad (2)$$

This fitness function has to be maximized. $Q(g_{ij})$ is a function that contains QoS attribute factors such as availability, reliability, response time and cost of the service variant represented by the genome on node j. $Q(g_{ij})$ can be implemented by the following function:

$$Q(g_{ij}) = \frac{(w_1.Availability + w_2.Reliability)}{(w_3.Cost + w_4.ResponseTime)} \quad (3)$$

The weights of this function can be tuned according to user needs or relative importance of the service compared to other services and applications. QoS attributes such as response time can depend on the node on which the service is deployed. Property predictor functions that predict the (QoS) properties of a service in a given context are used to calculate the value of these QoS attributes. The fitness function $F(g_{ij})$ can be seen as a specific utility function for a service (see section 4.2).

$D(g_{ij})$ is used to penalize individuals that do not meet the given constraints. We suppose that a set of k constraints is defined as follows:

$$cl_k(g_{ij}) \leq 0, \quad k = 1, ...n \quad (4)$$

$D(g_{ij})$ expresses hard constraints for the service variant to be able to run on node j. A given service variant needs for example a certain amount of memory and processing power available on the device. An example constraint expression is the following:

$$cl_1(g_{ij}) = w_1.\frac{(Memory(g_{ij}) - Memory(N_j))}{(Memory(g_{ij}) + Memory(N_j))} \quad (5)$$

This expression is positive if the memory needs for the service represented by genome g_{ij} can not be satisfied by node N_j. The expression is normalized by the division such that its value is between -1 and 1, and w_1 weights its importance relative to the other constraints in $D(g_{ij})$. Then the distance from constraint satisfaction is:

$$D(g_{ij}) = \sum_k cl_k(g_{ij}).y_k \quad (6)$$

where:

$$y_k = 0 \;\; if \;\; cl_k(g_{ij}) \leq 0$$

$$y_k = 1 \;\; if \;\; cl_k(g_{ij}) > 0$$

4.4 Service Deployment

Given the optimal fitness of each service on each node, the nodes on which to allocate the services can be chosen. Further on the communication costs, such as remote deployment overhead and available bandwidth, are considered. Also, the distance to a feasible composite deployment is taken into account.

The mapping of services to nodes is encoded as an integer string (Figure 3). The ordinal value of an integer in the string represents the identity of the abstract service and the cardinal value is the node on which the abstract service is deployed as a concrete service. Parent selection, crossover and mutation are as in the service composition step.

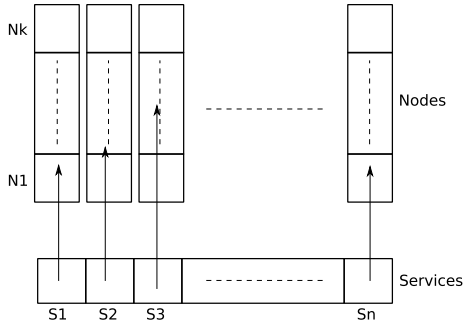

Figure 3: Genome encoding of service deployment

The fitness function for the allocation is as follows:

$$F(G) = \sum_i (F(S_{ij})) - C(G) - D'(G) \quad (7)$$

This function has to be maximized. $F(S_{ij})$ is the fitness of the optimal concrete service for abstract service i on node j, which was calculated in the first step. $C(G)$ is the cost of communication overhead. To minimize $C(G)$, services that communicate intensively with each other should be placed on the same node (if feasible), or on nearby nodes in a mobile ad-hoc network. Communication cost will also be higher on a device that has to use GRPS than on a device that can use WiFi.

$D'(G)$ is the distance from a feasible solution. This distance function represents a penalty for constraints that are not met, like in the service composition step. Since several services can be deployed on the same node, the following assumption has to be made for the approach to be sound:

$$F(S_{ln}) + F(S_{kn}) = F(S_{ln}, S_{kn}) \; if \; (S_{ln}, S_{kn}) \; is \; feasible. \quad (8)$$

(S_{ln}, S_{kn}) is the composite deployment of S_l and S_k on host n and $F(S_{ln}, S_{kn})$ is the composite fitness of (S_{ln}, S_{kn}).

5. EVALUATION

As this paper discusses some early results of our research, this section investigates whether and how the proposed approach satisfies the requirements and scenario needs that were identified in Section 2. By using GA, the system can self-organize its service composition and deployment configuration in an efficient and near optimal way [R01]. Moreover it scales well with the number of nodes on which to deploy the services. Scalability is also achieved with respect to the number of variants that have to be considered before adaptation [R02]. The set of constraints defined in the service composition step allows to take into account limited resources on mobile devices [R03]. Communication costs are minimized by incorporating cost of communication in the fitness function of the service deployment step [R04]. The adaptation reasoning is inherently distributed among the several nodes and makes the delay noticed by the user during reconfigurations minimal [R05]. At each moment in time, the search for a solution can be stopped, and the current best solution can be selected. The fitness function includes QoS properties and hardware constraints for the services [R06].

The *InstantSocial* scenario clearly shows the need for distributed service deployment in a real-life application. Services that are distributed in an evolving mobile ad-hoc network consisting of resource-constrained hosts must be composed and deployed in a self-organizing way. GAs provide a mechanism to orchestrate efficient and optimal deployment (re)configuration.

6. CONCLUSIONS

We have presented an approach to compose and deploy services in a distributed way. GAs provide a scalable mechanism which offers improvements over relevant solutions. We have shown that the approach meets the identified requirements and motivated this with an example application. Concerning future plans, the mechanism will be implemented and validation and testing will be done.

7. ACKNOWLEDGEMENTS

The authors of this paper would like to thank their partners in the MUSIC-IST [1] project and acknowledge the financial support given to this research by the European Union (6th Framework Programme, contract number 35166).

8. REFERENCES

[1] The MUSIC consortium, self-adapting applications for mobile users in ubiquitous computing environments, http://www.ist-music.eu.

[2] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. An approach for qos-aware service composition based on genetic algorithms. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1069–1075, New York, NY, USA, 2005. ACM.

[3] L. Cao, M. Li, and J. Cao. Using genetic algorithm to implement cost-driven web service selection. *Multiagent and Grid Systems*, 3(1), 2007.

[4] T. Chockalingam and S. Arunkumar. Genetic algorithm based heuristics for the mapping problem. *Comput. Oper. Res.*, 22(1):55–64, 1995.

[5] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjorven. Using architecture models for runtime adaptability. *Software, IEEE*, 23(2):62–70, March-April 2006.

[6] Chunming Gao, Meiling Cai, and Huowang Chen. Qos-aware service composition based on tree-coded genetic algorithm. *Computer Software and Applications Conference, 2007. COMPSAC 2007 - Vol. 1. 31st Annual International*, 1:361–367, 24-27 July 2007.

[7] Didier Hoareau and Yves Mahéo. Middleware support for the deployment of ubiquitous software components. *Personal Ubiquitous Comput.*, 12(2):167–178, 2008.

[8] Chengwen Zhang, Sen Su, and Junliang Chen. A novel genetic algorithm for qos-aware web services selection. *LNCS 4055*, pages 224–235, 2006.