Adaptable Model-based Component Deployment Guided by Artificial Ants

Máté J. Csorba, Poul E. Heegaard, and Peter Herrmann
Norwegian University of Science and Technology (NTNU)
Department of Telematics
N-7491 Trondheim, Norway
{csorba, poulh, herrmann}@item.ntnu.no

ABSTRACT

We investigate a means for efficient deployment of distributed services comprising of software components. Our work can be viewed as an intersection between model-based service development and novel network management architectures. In a service engineering context, models of services embellished with non-functional requirements are used as input to our swarm intelligence based deployment logic. Mappings between resources provided by the execution environment and components are the results of our heuristic optimization procedure that takes into account requirements of the services. Deployment mappings will be used as feedback towards the designer and the provider of the service. Moreover, our heuristic algorithm possesses significant potential in adaptation of services to changes in the environment.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Management

Keywords

component deployment, cross-entropy ant system, QoS requirements

1. INTRODUCTION

In the process of realizing a service system several important decisions have to be made that will affect performance of the system as well as the quality of service (QoS) perceived by its user. A state-of-the-art method to develop distributed services implemented as software systems is starting from a platform independent model and realizing the service following a top-down step-wise refinement approach. A significant factor influencing the perceived QoS from the user's perspective is the deployment model of the particular service, in other words the configuration of the building-blocks of the service and their mapping to run-time processing elements and resources required for execution.

Nodes hosting a service may consist of heterogeneous hardware and may provide a dynamic environment for the services being executed, i.e. nodes can join and leave the network in an unpredictable manner. The evolving nature of the context of distributed services and mobility of clients requires the capability of adaptation to satisfy QoS requirements while also considering costs on the service provider's side. The wide range of possible requirements against the service makes the deployment and adaptation problem a multi-faceted challenge demanding multi-dimensional optimization. The methodology we apply to solve the deployment problem can be viewed as an intersection between systems development and novel network management solutions.

There have been a couple of promising developments providing platforms for adaptivity and dependability. Autonomous replication management mainly for dependability is targeted by Meling in a framework based on group communication systems [20]. A distributed dynamic middleware, QuAMobile is presented in [18] that introduces independent application variants and selects between them for context-awareness and adaptation. Planning is based on service level agreements (SLAs) and QoS-aware metadata in the service model is used in the planning-based adaptation middleware of the MUSIC project (cf. [22]). A peer-to-peer middleware, CARISMA is utilizing an auctioning-like mechanism for conflict resolution and adaptation automatically triggered by context changes [2]. In the QoS brokering approach from Menasce and Dubey consumers can request services, after which the broker uses analytic queuing models to predict QoS of the services under various workloads, thus looking for maximized utility [21].

Traditional techniques were applied for configuration of server environments such as fuzzy learning, e.g. Xu et al. applied a two-level control mechanism targeting efficient resource utilization in [26]. Layered queuing networks are employed for generating optimal configurations and policies by Jung et al. in an offline framework [12]. Some existing approaches have addressed the improvement of perceived QoS through changing the deployment of applications, however due to the exact solution algorithms, complexity becomes NP-hard already with more than 2-3 hosts or several QoS dimensions restricting applicability of these methods. A review of approximative solutions trying to overcome scaling problems, such as greedy algorithms, genetic programming, can be found in [19]. These approaches try to maximize util-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
AUTONOMICS 2008, September 23-25, Turin, Italy
Copyright © 2008 ICST 978-963-9799-34-9
DOI 10.4108/ICST.AUTONOMICS2008.4581

ity of a service purely from the users' perspective, whereas we aim to formulate and solve the deployment problem from the providers perspective while also considering the users perception of QoS. Besides, we aim to handle the deployment of multiple services at the same time.

We are building a logic that brings QoS-awareness into the development cycle, and that can manage the deployment of services and adapt to the context once these services are executed in a real environment. The application of our approach should grant the same benefits that exist in distributed management architectures, such as increased dependability, better resource utilization, etc. Moreover, the output of the logic presented is platform independent, thus it can drive a suitable middleware platform and will eventually allow adaptation to the changing environment of the modelled services that are executed.

It is an important design criteria that the deployment logic should allow execution in a fully distributed manner, thus it shall not be prone to deficiencies of existing centralized algorithms, such as performance bottlenecks and single point of failures. Besides, it is desirable to omit the burden of keeping a centralized decision logic updated and synchronized and this way to achieve better reaction times for context-awareness. Consequently, our aim is to develop a method supporting run-time component (re-)deployment that allows execution of services within the allowed region of external parameters defined by the service requirements. Considering all these aspects we approach the problem using a distributed, robust and adaptive routing system called the Cross Entropy Ant System (CEAS) [10, 8]. The CEAS is an Ant Colony Optimization (ACO) system as introduced by Dorigo et al. [6], which is a multi-agent system for solving a wide variety of combinatorial optimization problems where the agents' behavior are inspired by the foraging behavior of ants. Examples of successful application in communication systems are load-balancing (Schoonderwoerd et al. [24]), routing in wired networks by AntNet [3], and routing in wireless networks by AntHocNet [4].

In [5] we presented our novel approach for the efficient deployment of software components taking into account QoS requirements captured during the modelling phase. The procedure starts from high-level QoS goals and, through requirement profiles, utilizes swarm intelligence to provide solutions and to aid dynamic deployment. In [5] this distributed approach was tested on component deployment in a static topology and compared with centralized deployment approaches. In this paper, we extend the approach to allow deployment of multiple service components simultaneously that adapt to changing topologies.

The remainder of this paper is organized as follows. The next section will present how the deployment logic fits into the development cycle. In Sect. 3 an introduction to CEAS, which is used throughout the paper as the basis of our heuristic optimization method, will be given. Next, in Sect. 4 we present our proposed solution giving the algorithm. After that, Sect. 5 sets the example scenario of three concurrent services. The results related to the examples are evaluated in Sect. 6. Finally, in Sect. 7 we conclude and touch upon our future work.

2. DEPLOYMENT CYCLE

The deployment approach we are proposing will extend the development cycle SPACE. SPACE is devoted to the rapid and correct engineering of distributed services [14]. The stepwise modelling and refinement of the models is depicted in Fig. 1. First, a purely functional service model is created, which is collaboration-oriented meaning that the service specification is not a composition of descriptions of physical software components realizing the service. Instead, the collaboration-oriented specification is built from models of distributed sub-functionalities fulfilling — in interaction — the complete service behavior. The functional service model is specified by UML collaborations and activities. One of the advantages of this specification style is that it enables service modelling by reusing building blocks from collections of domain specific model libraries to a significantly higher degree than it would be possible with component-based descriptions [11].

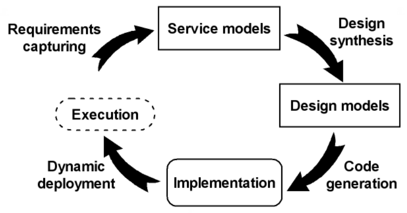

Figure 1: Development with SPACE

Service models undergo correctness checks, as described in [17], before they are transformed to a component-oriented design model by model transformation [15]. Next, using the component-oriented model, specified as UML state machines, code generators are used to create executable Java code enabling automated transformation of collaboration-oriented service models to executable implementations [16].

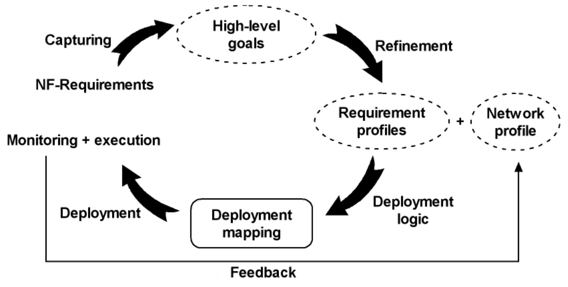

Figure 2: Deployment Support for SPACE

The dynamic deployment of the generated implementation is the step where the logic we propose can interact with the development cycle. The additional steps to support efficient deployment of the components that build up the service is shown in Fig. 2. In our deployment support cycle service models are amended by *high-level non-functional (NF) goals* that define non-functional requirements (NFRs) of the service being modelled in a rather abstract manner. Refinement of NF goals can be done in parallel with the transformation of service models to design models. *Requirement profiles* obtained in this step specify NFRs of the service components. In addition, a *network profile* is added representing provided properties describing the target environment the service will be executed in. Our deployment logic will be launched using these two profiles as input with requirements specifying the search goals and the *network profile* specifying the search space.

QoS requirements relevant to the service model are captured in a collaboration-oriented style design time, as a translation of traditional service level agreements. In NFRs, usually properties related to security, performance, availability, portability, etc. are addressed. More importantly, our view is that the deployment logic proposed will be able to handle any non-functional property of the service, as long as a suitable cost function is provided for the specific properties at hand. This feature will be provided by exploiting the advanced scalability of CEAS and the method of pheromone sharing.

Fig. 3 depicts a simple example of a collaboration between two components. This collaboration is enriched with NFRs for both the components and for the collaboration binding them. This basic collection of requirements contains two types of cost values, execution costs (f_{c_i}) and communication costs (f_{k_j}). The total number of cost values is equivalent to the total number of components in the service (E), plus the number of collaborations between them (K), i.e. $f_{c_i}, i = 1 \ldots E$ and $f_{k_j}, j = 1 \ldots K$. The execution cost is a local cost imposed on the host node or resource executing the particular component after deployment, whereas the communication cost loads the communication link between the two components involved in the collaboration. This simple example of collaboration-oriented specification and requirement capturing will be illustrated in the examples in Sect. 5.

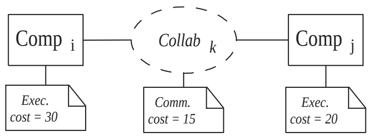

Figure 3: Collaboration with NFRs

Currently, existing deployment strategies and various approaches to aid deployment of software systems, e.g. ontology-based and reasoning engines, apply centralized decision logics based on centrally maintained databases. The disadvantages of approaching the problem this way are the burden of keeping a central database constantly updated and synchronized and the single point of failure introduced to the system. Accommodating the decision logic together with the central database on a single node may introduce bottlenecks both communication wise and storage wise.

In contrast to centralized approaches a distributed cooperative algorithm employs (semi-)autonomous agents, which cooperate to achieve certain common goals. To avoid the need for any type of global knowledge in deployment mapping, we employ autonomous agents operating in a distributed environment with their decisions based solely on information that is available locally to the place where they reside. At every node under the provision of the deployment logic some sort of shared memory is required that will be the vehicle for cooperation between the agents. Accordingly, the information required for optimization in our logic is distributed across all participating nodes. This property of the deployment mapping system contributes to robustness, scalability and fault tolerance. Furthermore, we intend to use the same logic first to obtain initial, optimal mapping of service components to hosts or resources, and second to guide necessary changes during execution of a service to satisfy the requirements it was launched with.

The objective of each ant species is to find either the optimal deployment mapping of component instances c_i onto nodes n_j or at least to find a mapping that satisfies the requirements within reasonable time. A component, $c_i \in \mathbf{C}$ ($\mathbf{C}$ is the set of components that together provide the service the species is responsible for) can have various properties and restrictions can apply regarding it's prospective hosts. For example, deployment of c_i can be restricted at node $n \in \mathbf{N}$ ($\mathbf{N}$ is the set of available nodes) as well as prescribed binding is allowed. A component can be bound to a node explicitly, this results in being excluded from the set of components that are available for the logic to be freely mapped to any available node. However, bound components are also taken into account by the ants during calculation of the resulting cost of a particular deployment mapping at a given iteration.

The basis of the heuristics, guiding the ants towards an optimized mapping, is the cost function $F(M)$ that is used to evaluate the resulting suggestion, $M : \mathbf{C} \rightarrow \mathbf{N}$, for deployment mapping. The ants target to minimize the cost calculated using $F(M)$ at every iteration, while the constraints given by the *mapping scopes* also have to be taken into consideration, i.e. $\mathbf{R}_i \subseteq \mathbf{N}$ for each component instance i. $\mathbf{R}_i$ is characterized by the policies given by the service provider (e.g. service level agreements of ISPs), as well as the access restrictions, the provided and requested capabilities (soft costs) and provided and requested capacity requirements (hard costs, e.g. bandwidth limitations). Using $\mathbf{R}_i$ component binding can easily be expressed assigning a single node to the set, thus restricting the search space.

Besides the *requirement profiles*, the service provider must provide the *net-map*, $\mathbf{N}$ for the decision logic as well, specifying the available nodes and links. Two types of constraints that can influence the optimal mapping are distinguished in the model. Constraints assigned to nodes and to links. For the latter type, constraints generally represent the cost of using the link for connecting two components, which have a functionality in the model requiring interaction between them. Constraints assigned to nodes or other resources related to execution of a component can, for instance, represent memory size limitations. Also, these constraints can be interrelated in a way that, for example, placement of a component on a node can lower the available amount of different types of resources at once with an amount depending on the available resources at the time of the mapping.

In the subsequent section the stochastic optimization background is introduced that is used throughout our logic to specify an algorithmic solution for the deployment problem.

3. CROSS ENTROPY ANT SYSTEM

The key idea is to let many agents, denoted *ants*, iteratively search for the best solution according to the problem constraints and cost function defined. Each iteration consists of two phases; the *forward* ants search for a solution, which resembles the ants searching for food, and the *backward* ants that evaluate the solution and leave markings, denoted *pheromones*, that are in proportion to the quality of the solution. These pheromones are distributed at different locations in the search space and can be used by forward ants in their search for good solutions; therefore, the best solution will be approached gradually. To avoid getting stuck in premature and sub-optimal solutions, some of the forward ants will explore the state space freely ignoring the pheromone values.

The main difference between various ant-based systems is the approach taken to evaluate the solution and update the pheromones. For example, AntNet [3] uses reinforcement learning while CEAS uses the *Cross Entropy (CE) method for stochastic optimization* introduced by Rubinstein [23]. The CE method is applied in the pheromone updating process by gradually changing the probability matrix $\mathbf{p}_r$ according to the cost of the paths. The objective is to minimize the cross entropy between two consecutive probability matrices $\mathbf{p}_r$ and $\mathbf{p}_{r-1}$. For a tutorial on the method, [23] is recommended.

The CEAS has demonstrated its applicability through a variety of studies of different path management strategies [8], such as shared backup path protection, p-cycles, adaptive paths with stochastic routing, and resource search under QoS constraints. Implementation issues and trade-offs, such as management overhead imposed by additional traffic for management packets and recovery times are dealt with using a mechanism called elitism [7] and self-tuned packet rate control [9]. Additional reduction in the overhead is accomplished by pheromone sharing [13] where ants with overlapping requirements cooperate in finding solutions by (partly) sharing information.

In this paper, the CEAS is applied to obtain the best deployment mapping $M : \mathbf{C} \to \mathbf{N}$ of a set of components, $\mathbf{C}$, onto a set of nodes, $\mathbf{N}$. The nodes are physically connected by links used by the ants to move from node to node in search for available capacities. A given deployment at iteration r is a set $\mathbf{M}_r = \{\mathbf{m}_{n,r}\}_{n \in \mathrm{N}}$, where $\mathbf{m}_{n,r} \subseteq \mathbf{C}$ is the set of components at node n at iteration r. In CEAS applied for routing the path is defined as a set of nodes from the source to the destination, while now we define the path as the deployment set $\mathbf{M}_r$. The cost of a deployment set is denoted $F(\mathbf{M}_r)$. Furthermore, in the original CEAS we assign the pheromone values $\tau_{ij,r}$ to interface i of node j at iteration r, while now we assign $\tau_{mn,r}$ to the component set m deployed at node n at iteration r. In Sect. 4 we describe the search and update algorithm in details.

In CEAS applied for routing and network management, selection of the next hop is based on the *random proportional rule* presented below. In our case however, the *random proportional rule* is applied for deployment mapping. Accordingly, during the initial exploration phase, the ants randomly select the next *set of components* with uniform probability $1/E$, where E is the number of components to be deployed, i.e. the size of $\mathbf{C}$, while in the normal phase the next set is selected according to the *random proportional rule* matrix $\mathbf{p}_r = \{p_{mn,r}\}$, where

$$p_{mn,r} = \frac{\tau_{mn,r}}{\sum_{l \in \mathbf{M}_{n,r}} \tau_{ln,r}} \tag{1}$$

A parameter γ_r denoted the *temperature*, controls the update of the pheromone values and is chosen to minimize the performance function

$$H(F(\mathbf{M}_r), \gamma_r) = e^{-F(\mathbf{M}_r)/\gamma_r} \tag{2}$$

which is applied to all r samples and the expected overall performance satisfies

$$h(p_{mn,r}, \gamma_r) = E_{\mathbf{p}_{r-1}}(H(F(\mathbf{M}_r), \gamma_r)) \geq \rho \tag{3}$$

$E_{\mathbf{p}_{r-1}}(X)$ is the expected value of X s.t. the rules in $\mathbf{p}_{r-1}$, and ρ is a parameter (denoted search focus) close to 0 (typically 0.05 or less). Finally, a new updated set of rules, $\mathbf{p}_r$, is determined by minimizing the cross entropy between $\mathbf{p}_{r-1}$ and $\mathbf{p}_r$ with respect to γ_r and $H(F(\mathbf{M}_r), \gamma_t)$. Minimized cross entropy is achieved by applying the random proportional rule in (1) for $\forall_{mn}$ with

$$\tau_{mn,r} = \sum_{k=1}^{r} I(l \in \mathbf{M}_{n,r}) \beta^{\sum_{j=k+1}^{r} I(j \in \mathbf{M}_k)} H(F(\mathbf{M}_k), \gamma_r) \tag{4}$$

where $I(x) = 1$ if x is true, 0 otherwise. See [23] for further details and proof.

To avoid centralized control and synchronized batch oriented iterations, in CEAS the cost value $F(\mathrm{M}_r)$ is calculated *immediately* after each sample, i.e., when all components are mapped, and an auto-regressive performance function, $h_r(\gamma_r) = \beta h_{r-1}(\gamma_r) + (1 - \beta) H(F(\mathrm{M}_r), \gamma_r)$ is applied approximated by

$$h_r(\gamma_r) \approx \frac{1-\beta}{1-\beta^r} \sum_{i=1}^{r} \beta^{r-i} H(F(\mathrm{M}_r), \gamma_r) \tag{5}$$

where $\beta \in < 0, 1 >$ is a memory factor weighting (geometrically) the output of the performance function. The performance function will smoothen variations in the cost function, hence rapid changes in the deployment mapping and undesirable fluctuations will be avoided. This mechanism helps cooperation between the species.

As for the CE method, the temperature γ_r is determined by minimizing it subject to $h(\gamma) \geq \rho$. In [10] it is shown that the temperature equals

$$\gamma_r = \{\gamma \mid \frac{1-\beta}{1-\beta^r} \sum_{i=1}^{r} \beta^{r-i} H(F(\mathbf{M}_i), \gamma) = \rho\} \tag{6}$$

However (6) is a complicated (transcendental) function that is both storage and processing intensive since all observations up to the current path sample, i.e. the entire path cost history $F(\mathbf{M}_r) = \{F(\mathbf{M}_1), \cdots, F(\mathbf{M}_r)\}$ must be stored, and weights for all observations have to be recalculated. In an on-line operation of a network node, such resource requirements are impractical. Instead it is assumed, given a β close to 1, that the changes in γ_r are typically small from one iteration to the next. This enables a first order Taylor expansion of (6), and a second order Taylor expansion of (4), see [10, 25] for more details.

4. DISTRIBUTED DEPLOYMENT LOGIC

Our deployment logic can be considered as a swarm of independent ant-like agents executing an optimization task continuously in the target network hosting the service we model. This continuous behavior contributes to the advantage of our approach, i.e. that the same logic provides an initial static mapping and can be used for online redeployment. In this paper we extend the deployment approach to handle multiple services deployed simultaneously by allowing interoperation of artificial ant species, each of them representing a particular service realized by distributed software components. More importantly, beyond achieving a scalable extension for multiple services we target other scalability issues as well. By using a new cost function for evaluating solutions during the heuristical optimization process we eliminate the need for any global knowledge, i.e. species can now be launched and operate independently from each other without having a global view on the requirements set for all the services in the system. Another step towards scalability is to limit the ants to visit only those nodes within the network-map that are effectively used for deploying components of the service represented by their species. This way, we address significantly larger problem sizes, consisting of a higher number of nodes and simultaneous services.

Initially, each ant is assigned a task of deployment of $\mathbf{C}$ components stemming from the set of components of the service represented by its species. After initialization the ants start a random-walk in the network of nodes, described by the *net-map*, selecting every next hop randomly. After an ant arrives at a node its behavior depends on if it is a *explorer* or a *normal* ant. The latter type of ant uses the corresponding instance of the distributed pheromone database at the node to select a subset, $\mathbf{m}_{n,r}$, of $\mathbf{C}$ for mapping and stores this selection in a mapping list. The mapping list $\mathbf{M}_r$ is carried along by the ant during its search. On the contrary, an *explorer* ant selects a subset $\mathbf{m}_{n,r}$ based on a random decision without using the pheromone values available at the node. The ratio of *explorer* ants can be regulated and this type of ants are used to initially explore the *net-map* and also to cover up fluctuations, e.g. new nodes appearing, in the network later on in the optimization process. This type of exploration can be considered as random sampling from the problem space and results in a random cost figure. The number of iterations in the initial *exploration* phase depends on the problem size, however, the end of this phase can be detected by monitoring the pheromone database size as it extends with the growing number of possibilities covered up by the ants doing a random-walk in the problem space. After the *normal* phase starts only a fraction of the ants, e.g. 5-10%, are flagged as *explorers*, allowing for the required responsiveness to changes in the environment, while *normal* ants are focusing on finding the optimum.

To support finding the optimal deployment mapping for multiple concurrent services a means of interoperation is needed among the species responsible for the different services being deployed. As we consider optimal deployment of services from the provider's perspective we target balancing of execution costs imposed on the nodes that are used for execution of the services, or in other words load-balancing, which generally requires global overview of the system's operating conditions. Nevertheless, we want to avoid any centralized structure, and use a completely distributed optimization method. For this reason, we have introduced a processing power reservation mechanism that has to be implemented in every node in addition to the pheromone database. The different ant species use this allocation mechanism to indicate their latest resource usage in a node at iteration r. As ants from every species use the allocation in every node they actually use for deployment mapping, this mechanism will provide interaction between the components. Sampling the current sum of allocations in every visited node can give a general overview for the ants, thus load-levels in participating nodes can be incorporated into the cost calculations at the end of each iteration. We refer to load-level samples taken during an ant run with the set $\mathbf{NL}_r$. Samples that suggest exceeding the capacity of a node are quickly outranked by better solutions as a high penalty is assigned to infeasible solutions. The actual implementation of sampling is left to the middleware. Allocation entries that are outdated are invalidated to preserve consistency.

After the forward search is over, i.e. an ant managed to come up with a mapping for all the components it was assigned with, the resulting mapping can be found in the set $\mathbf{M}_r$ and will be evaluated using the cost function of the service. How to formulate the cost function $F()$ depends on the NFRs that the service model is extended with. Currently, we use two parameters for the cost function, the deployment mapping set $\mathbf{M}_r$ and the load-level samples taken during an iteration $\mathbf{NL}_r$ and we consider execution and communication costs derived from the service model as introduced in Sect. 2. Thus, our cost function consists of two components, node related costs ($\mathbf{NC}$) and link, i.e. collaboration related costs ($\mathbf{LC}$). The aim is to minimize the overall value of (7).

$$F(\mathbf{M}_r, \mathbf{NL}_r) = [\sum_{\forall n_j \in \mathbf{H}_r} \mathbf{NC}(n_j)] \cdot (1 + x \cdot \mathbf{LC}) \tag{7}$$

where x is a parameter. $F(\mathbf{M}_r, \mathbf{NL}_r)$ is used by all species the same way, and has a component strictly local to the species, $\mathbf{LC}$, which incorporates the collaboration costs

$$\mathbf{LC} = \sum_{j=1}^{K} I_j \; f_{k_j} \tag{8}$$

where I_j is an indicator function to sum all the communication costs of the collaborations that happen between different nodes

$$I_j = \begin{cases} 1, & \text{if } k_j \text{ external} \\ 0, & \text{if } k_j \text{ internal to a node} \end{cases} \tag{9}$$

The first component, $\sum_{\forall n_j \in \mathbf{H}_r} \mathbf{NC}(n_j)$, of the overall cost function is related to node local costs and aims to incorporate load-balancing among the nodes providing the services being executed. Furthermore, it is important to note that only those nodes that are visited during the search phase in iteration r are included in the hop-list, $\mathbf{H}_r$. The node related cost is calculated individually for every visited node according to

$$\mathbf{NC}(n_j) = [\sum_{i=0}^{\mathbf{NL}_{n,r}(n_j)} \frac{1}{\sum_{\forall n_j \in \mathbf{H}_r} \mathbf{NL}_{n,r} + 1 - i}]^y \tag{10}$$

Equation (10) calculates the execution costs for node n_j based on the load-levels sampled and it is the basis for counteracting the cost component $\mathbf{LC}$. On one hand, $\mathbf{LC}$ tries to put weight on component mappings that have as much as possible of the collaborations within the same node(s) by favoring mappings that use less nodes for deployment with a low cost value. This way minimizing external communication. On the other hand, Equation (10) has an effect of distributing components, thus equalizing execution load among the available hosts to the highest extent possible. This way two counteracting requirement types are tackled in the same cost function. The exponent $\mathbf{y}$ (in Equation (10)) allows to focus more on load-balancing instead of minimization of collaboration costs by selecting a larger value, while the multiplier $\mathbf{x}$ (in Equation (7)) can be used to scale collaboration costs if needed. We generally use $\mathbf{x} = 0.1$ and $\mathbf{y} = 2$, as well as with the cost values in the example scenario presented in Sect. 5.

The cost function (7) is used at the end of an iteration to evaluate the mapping found by the ant. Thereafter, the ant travels backward along the path stored in the hop-list $\mathbf{H}_r$. This mechanism is called *backtracking.* During *backtracking* the pheromone values are updated according to Equation (4). This ends the behavior of a single ant and unless a stopping criteria is met a new ant can be initiated and emitted. There are different options for constructing a stopping criteria. One can be for example the observation of the moving average of the evolving cost value and detecting convergence to a suggested solution. Another option is sampling the size of the distributed pheromone database during an iteration. Convergence can be detected by observing a very strong pheromone value that will emerge in the database, while inferior solutions will evaporate.

It is important to note that the same ant behavior can be used for all the species, i.e. for all the services being deployed simultaneously. The described process is summarized in Algorithm 1.

Algorithm 1 Deployment mapping of $\mathbf{C}$

```
1. Select the initial node $n \in \mathbf{N}$ where the search will start randomly.
2. Select a set of components $\mathbf{m}_{n,r} \subseteq \mathbf{C}$ which satisfies $n \in \mathbf{R}$ for every $c_i \in \mathbf{m}_{n,r}$ according to the random proportional rule (normal ant), Equation (1), or in a totally random manner (explorer ant). If such a set cannot be found, goto step 7.
3. Update the ant's deployment mapping set, $\mathbf{M}_r = \mathbf{M}_r + \{\mathbf{m}_{n,r}\}$.
4. Update the set of components to be deployed, $\mathbf{C} = \mathbf{C} - \mathbf{m}_{n,r}$.
5. (Re-)allocate processing power at the current node, $n$ according to $f_{c_i}, \forall c_i \in \mathbf{m}_{n,r}$.
6. Sample the estimated load-level, $\mathbf{nl}_{n,r}$ at the current node $n$, and $\mathbf{NL}_r = \mathbf{NL}_r + \{\mathbf{nl}_{n,r}\}$.
7. Select next node, $n$ randomly and add $n$ to the hop-list $\mathbf{H}_r = \mathbf{H}_r + \{n\}$.
8. If $\mathbf{C} \neq \emptyset$ then goto 2., otherwise evaluate $F(\mathbf{M}_r, \mathbf{NL}_r)$ using the mapping set $\mathbf{M}_r$ and the samples taken ($\mathbf{NL}$).
9. Update the pheromone values, Equation (4), corresponding to the $\{\mathbf{m}_{n,r}\} \in \mathbf{M}_r$ mappings going backwards along $\mathbf{H}_r$.
10. If stopping criteria is not met then start new iteration (increment $r$), initialize and emit new ant and goto 1.
```

For optimization to be successful the pheromone values have to be aligned with the sets of deployed components. During an iteration each ant visits $\mathbf{n} \subseteq \mathbf{N}$ nodes and will form n discrete sets from the available components ($\mathbf{C}$) carried along. At the end of an iteration the suggested deployment mapping, $\mathbf{M}_r$ is evaluated. The pheromone database for each species is built by assigning a flag to every component that is free for deployment mapping, i.e. which is not bound to a specific node by requirements. Thus, the number of components available for the species will be $E^* \subseteq E$, and the size of the pheromone database becomes 2^{E^*}, equal to the number of possible combinations for a set at a node, which is specific for each service. Accordingly, the physical requirement for an execution platform supporting our approach is to accommodate 2^{E^*} floating point numbers at every node. If the pheromone database in a node is normalized between $0 \ldots 1$ it can be observed as a probability distribution of component sets mapped to that node by the artificial ants. Once a converged state is reached the optimal solution(s) emerge with probability one.

Indexing of the pheromone database can be done using component set identifiers. For example, consider a basic set of 5 components in a service, $\mathbf{C} = \{c_1, c_2, c_3, c_4, c_5\}, E = 5$. Then indexing is done using an E long binary bitstring. In this case, e.g. element 17 of the pheromone database, which is equivalent to $'10001'B$, refers to the deployment of components c_1, c_5 at the current node. Besides, we propose to use a dynamically allocated pheromone database based on thresholds that can be used for evaporating pheromone entries under a given significance level to achieve better scalability. Currently, we apply a threshold of 1%, i.e. pheromone values lower than 1% of the highest value are considered insignificant and are eliminated from the database.

5. DESIGN EXAMPLES

In this section we introduce 3 different service models for demonstrating the deployment logic. The first example has been introduced originally in [14]. $S1$ has a component that operates a security door and a card reader with a keycode entry panel. The two latter components are bound to n_1 by requirements. Besides, a central component administers access rights by using an authentication and a authorization server with corresponding databases as separate components (Fig. 4).

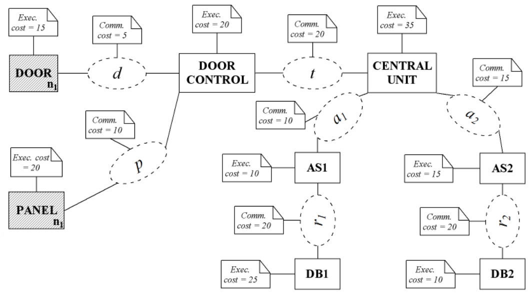

Figure 4: S1 - The Access Control System

The second example, Fig. 5 models a video surveillance system that has one surveillance camera component bound to each of the five nodes by default. A central control and a recording unit manages the system and uses a main and a backup storage device for storing surveillance information in a replicated database.

$S3$, the third service is a model of a process controller that consists of 4 main stages of processing. In addition, S3 has a main generator component that produces the input impulses for the processing stages and a logging module monitoring the output of the four stages. On top of that, a user interface component can be used for direct human interaction with the system (Fig. 6). In $S3$ all the components can freely be mapped to any of the nodes in N, depending on current availability of resources.

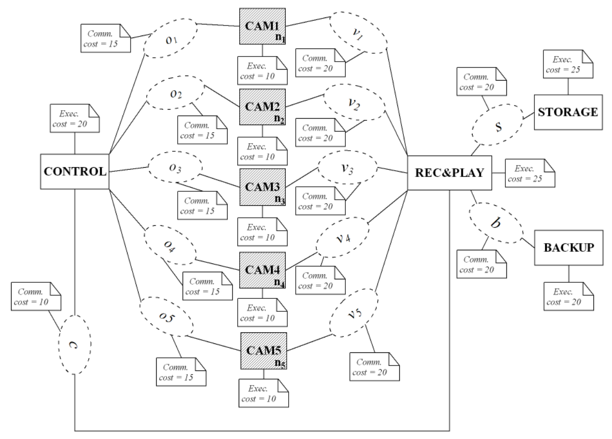

Figure 5: S2 - The Video Surveillance System

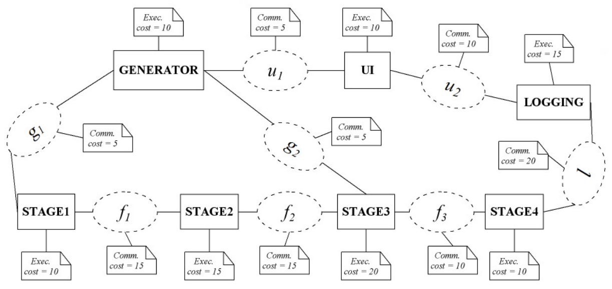

Figure 6: S3 - The Process Controller System

An ant species is assigned to each of the services and deployment mapping is conducted on the underlying network of hosts, which consists of 5 nodes with equivalent capabilities in the example setting. The execution and collaboration costs assigned to each element of the models are summarized in Table 1.

Table 1: Components and costs in the examples

S1 - Access Control		S2 - Surveillance		S3 - Process Control.	
ci / kj	fc / fk	ci / kj	fc / fk	ci / kj	fc / fk
DOOR	15	CAM1..5	10	GENERATOR	5
PANEL	20	CONTROL	20	STAGE1	10
DOOR CTRL	20	REC./PLAY	25	STAGE2	15
CENTRAL UNIT	35	STORAGE	25	STAGE3	20
AS1	10	BACKUP	20	STAGE4	10
AS2	15			LOGGING	15
DB1	25			UI	10
DB2	10				
d	5	o1..5	15	g1..2	5
p	10	v1..5	20	u1	5
t	20	c	10	u2	10
a1	10	s	20	f1	15
a2	15	b	20	f2	15
r1	20			f3	10
r2	20			l	20

The behavior and the output of our artificial intelligence approach will further be evaluated in the next section.

6. EVALUATING THE SCENARIO

The service deployment mapping problem with execution and communication costs can be NP-hard even in case of a single service (c.f. [5]). The example provided here has multiple optimal and near-optimal solutions with different sets of components deployed on various nodes. However, it is important to recall that we are interested in providing solutions satisfying the requirements in reasonable time and not necessarily in always finding the optimum. For demonstration a solution taken from the output of the logic is shown in Table 2.

Table 2: Example deployment mapping

	S1	S2	S3
n1	DOOR, PANEL	CAM1	STAGE4, LOGGING
n2	AS2, DB2	CAM2, REC/PLAY, STORAGE, BACKUP	
n3	DOOR CTRL.	CAM3	STAGE1, STAGE2, STAGE3
n4	AS1, DB1	CAM4	UI
n5	CENTRAL UNIT	CAM5, CONTROL	GENERATOR

To evaluate the algorithm we propose, first we compare it to two different approaches. In the first one (denoted $globT, allnodes$) ants use a simpler cost function, Equation (11) that is easier to calculate, but requires the shared knowledge of the sum of all offered execution costs, T.

$$F(\mathbf{M}_r) = \sum_{\forall n \in N} |\mathbf{nl}_{n,r} - T| + \sum_{j=1}^{K} I_j \ f_{k_j} \tag{11}$$

That means that to apply (11) all of the species associated to the multiple services being deployed simultaneously have to be aware of each others total processing power demand and incorporate it into T. Knowing the global constant, T, ants can calculate the deviation of the execution load from a global average, i.e. share the load among the participating nodes. This is included in the first part of (11). The second part of this cost function includes collaboration costs the same way as in Equation (8). For details see [5].

The second approach used for comparison (denoted $globT$) uses the same cost function, i.e. Equation (11), with the exception that ants are not required to visit and sample all the nodes available in the *net-map*, only those that are actually used for deployment by their species, i.e. we use $n \in \mathbf{H}_r$ instead of $\forall n \in N$. This is a significant difference with respect to scaling as the set of available nodes, N can be high thus putting a heavy burden on the ants that have to visit and sample all of the nodes. Our approach, Algorithm 1, aims to provide deployment mapping without requiring any *global* prerequisite and also without driving ants to cover $\forall n \in N$, which contributes to increased scalability. Algorithm 1, that uses Equation (7) as cost function, is denoted $distF$ throughout this section.

Table 3: Number of iterations until convergence

	avg	stdev
globT, all nodes	23679	14795
globT	18487	5469
distF, 5%	8234	12800

First, in Table 3 we compare the amount of iterations, i.e. ant runs executed before reaching a converged state. The results are derived from the output of 100 runs of each approach and we conclude that $distF$ requires significantly less iterations to converge compared to the semi-*global* approaches. The 5% indicates the percentage of explorer ants used during the *normal phase* that is standard procedure to achieve context-awareness by constantly allowing some of the ants randomly explore the *net-map*.

It can be noted that if adaptability to events such as nodes appearing/disappearing or link fluctuations is not necessary, hence an initial deployment mapping satisfying the

NF-requirements is sufficient, the search for a static mapping can even more be accelerated.

In Table 4 we compare the average and the deviation of the converged cost values produced by the three different approaches solving the simultaneous deployment mapping of the example services $S1$, $S2$ and $S3$. For this comparison the converged deployment mapping suggested by the different approaches is observed and evaluated by applying the cost function Equation 7.

Table 4: Average cost of mapping

	S1	S2	S3
globT, all nodes, avg	2.0246	4.1725	1.7251
globT, all nodes, stdev	0.0513	0.1754	0.297
globT, avg	1.87	4.7977	1.7033
globT, stdev	0.1272	0.2275	0.1818
distF, 5%, avg	1.7945	4.3021	1.5932
distF, 5%, stdev	0.0943	0.1773	0.0588

We can see that the deviation of the costs values in *distF* is at least as low as in case of the least scalable approach, *globT* with *all nodes* sampled. Besides, the actual cost values for *distF* are the best for services $S1$ and $S3$, and for $S2$ it is between the least scalable and the scalable *globT* versions. Accordingly, our algorithm works without the requirement for any *global* knowledge and using a more scalable sampling of nodes while producing equivalent or better results for the deployment problem.

To evaluate the capability of the logic to adapt to changes in the context we have investigated two simple scenarios with the example services. In the first scenario a single node failure occurs and sometime later on the node is repaired and operational again. However, we experiment with *soft-errors* meaning that deployment to the selected node will be disallowed for the species after the error event occurs, but the node will still be operational for the components that are bound to it by requirements. We allow this exception for those services that would otherwise go down and hence the deployment mapping would be meaningless for them (e.g. $S2$, which has a component bound to each of the five nodes). The second scenario introduces an additional new node to the network after the species have converged to a solution with 5 nodes.

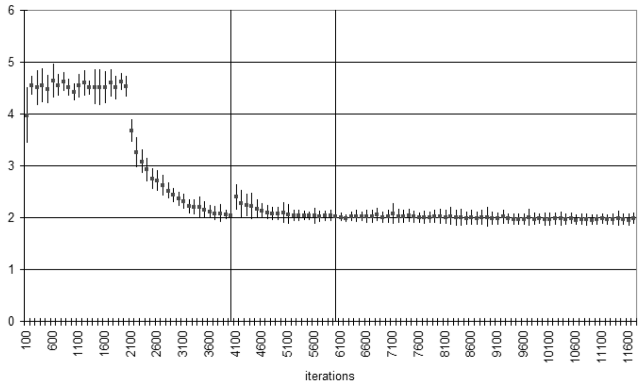

Figure 7: Costs for S1 with node error and repair

The results of injecting a node error followed by a repair of the same node later on can be seen in Figures 7, 8 and 9. These figures show how the cost values found by the three species evolve. The figures display the average (as dots) and the deviation (as error bars) of results from 100 runs of the same scenario. The first 2000 iterations represent the initial *exploration* phase with considerably high costs, but as the *normal* phase starts from iteration 2001 the costs of component mappings start to decrease as species start to cooperate and better and better solutions are found. Vertical markers show the events of node n_3 going down and then coming back to operation. We can see that the cost values increase for every service after a node goes down, but interestingly species for $S1$ and $S3$ are able to find an almost equally low cost level by redeploying their components on the remaining nodes. Species corresponding to $S2$ in turn remains at a slightly higher cost level because it has a component bound to the erroneous node thus it has to face degraded load-sharing among the nodes. After n_3 is repaired components of $S2$ can be re-mapped to achieve a lower cost level again with 5 nodes, which happens within a few iterations.

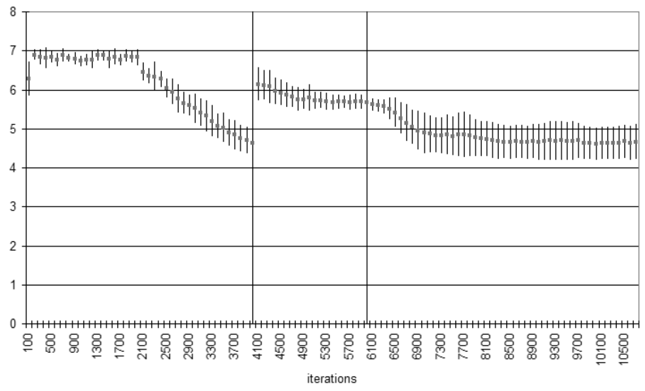

Figure 8: Costs for S2 with node error and repair

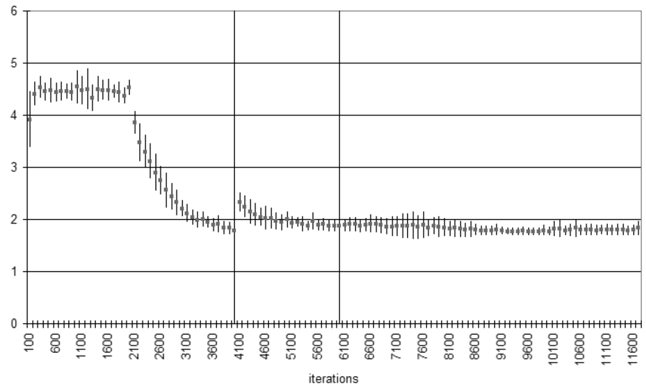

Figure 9: Costs for S3 with node error and repair

It is also interesting to observe the control parameter γ_r, i.e. the temperature that governs the performance function during a run with error and repair events. In Figure 10 we can observe the changes in the temperature and see how species react to changes in the environment. First, the node failure is detected very quickly and the temperature increases as lower cost solutions disappear from the solution space. The temperature increase is significant in $S2$ that is mostly affected by the change but it is also noticeable for $S3$, whereas $S1$ seems to be slightly influenced. After the repair has been made the swarm reacts slower only after some iterations can we observe decreasing temperatures. $S3$ notices the better conditions first and $S2$ follows. More iterations later all species converge to a stabilized state with lowest temperatures.

Our second test scenario introduces a 6th node to the environment of the services. The 3 species are already in a

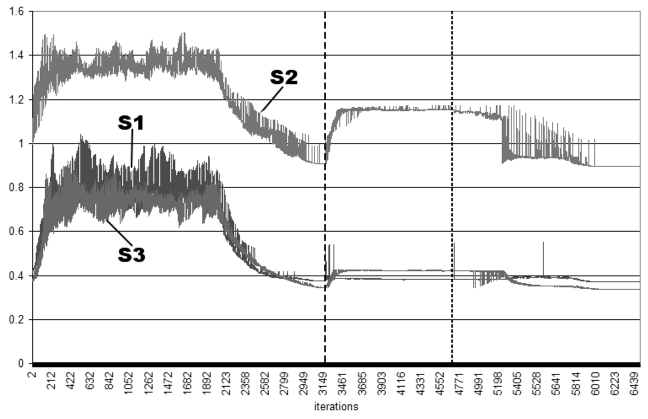

Figure 10: Temperature within the 3 species

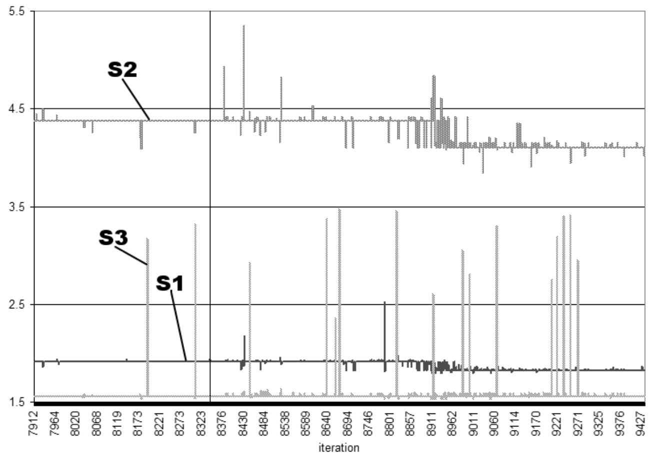

Figure 11: Costs after a new node appears

converged state when the new node appears, indicated by a vertical line in Figure 11. Some iterations later explorer ants, which represent 5% of all ants in every species, start to indicate that a better, lower cost solution exists in the changed environment. Deployment mapping of services $S1$ and $S2$ is adapted to the changes giving a somewhat lower cots value, while the cost level for $S3$ does not change with an additional node.

Moreover, in Figure 12, the number of pheromone entries in nodes $n_1 \ldots n_6$ corresponding to $S1$ can be seen over time. As the problem space is explored initially the database size tops somewhat below 2^6 as $E^* = 6$ for $S1$ (cf. Section 4). After exploration the swarm quickly converges to a low cost solution so there is little variation in the pheromones. The additional node is inserted at the iteration indicated by a vertical line, before that node n_6 has an empty pheromone table. Explorer ants discover the new node quickly, thus new entries appear in the database pointing to mappings with a lower cost. After a short period of fluctuation caused by the re-reservation of processing power by the 3 species the database contains a few pheromone entries until convergence is completed and only the optimal mapping remains.

7. CONCLUSIONS

We presented a new swarm intelligence logic for the efficient deployment mapping of software components to execution resources. A model-driven approach was presented that

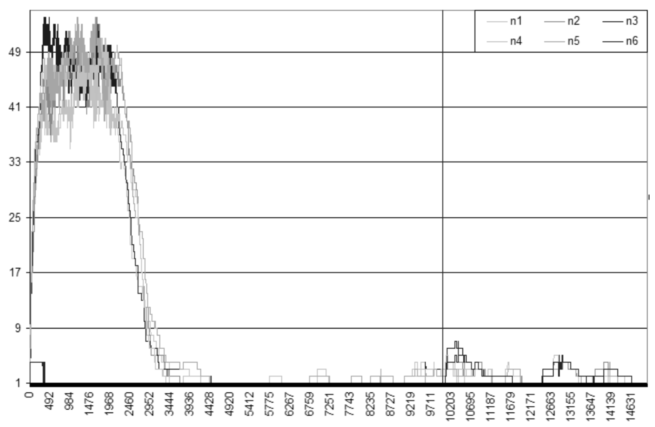

Figure 12: Database size for $S1$, with a 6th node inserted

drives optimization of the component mapping using non-functional requirements incorporated into the model during the modelling phase. The deployment logic is executed in a fully distributed manner, thus it is free of deficiencies of most of the existing centralized approaches, such as performance bottlenecks and single points of failure. The presence of a central database or decision entity is not required to run the logic, instead we use the analogy of pheromones distributed across the network of execution hosts to store information. All the intelligence is carried along by ant-like agents.

Besides, we have showed that using CEAS our deployment logic is capable of handling multiple services simultaneously and does not require global knowledge to achieve better load-balancing among the nodes, while striving to minimize remote communication at the same time. Our goal is to develop support for run-time redeployment of components to keep the services within an allowed region of parameters defined by the requirements. With methods in CEAS an the development cycle SPACE we target a robust and adaptive service execution platform. Furthermore, we intend to address scalability issues and consider larger network domains within the deployment problem.

Considering convergence time we have a trade-off between convergence speed and solution quality. Nevertheless, while deploying services in a dynamic environment pre-mature solutions satisfying both functional and non-functional requirements often suffice. More importantly, ACO systems have been proven to be able to find the optimum at least once with probability close to one and as this happens convergence to the optimum is secured in a finite number of iterations. The optimal deployment mapping can be obtained with high confidence since CEAS can be considered as a subclass of ACO algorithms. Another advantage of our approach is the capability to provide alternative solutions weighted by their cost values, which can be selected for deployment easily as the corresponding pheromones indicate their proposed mapping. Currently, the deployment logic is implemented in a simulator written in the Simula/DEMOS language [1] for evaluation purposes.

Our work is conducted in cooperation with the ISIS (Infrastructure for Integrated Services) project funded by the Research Council of Norway. The algorithm and approach presented are in-line with the objectives of ISIS that are to create an established service engineering platform for collaboration-oriented models, covering the development cycle from the requirements to seamless execution in a heterogenous and dynamic environment.

Future work on the topic will investigate inclusion of a wider range of QoS requirements and develop necessary improvements on the cost functions. It is an interesting topic to look into how larger networks of nodes influence scalability and convergence times. Similarly, introduction of a new type of species corresponding to user demands towards services targeting better resource utilization possesses challenges. Besides, we plan to experiment with distributed optimization methods other than the CE method guiding the ant-based deployment logic and also to do more extent comparison between state-of-the-art optimization methods and our work.

8. REFERENCES

[1] G. Birtwistle. *Demos - a system for Discrete Event Modelling on Simula.* 2003.

[2] L. Capra, W. Emmerich, and C. Mascolo. Carisma: Context-aware reflective middleware system for mobile applications. *IEEE Trans. on Software Engineering*, 29(10):929–945, 2003.

[3] G. D. Caro and M. Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9, 1998.

[4] G. D. Caro, F. Ducatelle, and L. M. Gambardella. Anthocnet: An adaptive nature-inspired algorithm for routing in mobile ad hoc networks. *European Trans. on Telecomm. (ETT) - Special Issue on Self Organization in Mobile Networking*, 16(5), 2005.

[5] M. J. Csorba, P. E. Heegaard, and P. Herrmann. Cost-efficient deployment of collaborating components. In *Proc. of the 8th Int'l Conf. on Distributed Applications and Interoperable Systems (DAIS), LNCS 5053, Oslo.* IFIP, June 2008.

[6] M. Dorigo et al. The ant system: Optimization by a colony of cooperating agents. *IEEE Trans. on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1), 1996.

[7] P. E. Heegaard et al. Distributed asynchronous algorithm for cross-entropy-based combinatorial optimization. In *Rare Event Simulation and Combinatorial Optimization, Budapest*, 2004.

[8] P. E. Heegaard, B. E. Helvik, and O. J. Wittner. The cross entropy ant system for network path management. *Telektronikk*, 104(01):19–40, 2008.

[9] P. E. Heegaard and O. Wittner. Self-tuned refresh rate in a swarm intelligence path management system. In *Proc. of the EuroNGI Int'l. Workshop on Self-Organizing Systems, LNCS 4124*, 2006.

[10] B. E. Helvik and O. Wittner. Using the cross entropy method to guide/govern mobile agent's path finding in networks. In *Proc. of 3rd Int'l Workshop on Mobile Agents for Telecommunication Applications*, 2001.

[11] P. Herrmann and F. A. Kraemer. Design of trusted systems with reusable collaboration models. In *Proc. of the Joint IFIP iTrust and PST Conferences on Privacy, Trust Management and Security, Moncton*, 2007.

[12] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu. Generating adaptation policies for multi-tier applications in consolidated server environments. In *Proc. of the Int'l. Conf. on Autonomic Computing (ICAC)*, 2008.

[13] V. Kjeldsen, O. Wittner, and P. E. Heegaard. Distributed and scalable path management by a system of cooperating ants. In *Proc. of the Int'l. Conf. on Communications in Computing (CIC)*, 2008.

[14] F. A. Kraemer and P. Herrmann. Service specification by composition of collaborations - an example. In *Proc. of the 2006 Int'l Conf. on Web Intelligence and Intelligent Agent Technology, Hong Kong.* IEEE/WIC/ACM, 2006.

[15] F. A. Kraemer and P. Herrmann. Transforming collaborative service specifications into efficiently executable state machines. *Electronic Communications of the EASST*, 6, 2007.

[16] F. A. Kraemer, P. Herrmann, and R. Bræk. Aligning uml 2.0 state machines and temporal logic for the efficient execution of services. In *Proc. of the 8th Int'l Symp. on Distributed Objects and Applications (DOA), LNCS 4276, Montpellier*, 2006.

[17] F. A. Kraemer, V. Slåtten, and P. Herrmann. Engineering support for uml activities by automated model-checking - an example. In *Proc. of the 4th Int'l Workshop on Rapid Integration of Software Engineering Techniques (RISE), University of Luxembourg*, 2007.

[18] S. A. Lundesgaard, A. Solberg, J. Oldevik, R. France, J. Ø. Aagedal, and F. Eliassen. Construction and execution of adaptable applications using an aspect-oriented and model driven approach. In *Proc. of DAIS, LNCS4531*, pages 76–89. IFIP, 2007.

[19] S. Malek. A user-centric framework for improving a distributed software system's deployment architecture. In *Proc. of the doctoral track at the 14th ACM SIGSOFT Symp. on Foundation of Software Engineering, Portland*, 2006.

[20] H. Meling. *Adaptive Middleware Support and Autonomous Fault Treatment: Architectural Design, Prototyping and Experimental Evaluation.* PhD thesis, NTNU, Dept. of Telematics, Norway, 2006.

[21] D. Menasce and V. Dubey. Utility-based qos brokering in service oriented architectures. In *Proc. of the Int'l Conf. on Web Services (ICWS), Salt Lake City, Utah*, July 2007.

[22] R. Rouvoy, F. Eliassen, J. Floch, S. Hallsteinsen, and E. Stav. Composing components and services using a planning-based adaptation middleware. In *Proc. of SC, LNCS4954*, pages 52–67. Springer-Verlag, 2008.

[23] R. Y. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Prob.*, 1999.

[24] R. Schoonderwoerd et al. Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, 5(2), 1997.

[25] O. Wittner. *Emergent Behavior Based Implements for Distributed Network Management.* PhD thesis, NTNU, Dept. of Telematics, Norway, 2003.

[26] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif. On the use of fuzzy modeling in virtualized data center management. In *Proc. of the Int'l. Conf. on Autonomic Computing (ICAC)*, 2007.