Framework for Ensuring Runtime Stability of Control Loops in Multi-agent Networked Environments

Nikolay Tcholtchev and Ina Schieferdecker

Fraunhofer FOKUS Institute for Open Communication Systems, Berlin, Germany
{nikolay.tcholtchev,ina.schieferdecker}@fokus.fraunhofer.de

Abstract. The idea of autonomic computing, and accordingly autonomic networking, has drawn the attention of industry and academia during the past years. An autonomic behavior is widely understood as a control loop which is realized by an autonomic entity/agent that manages some resources, in order to improve the performance and regulate diverse operational aspects of the managed network or IT infrastructure. Self-management, realized through autonomic behaviors, is an appealing and dangerous vision at the same time. On one hand, it promises to reduce the need for human involvement in the network and system management processes. On the other hand, it bears a number of potential pitfalls that could be even dangerous to the network, the IT infrastructure, and the corresponding services. One of these pitfalls is constituted by the stability of the control loops, and correspondingly by the interference among multiple autonomic agents operating in parallel. In this paper, a novel approach to ensuring runtime synchronization and stability of multiple parallel autonomic control loops is presented. We formally model the problem of runtime action synchronization, propose different possible solutions, and provide a case study, as well as different performance measurements based on a prototype that implements our approach.

Keywords: Multi-Agent Systems, Autonomic Networks, Stability, Control Loops.

1 Introduction

The continuously increasing complexity of network and systems management has provoked a discussion on the possibility to extend management processes into the device architectures. That way, a certain degree of autonomic decision making within the network or IT infrastructure in question is enabled. This can be best realized by introducing autonomic software agents inside the managed equipment. These agents monitor the surrounding networked environment, exchange information, and undertake corrective or regulative actions. A number of models aiming to realize autonomic control loops have been proposed in the past decade. Initially, IBM introduced an entity called Autonomic Manager [1]. An Autonomic Manager is responsible for managing particular resources via a control loop consisting of the following phases – Monitor, Analyze, Plan, and Execute (MAPE). Furthermore, the

M.L. Gavrilova, C.J.K. Tan (Eds.): Trans. on Comput. Sci. XXII, LNCS 8360, pp. 64–92, 2014.

FOCALE [14] [15] architecture introduced a two-layer hierarchy of control loops. The lower of these hierarchies is technology specific - analogue to the Element Management System in traditional Network Management. The higher hierarchy is concerned with the management of the overall system - similar to the traditional Network Management System. Additional initiatives trying to go for an Autonomic Management of future networks, IT infrastructures, and services include CASCADAS [13] with the concept of ACEs (Autonomic Communication Elements) [16], and ANEMA (Autonomic Network Management Architecture) [9][49]. ACEs were used in the CASCADAS project for the management of services (e.g. video streaming services) and implement a control loop, which is inspired by the one described in the initial IBM white paper for autonomic computing [1]. The intelligence of these control loops is given as executable plans. The ANEMA architecture provides a framework that uses utility functions in order to derive behavioral policies which can be executed during the operation of the network. All these approaches address differently the problem of synchronization and ensuring the stability of multiple parallel autonomic control loops. These synchronization and stability aspects constitute a vital issue, since in a complex environment or architecture involving several control-loops executing in parallel, there is an inherent challenge to ensure that the autonomic elements' behaviors are synchronized towards a common goal. This is required in order to avoid a situation whereby each autonomic agent is working towards its own goal, but the overall set of actions/policies degrades drastically the performance and dependability of the system. Such a situation could even result in unwanted oscillations and instabilities of the control loops, leading to a decrease in the provided quality of service.

Coming back to the existing options for achieving action synchronization within the above listed initiatives, a number of approaches can be easily identified. For example, since CASCADAS ACEs operate based on executable plans, it is straight possible for the autonomic system's developer to embed stability constraints in the resulting (collaborative) behavior(s). The utility function based approach for deriving policies of ANEMA suggests that the resulting behaviors will be intrinsically stable and conflict free during the operation of the network. In addition, [27] has investigated possibilities to design collaborative control loops such that they operate in a stable manner. This includes the application of game theory concepts during the design phase, and the use of model driven engineering techniques. The latter can be achieved through different tools or combinations of tools (tool chains), which are applied during the development phase of the overall set of autonomic entities, such that their control loops are intrinsically stable and synchronized by design. The set of applicable tools includes modeling environments such as GME [4], and EMF [35], simulation and verification tools such as UPPALL [36], and finally code generation tools. Examples for tool chains or standalone tools enabling the model driven specification and design of autonomic agents are given in [17] as well as in [18]. Thereby, the approach in [17] is based on the interplay between existing tools facilitated through a model sharing system called ModelBus, whilst [18] [19] [20] and [21] aim at defining specific modeling languages that enable the (stable) design and specification of self-managing entities. In addition, the synchronization between

control loops from traditional control theory was widely investigated. Examples of such research, based on game theory and conducted in the scope of cloud computing, is given by [44] [45] and [46]. Moreover, [5] introduces a hybrid approach to the synchronization of multiple self-organizing agents/entities. This hybrid approach is based on the use of so called *archetypes* which are architectural templates embedded inside the software agents in question. The archetypes constraint and control the runtime self-organizing behavior of the entities in a way that the agents work collaboratively and do not contradict. The hybrid nature is given by the fact that different archetypes are specified during the design phase and are dynamically instantiated according to the emerging situation. The authors of [48] have focused on integration patterns for components of autonomic management systems. These patterns semi-formalize the way existing autonomic agents can be brought to work together. Thereby, the integration patterns set the path towards conflict-free interaction of the control loops belonging to the autonomic components being integrated[1]. [31] and [33] propose a methodology to synchronize different independent autonomic control loops by introducing an additional "on top" layer. This additional layer contains a *coordination manager* utilizing finite-state-machines which are specifically developed in a way as to synchronize the actions resulting from the underlying control loops. [42] introduces a synchronization approach for multiple autonomic control loops based on the use of a common knowledge base. That way, independent control loops can share information about their operation and avoid conflicting situations. Finally, the current paper capitalizes on and extends another concept that has been initially presented as part of [27] and [26]. This is the concept of runtime action synchronization for multiple independent parallel running control loops - including policy actions and control theory type of loops. The novelty of this idea and progress beyond state of the art is that it allows, in a generic way, for autonomic agents, which were not intrinsically designed as to collaborate, to synchronize their tentative actions and work together towards improving the performance of the network or IT infrastructure in question. Based on a model regarding the impact of potential actions on selected key performance indicators (KPIs), as well as the importance of these KPIs, we propose and evaluate methods and techniques for the selection of an optimal subset of tentative actions, which are intended for execution by independent autonomic agents.

The proposed framework is complementary to self-organization approaches (e.g. self-organizing maps) where the autonomic agents collaboratively try to achieve an optimal set of reactions without referring to an arbiter for the sake of conflict resolution. We argue that in a real world environment, it is best to combine the framework presented here with traditional self-organization techniques embedded inside the autonomic agents in question.

In order to show the feasibility of the proposed concepts, our approach is validated based on a prototype that allows conducting a representative performance and scalability analysis.

[1] In that context, the approach towards action synchronization presented in this paper can be largely map to the *Hierarch pattern* of [48].

The rest of this paper is organized as follows: Section 2 introduces the architectural aspects of our proposed framework, and presents a case study with respect to how a specific architecture for autonomic networking can be extended by components implementing our approach. Section 3 formally defines the problem of runtime action synchronization and arrives at a mathematical model that allows for applying different types of algorithms for the purpose of ensuring the runtime stability of parallel autonomic control loops. Section 4 reformulates the problem in a way that it is more efficiently solvable, and presents a machine learning approach to obtaining some of the vital parameters for the resulting new model. Section 5 relates our proposed mechanisms to traditional concepts from the area of Control Theory. The following section describes some technical details and design decisions regarding our prototype implementation. Thereafter, based on the described prototype, section 7 describes the experimental setup and results related to the performance of the proposed techniques, in terms of synchronization quality and scalability. The next section presents a case study that demonstrates how the proposed techniques can be applied in the domain of autonomic networking. Finally, section 9 concludes the paper and outlines potential future research directions.

2 Architectural Aspects

This section elaborates on the architectural aspects with respect to the problem of runtime action synchronization.

2.1 Architectural Setup for Runtime Action Synchronization

We propose to introduce an entity that can be requested by other agents to allow or disallow tentative actions based on the goal of optimizing a set of key performance indicators. This results in an architectural setup as the one illustrated in figure 1. The entity denoted as Action Synchronization Engine (ASE) is requested by a number of independent agents - designed and implemented without intrinsic synchronization - to check their tentative actions for potential conflicts, and to inform them back on whether particular actions are allowed or disallowed for execution. Specifically, we consider plain actions (e.g. reset a network interface card) and policy enforcements - an *if(condition)-then(action)* - as (management) actions, since each of those items can be seen as an action (make/enforce) in the corresponding context. Indeed, the ASE is expected to act as an arbiter for negotiation that removes some overlapping and possible contradictions in the actions. Hence, an ASE component would allow only those management actions to proceed that are beneficiary for the overall fitness of the managed infrastructure.

Based on the architectural principle in figure 1, we propose two possible realizations of an ASE component. An ASE can be either put in place as a separate agent, i.e. as a standalone process solely responsible for the synchronization of autonomic control loops executed in parallel, or as a sub-component inside an agent that is primarily designed to manage some operational resources. In the latter case, the

ASE sub-component is started, orchestrated, and configured by the responsible autonomic entity. Clearly, an autonomic agent hosting an ASE sub-component may also refer to it for the sake of synchronization and conflict resolution regarding parallel control loops. This is exemplified in the next section, where we show how one of the emerging architectures for autonomic networking can be extended in a way as to realize action synchronization functionality for its control loops.

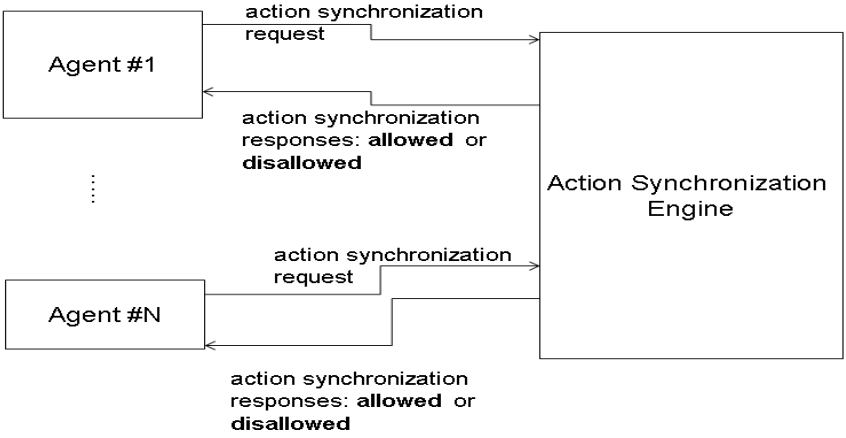

Fig. 1. Basic architectural Setup for Runtime Action Synchronization

2.2 Case Study: Ensuring Runtime Stability of Autonomic Control Loops in the Generic Autonomic Network Architecture (GANA)

In this section, a case study is presented that illustrates how ASE components can be deployed along a specific architecture for autonomic networking. First, we give an overview of the key features of the architecture selected for our case study, and then propose the necessary extensions.

For illustrative purposes, the Generic Autonomic Network Architecture (GANA) [3] has been selected, since it tries to create a complete view - of the node and the network as a whole - with respect to autonomic self-management of future networks. In addition GANA is currently being worked on at ETSI as to mature towards a reference model for Autonomic Networking [47]. In addition, an initial implementation of the GANA architecture was provided within the EFIPSANS project [6].

GANA defines generic autonomic elements for each required networking function, e.g. routing, forwarding, mobility etc. The core concept in GANA is that of a generic autonomic entity denoted as Decision Element (DE). A Decision Element (DE) executes the logic of a control loop using the management interfaces of its assigned Managed Entities (MEs). That is, a DE is responsible for autonomically regulating the parameters of concretely assigned MEs, and realizes a control loop based on the information it acquires directly from the MEs or from other sources such as embedded

network monitoring processes. The information is analyzed, subsequently a decision is made, and an action is executed on the MEs in order to dynamically (re)-configure and regulate their behavior, while striving to achieve a predefined goal. Taking into account that control loops on different levels of functionality are possible, GANA defines the Hierarchical Control Loops (HCLs) framework. In the context of the HCLs framework, four levels exist at which generic Decision Elements and associated control-loops can be designed: (1) protocol-level – autonomic mechanisms within the network protocols, e.g. control loops in OSPF or TCP, (2) functions-level – autonomic control loops responsible for a specific network function, e.g. routing, forwarding, mobility management, (3) node-level - a device as a whole is also considered as a level at which autonomic functions considering the overall node can be implemented, (4) network level – autonomic functions which are executed network wide. Thereby, Decision Elements and corresponding control loops on a higher level manage DEs on a lower level down to the lowest-level MEs i.e. protocols and fundamental mechanisms. Therefore, DEs are designed following the principles of hierarchical, peering, and sibling relationships among each other. These relationships are realized within a node or among the nodes enabling DEs to realize both distributed and centralized control loops. For detailed information regarding the presented concepts, the reader is referred to [3].

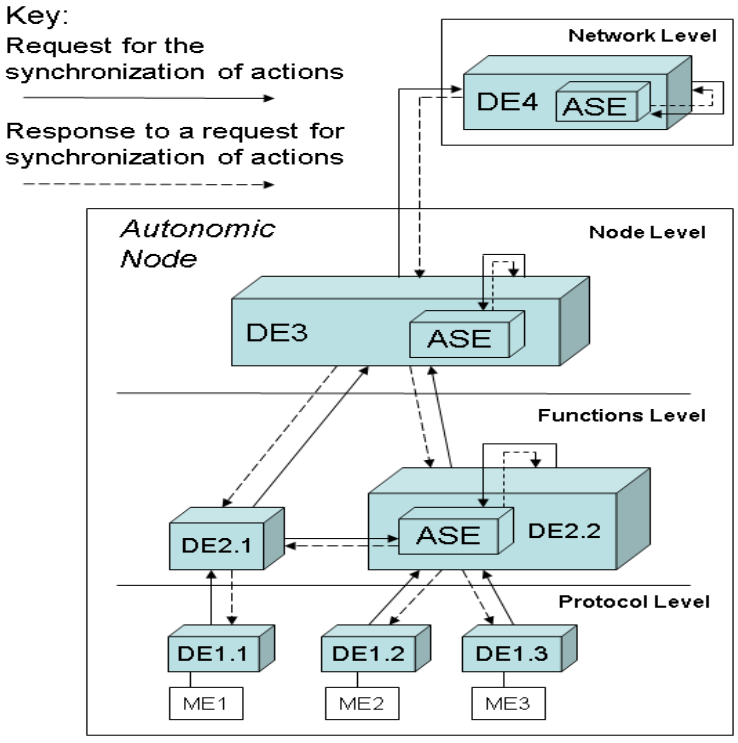

Fig. 2. The extended GANA Architecture

In order to elegantly extend GANA and its generic DEs following the rules defined by HCLs, we propose that ASE components should be embedded inside the corresponding DEs. That is, ASE sub-components should be initialized and started by some of the DEs in order to achieve a runtime configuration as the one depicted in figure 2. An ASE is considered as part of a dedicated generic GANA DE that has been elected or is by design the most appropriate one for acting as an arbiter enabling the negotiation over tentative actions. In that context, the DEs in question can refer to the belonging ASE by using the hierarchical, peering or sibling relations defined within HCLs. Therefore, we require that every DE should keep a list (catalogue) of the actions it is allowed to issue without having to consult an upper level or a sibling DE. As illustrated in figure 2, if a DE (e.g. DE1.1, DE1.2 or DE1.3) faces a problem that is beyond its local scope, i.e. the action to be issued as a response to some challenging conditions is not inside the aforementioned catalogue, it should consult its upper level DE (DE2.1 or DE2.2). The upper level DE should in turn consult the corresponding ASE (hosted by DE2.2) that is expected to resolve potential conflicts and to respond back with a set of actions that are allowed to be executed. After the optimal actions have been selected, the upper level DE informs the lower level DEs in question, whether they are allowed to "fire" some of the actions on their corresponding Managed Entities (ME1, ME2, and ME3). On the other hand, if the upper level DE recognizes that the actions it has been requested to synchronize are beyond its competence, it should further consult its corresponding upper DE (e.g. DE3 or even DE4 in figure 2) on the higher level.

3 The Action Synchronization Problem

This section formally defines the runtime Action Synchronization Problem (ASP). We start with a short introduction and overview of similar efforts in the past years and continue with deriving a model that allows to mathematically reason about the optimal subset of tentative actions requested for synchronization. This model consists of a set of mathematical objects, such as a utility function to optimize, and a set of constraints represented by corresponding matrices and vectors.

3.1 Introduction

In a complex multi-agent environment or architecture involving more than one autonomic control loop that need to execute in parallel, the challenge is to ensure that the autonomic entities' behaviors are synchronized towards a common goal. That is required in order to avoid a situation whereby each autonomic entity is working towards its own goal but the overall set of actions/policies degrades the performance and dependability of the system in question. In order to achieve the aforementioned common goal, we present an approach based on the concept of a utility function that incorporates the state of the network/system, and must be optimized by selecting the optimal subset of tentative actions, resulting from decisions made by the autonomic entities.

Research related to the application of utility functions in the area of Autonomic Computing/Networking has been ongoing since the release of the IBM autonomic computing white paper [1]. The idea of Autonomic Management based on goals, for

which utility functions are a way to express, was initially investigated in [12]. [10] is one of the pioneer publications discussing the role of utility functions in self-management. It elaborates on the different classes of policies that can be applied in order to optimize the corresponding utility function and presents a case study carried out in the context of a commercial service management framework. [11] describes a two-level utility function based architecture for monitoring and actively optimizing the performance of an application server. However, the application of the utility functions is mainly embedded inside the autonomic elements and there is no additional arbiter component that handles synchronization requests from autonomic elements, and chooses to allow or disallow specific tentative actions.

3.2 Deriving the ASP Problem

Based on ideas from the theory of Optimal Control [38], we define an optimization problem that should be solved by an ASE agent whenever a set of actions needs to be synchronized. Let Q be a set of performance metrics/KPIs, and $|Q|=n\in N^+$. Let W be a set of weights, each of which is assigned to one of the metrics contained in Q and $|W|= n\in N^+$. These weights represent the importance of a single KPI for the overall health of the system, and thus they all should be positive, i.e. $w_i\geq 0$ for $w_i\in W$, $w_i\in R$ and $i\in\{1, \ldots, n\}$.

Considering a particular point in time t_0, the values of the KPIs in Q are represented by a vector $Q(t_0)=(q_1(t_0), \ldots, q_n(t_0))^T\in R^n$. Suppose that, the higher the values of $q_1(t_0), \ldots, q_n(t_0)$ the better the performance of the system (for KPIs which require to be minimized one can take the reciprocal value), then the following expression gives the system fitness at t_0.

$$SF(t_0) = \sum_{i=0}^{n} w_i q_i(t_0) = \langle w, Q(t_0) \rangle \tag{1}$$

Hence, the goal of the autonomic mechanisms in the node/device and the network is to maximize $SF(t)$ throughout the operation of the system.

Additionally, let A be the set of possible actions that can be potentially issued by the involved autonomic agents, and $|A|=M\in N^+$. By $a_j\in A$ with $j\in\{1, \ldots, M\}$ we denote a single action. Further, the domain relation of an action $d:A\rightarrow\{0,1\}^n$ is introduced. The relation takes as an input an action and returns a (0-1) vector, which contains mappings to the metrics the input action influences if executed. Indeed, if the i^{th} component of the vector is 1, then the i^{th} metric is influenced by the input action, and respectively if the latter is 0, then the metric is correspondingly not influenced. Putting together the domain relation outputs for all actions as columns in matrix form results in what we denote as the domain matrix of A.

$$D = (d(a_1), \ldots, d(a_M)) \in \{1,0\}^{n\times M} \tag{2}$$

The last and most important ingredient for formulating the optimization problem is the impact relation of $A - I:QxA\rightarrow R$. The output value of $I(i,j)$ stands for the impact the j^{th} action has on the i^{th} KPI. Thus, if only action a_j is executed at point in time t_0, then the new value of q_i will be $q_i(t_0)+I(i,j)$.

Based on the above definitions, and assuming that in a particular time slice a total number of $m \in N^+$ $(m \leq M)$ actions have been requested for synchronization, the following optimization problem is defined.

$$\max_{p \in \{1,0\}^n} \sum_{i=0}^{n} w_i (q_i(t_0) + \sum_{j=0}^{m} p_j I(i,j)), \; s.t. \; D_m p \leq c \tag{3}$$

In (3), p stands for a (0-1) vector, which gives whether a particular action was allowed to execute or not. Indeed, if the j^{th} position of p is 1, then the corresponding action has been selected for execution; otherwise it has to be dropped. Hence, an optimization with respect to p is equivalent to selecting the optimal set of actions requested in a particular time frame. Moreover, the matrix D_m is a sub-matrix of the domain matrix D of A consisting only of the columns representing the domains of the requested actions. The vector $c \in N^n$ determines the extent, to which actions with overlapping domains are allowed. For example, if $n=4$ and $c=(1,1,1,1)^T$, i.e. only four KPIs are considered, then the additional constraint says that only one action influencing a single metric is allowed. In the case of $c=(2,1,1,1)^T$ two actions are allowed that influence the first metric. Hence, the additional constraint can be used to enforce the resolution of conflicts between actions with overlapping domain regions. Rewriting (3) in vector-matrix form results in

$$\max_{p \in \{1,0\}^n} w^T I_m p + w^T Q(t_0), \; s.t. \; D_m p \leq c \tag{4}$$

where I_m stands for an $n \; x \; m$ real-valued matrix that contains the impact of the requested actions on the considered KPIs. Thus, $I_m(i,j)$ represents the impact of the requested j^{th} action on the i^{th} metric.

Since the term $w^T Q(t_0)$ in (4) is just a constant that reflects the current state of the network, it can be dropped, which is very good news because it means that the values of the KPIs are not needed for the optimization. That fact reduces the overhead produced by the ASE, since no interaction with monitoring functions measuring the KPIs is needed. Hence, the final optimization problem takes the following form:

$$\max_{p \in \{1,0\}^n} w^T I_m p \; , s.t. \; D_m p \leq c \tag{5}$$

The above optimization problem can be interpreted as *"selecting the most appropriate subset of tentative actions such that the change in the state of the system is positively maximized"*.

3.3 On the Complexity of ASP

This subsection investigates around the complexity and the challenges that are expected while solving instances of ASP. First of all, ASP is a special case of integer/binary programming which potentially classifies it as an NP-complete problem, i.e. a hard to tackle problem which can be only solved by a non-deterministic polynomial-time Turing machine. More specifically, ASP corresponds to

a special instance of the thoroughly studied 0-1 multi-constraint knapsack problem (MKP) [30]. MKP is defined as follows:

$$\max_{x_i \in \{1,0\}} \sum_{i=0}^{n} g_i x_i \ , \ s.t. \ \sum_{j=0}^{n} v_{ij} x_j \leq C_i \tag{6}$$

with gains (profits) $g_i \geq 0$, volumes (or weights) $v_{ij} \geq 0$ and capacities $C_i \geq 0$ with respect to the overall volume (or weight) for each knapsack. Thereby, $g_i \in R$, $v_{ij} \in R$, and $C_i \in R$. The difference between ASP and MKP is constituted by the ranges for v_{ij} and C_i, and correspondingly d_{ij} - the elements of D_m, as well as c_i - the elements of the constraint vector c. d_{ij} can take only binary values, and c_i can take only positive integer values according to the definition of ASP. Hence, each ASP instance is an instance of MKP. MKP has drawn the attention of the research community for years due to its wide field of application (e.g. resource and budget planning). It is known to be an NP-hard problem [30] with exact algorithms existing for some special cases, as for example described in [32] and [34]. However, to our best knowledge, no exact algorithm exists for high dimensional special instances as constituted by ASP.

These considerations show that finding a solution for the runtime action synchronization problem is not an easy task. Therefore, different heuristics and approximation algorithms can be considered. For example, modern solvers such as GLPK [2] and Coin-OR [8] are quite advanced and would provide a solution that is the best possible based on the underlying optimization algorithm, e.g. branch-and-bound, simulated annealing, tabu search, etc. Moreover, in the next section we propose an approach that allows to partially overcome the obstacles arising due to ASP's computational complexity.

4 Machine Learning Approach to Action Synchronization

In this section, an approach to runtime action synchronization is proposed that allows overcoming the obstacles of inconvenient problem complexity identified in the previous sections. We reformulate the ASP problem and provide a machine learning methodology to handling the uncertain parameters resulting from the reformulation.

4.1 RASP: Relaxation of the ASP Binary Optimization

We start with the previously derived optimization problem, and relax the binary constraints such that the resulting problem belongs to the complexity class *P*, i.e. to the problems solvable in polynomial time. That is, we turn the condition $p_i \in \{0,1\}$ into an inequality: $0 \leq p_i \leq 1$, for $i \in \{1, \ldots, m\}$. Hence, the new optimization problem is given by:

$$\max_{p} \ w^T I_m p \ , \ s.t. \ 0 \leq p_i \leq 1 \ \text{ and } \ D_m p \leq c \tag{7}$$

This optimization problem is a linear program which constitutes a convex optimization problem. Hence, there is only one optimum and every local optimal solution is also a global one, which means that the diverse optimization techniques will always improve iteratively the quality of the obtained solution. The above

formulation constitutes a problem belonging to the complexity class *P*, i.e. efficient polynomial algorithms exist for solving this problem. The result of this optimization is a vector containing values from the interval *[0,1]*. If i^{th} component of this vector is 0 then the corresponding action is disallowed. Correspondingly, if it is 1 then the action should be issued. If an agent, requesting for synchronization, receives as response a value from the interval *(0,1)*, then it can either execute or drop the action, based on an internal threshold θ_i. These thresholds can be supplied by the human experts tweaking the autonomic network. For instance, the history of requests for synchronization can be analyzed offline (e.g. by employing statistical and/or machine learning methods) and appropriate thresholds can be extracted using some optimization tools. In the following sub-section such a methodology is proposed and elaborated in detail.

4.2 Methodology for Obtaining Threshold Parameters for RASP

Assuming that during the operation of a particular system, a history of action synchronization requests was collected $r:=\{r_1, \ldots, r_{tr}\}$, $tr \in N^+$, a methodology is proposed how to obtain thresholds $\{\theta_1, \ldots, \theta_M\}$ such that the involved agents are able to decide whether to execute an action or not based on the solutions of RASP obtained by the corresponding ASE component. One can assume that as long as training data is being collected, the system operates based on ASP. The elements of *r* are *(0-1)* vectors, where a 1 is set in case the action at the corresponding position in the vector was requested for synchronization within the particular request, and a 0 in case the action was not requested. The results of the RASP optimization based on the requests in *r* are then given by $X=\{X_1, \ldots, X_{tr}\}$, $tr \in N^+$ whereby $X_i \in [0,1]^M$. By defining and solving an optimization problem based on the training data in *X*, we would like to obtain the thresholds $\{\theta_1, \ldots, \theta_M\}$ which are in turn given to the requesting agents and used in the course of runtime action synchronization. This leads to the following maximization objective:

$$\max_{\theta \in [0,1]^n} \sum_{i=1}^{tr} w^T I(X_i - \theta) \tag{8}$$

(8) can be explained as follows: the distances between the threshold values and the RASP results should be optimized in a way that the impact (reflected by these distances) of the corresponding actions is positively maximized. (8) can be further reformulated as follows:

$$\begin{aligned} &\max_{\theta \in [0,1]^n} \sum_{i=1}^{tr} w^T I(X_i - \theta) = \\ &= \max_{\theta \in [0,1]^n} \left[\sum_{i=1}^{tr} w^T I X_i - \sum_{i=1}^{tr} w^T I \theta \right] \\ &\Leftrightarrow \min_{\theta \in [0,1]^n} \sum_{i=1}^{tr} w^T I \theta \Leftrightarrow \min_{\theta \in [0,1]^n} w^T I \theta \end{aligned} \tag{9}$$

The final result in (9) is based on the fact that the first sum in the rearranged maximization is a constant, because it is computed only based on the available training data.

In order to preserve the constraints given in the ASP definition (5), the following constraint is added

$$\sum_{j \in \mathrm{Im}(q_k)} 1_{\{X_{ij} \geq \theta_i\}} \leq c_k \tag{10}$$

with $Im(q_k)$ being the set of actions that impact KPI q_k, and c_k being the bounding value for k as given in the "subject to" part of (5). Combining (9) and (10) results in the following non-linear optimization problem:

$$\min_{\theta \in [0,1]^n} w^T I\theta \text{ , } s.t. \sum_{j \in \mathrm{Im}(q_k)} 1_{\{X_{ij} \geq \theta_i\}} \leq c_k \tag{11}$$

As a further step towards increasing the solvability of (11), one might consider to treat the constraint in a way that it becomes differentiable. That way, optimization algorithms can be applied that explicitly make use of the gradient. A possible direction to go is applying a linear function on both sides of the constraint, since a linear operator would preserve the inequality. A good candidate is the expected value *E[.]* operator, which would also mean that the constraint is relaxed by ensuring that on average the selected subset of actions does not violate the constraint vector. This yields the following:

$$\begin{aligned} & E\left[\sum_{j \in \mathrm{Im}(q_k)} 1_{\{X_{ij} \geq \theta_i\}}\right] \leq E[c_k] \\ \Leftrightarrow & \sum_{j \in \mathrm{Im}(q_k)} P(X_{*j} \geq \theta_i) \leq c_k \\ \Leftrightarrow & \sum_{j \in \mathrm{Im}(q_k)} (1 - P(\theta_i \geq X_{*j})) \leq c_k \end{aligned} \tag{12}$$

Thereby *P(.)* on the last line stands for the cumulative distribution of X_{*j}, i.e. the distribution of RASP results for the j^{th} action. Thus, in (12), *P(.)* gives the probability for θ_i to be greater than X_{*j}. Thereby *P(.)*, in terms of a probability model, must be a distribution over the set *[0,1]*. Furthermore, a distribution of X_{*j} can be computed out of the trainings data, e.g. by calculating the maximum likelihood estimators for the targeted model distribution. Combining (9) and (12) results in the following optimization problem:

$$\min_{\theta \in [0,1]^n} w^T I\theta \text{ , } s.t. \sum_{j \in \mathrm{Im}(q_k)} (1 - P(\theta_i \geq X_{*j})) \leq c_k \tag{13}$$

The *beta distribution* [41] is a good candidate for *P(.)*, since it generalizes different possible distributions over the *[0,1]* interval. The cumulative distribution function (CDF) of the beta distribution, adapted to the current context for a threshold θ_i, is given by:

$$\mathrm{B}_{X_{ij}}(\theta_i, p_i, q_i) = \frac{\int_0^{\theta_i} t^{p_i}(1-t)^{q_i - 1} dt}{\int_0^1 t^{p_i}(1-t)^{q_i - 1} dt}, with\ p_i, q_i > 0 \tag{14}$$

(14) has two parameters p_i and q_i, which can be estimated from the training data by using the method of moments [39]. The gradient of (14) can be also easily calculated by taking into account that the derivative of a CDF is given by the corresponding PDF (probability distribution function). This gradient can be used to improve the quality of the obtained thresholds based on algorithms, which can explicitly make use of it for the sake of gradient based optimization.

In the following sections we compare the quality of action synchronization by solving directly the binary optimization problem in (5), and by solving the relaxation based on threshold estimations provided by (11), and by (13) thereby applying (14) as a relevant probability measure. We denote the combination between (13) and (14) as the *beta-distribution approach*. Next, we shortly relate our approach to traditional control theory, and present a prototype implementation, based on which the experiments are conducted.

5 Relating Runtime Action Synchronization to Traditional Control Theory

The ASP problem derived in the previous sections aims at ensuring the stability of an autonomic system implementing multiple control loops. Thereby, the control loops may be based on generic models defined for autonomic computing and networking in the past years (e.g. IBM MAPE [1], FOCALE [14], or GANA [3]) or on control loops as specified within the area of traditional control theory [38]. The latter type of control loops are based on specific transfer functions, and on the state of the system under control in case of feedback loops like the PID (Proportional-Integral-Derivative) controller. Transfer functions can be seen as mathematical artifacts, which describe the required regulative mechanisms based on different parameters such as time or system feedback. Fundamental properties, which the control loop designer tries to achieve by adjusting the parameters of the belonging transfer functions, are the so called SASO properties – Stability, Accuracy, Settling Time and Overshoot [38]. Stability in that case is considered mainly with the oscillations that could be potentially caused by an improperly selected transfer function parameters. Accuracy deals with the degree to which particular desired values of the regulated parameters are achieved. The settling time property can be seen as an indicative for *"how long it takes"* until the effects of the regulative behavior are achieved. Finally, Overshoot gives the maximum deviation (from the desired value) which results from the regulative action until the belonging Accuracy is achieved. Within our framework, we would expect that the underlying control loops are designed considering the SASO properties. The KPI impact of the actions, which result from these control loops, should be modeled as considered after the corresponding settling time. That is, we presume that the final effect of the action is taken into account when modeling the multi-agent environment with the corresponding actions and impact on KPIs. This way the interplay between our approach and traditional control loops is facilitated.

6 Implementation of an Action Synchronization Engine

In order to analyze the technical feasibility of our approach with respect to functionality, scalability and overhead produced by solving the previously derived

optimization problems in real time, we implemented an ASE component, which can be started and configured by any "hosting" agent (see section 2), or can operate as an autonomous entity in a multi-agent networked environment. Additionally, an API was defined and developed that can be used by "client" agents willing to synchronize tentative actions resulting from their intrinsic control loops. The implementation was conducted on a Linux platform using C/C++ and POSIX threads. The "client" agents use a specially implemented library for communicating with the corresponding ASE.

The first issue that has to be tackled is related to the solver required for solving the ASP and RASP optimization problems. Research done in the field of electricity spot market optimization problems [7] has compared different open-source solvers for linear and mixed integer programs and gives hints that the SYMPHONY solver (for solving ASP) and the CLP solver (for solving RASP), which are both part of the Coin-OR project [8], perform best. Thus, the ASE implementation was built around the solvers provided by the Coin-OR project. Moreover, the internal architecture of an ASE component consists of two modules – one that takes care of reading and interpreting the configuration data passed to the ASE, and a second one that manages the threads serving the "clients". The configuration data required by an ASE component includes: 1) a file containing the *impact matrix*, 2) a file containing the weights corresponding to the KPIs' significance, 3) a file containing the *constraints* vector that restricts the number of actions that influence overlapping sets of KPIs, 4) the *time interval* after which the solving of the optimization problem is automatically triggered independently of the number of requests, 5) a *maximum number of requests* that automatically triggers the solving of the optimization problem, and 6) a flag indicating whether ASP or RASP should be solved by the ASE entity in order to perform runtime action synchronization. All these parameters need to be supplied to the ASE agent by the time it gets started, and can be provided by a human expert.

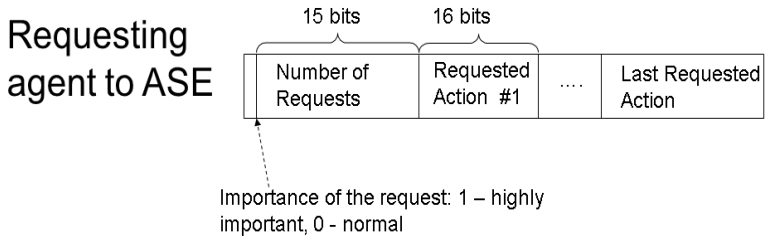

Fig. 3. A Message carrying an Action Synchronization Request

After the module parsing the parameters has completed the configuration of the ASE entity, a main "server" thread is started that listens for, and accepts "client" connections. In parallel a periodical thread is started that "wakes up" up every X seconds (corresponds to the fourth configuration parameter), copies all requests for synchronization, prepares the requests, the impact matrix, the constraints vector etc. in the form required by the native interface of the solver in use, and finally invokes it in order to obtain a solution. Consequently, the relevant parts of the solution are communicated back to the corresponding requesting agents. Apart from the periodical task, every time a connection gets accepted by the ASE component, the requests are

stored in a common (for all threads inside the ASE process) registry. In cases that the request was submitted and marked as *highly important* or the *maximum number of requests* after which synchronization is automatically triggered has been reached, the solving of the optimization is invoked, taking into account all synchronization requests stored in the common registry since the last time optimization was triggered. In order to synchronize all the concurrent threads around the common registry and the solving process, POSIX mutexes are utilized.

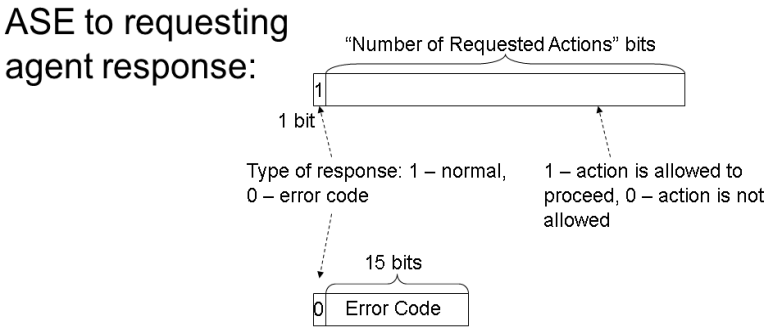

Fig. 4. Response as a Result of an ASP based action Synchronization Request

The communication between the requesting entities and an ASE component requires the specification of a message format for the exchange of requests and responses in case of an ASP or RASP solving ASE. These messages can be exchanged over Unix Domain Sockets inside a device, or over specially designed protocols for control information exchange between nodes, e.g. ICMPv6. Figure 3 presents the format of an action synchronization request message as implemented in our prototype. The first bit is used to indicate the urgency of the request. The following 15 bits are used to encode the number of requested actions, followed by a set of 16 bit integers representing the requested actions.

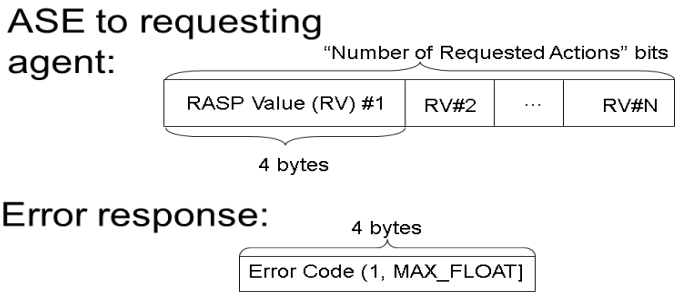

Fig. 5. Response as a Result of a RASP based Action Synchronization

Depending on whether the ASE entity in question is pre-configured to perform an ASP or a RASP optimization, different response formats are required. Figure 4 illustrates the message carrying the response from an ASP based action synchronization. The first bit of the message indicates whether the synchronization was successful or not. In case of a failed synchronization, different error codes can be

reported within the next 15 bits. In case of a success, a corresponding number of bits is conveyed which reflect the request for action synchronization thereby indicating whether a tentative action is allowed (bit value 1) or disallowed (bit value 0). Additionally, figure 5 depicts the response message in case of RASP synchronization. It consists of a sequence of thresholds each encoded in 4 bytes. These are meant to be floating point numbers in the range between 0 and 1. In case of failed action synchronization, an error code is returned that indicates the type of problem occurred within the ASE component.

Based on this prototype, the next sections present different numerical evaluations of our approach as well as a case study illustrating its application.

7 Experimental Results

In order to evaluate our approach, we designed a set of experiments such that we can compare the quality of the action synchronization achieved by the different mechanisms and techniques presented so far. Since there are no such systems deployed in practice, we simulated requests and models (including weight vectors, impact matrices, and constraint vectors) of different sizes for our experiments. This allowed us to gain an impression of the performance of the different techniques on a number of models and combinations of tentative actions.

7.1 Qualitative Measurements

The measurements presented in this section are meant to benchmark the quality of the action synchronization procedure, in terms of achieved utility value and constraint satisfaction as determined by the corresponding constraint vector. This utility value is obtained by solving ASP directly, or by solving the corresponding relaxation – RASP, and applying thresholds for accepting for execution or dropping actions.

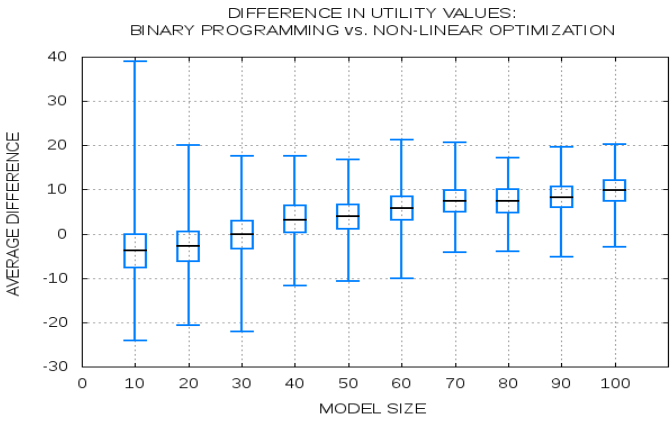

Fig. 6. The Average Difference between the Utility Values obtained by solving RASP combined with applying Thresholds based on optimizing (11), and directly solving the ASP Synchronization

We used models with equal numbers of KPIs and potential tentative actions. For each model size, we took 1000 different model instances. Additionally, 1000 action synchronization requests were simulated as training data for each model size, in order to evaluate the machine learning approach from section 4. Moreover, 100 action synchronization requests were used as test data for each model size, upon which we validated the quality of the different techniques. We developed a Matlab script that implements the machine learning procedures as specified by (11), as well as by (13) with applying the beta-distribution (14) as a probability measure. In the course of this, we made use of the NLOPT [22] library for non-linear optimization. Thereby, we employed first an "Improved Stochastic Ranking Evolution Strategy" [23] to perform a global search for both problems defined in (11) and (13). Afterwards, we conducted a local search based on the "Constrained Optimization by Linear Approximations" [24] algorithm for (11), and on the "Augmented Lagrangian algorithm" [25] for (13), in order to find a precise solution locally. Thereby, the algorithm applied for (13) can explicitly benefit from the derivatives of the utility function and the constraints, which can be computed based on the beta-distribution (14) function. These steps resulted in thresholds obtained based on (11) and (or) (13) for each model instance. The belonging test data was used to evaluate the quality of these thresholds in terms of accepting or dropping an action based on a solution for RASP (7). That is, we first solved RASP, and using the corresponding set of thresholds, we accepted or dropped actions based on the obtained RASP solution vector. The selected actions resulted in a particular value of the utility function, and potentially violated the constraints vector, since the relaxations influence directly the involved constraints. For that reason, on one hand, it is worth to compare the difference between the utility value achieved by using thresholds and the one obtained by solving ASP directly. On the other hand, it is required to measure the magnitude of constraint violation achieved by solving RASP and applying the thresholds. At this point, it is worth mentioning that straight solving the binary program (5) does not lead to any constraint violations, since this is the original version of the optimization problem.

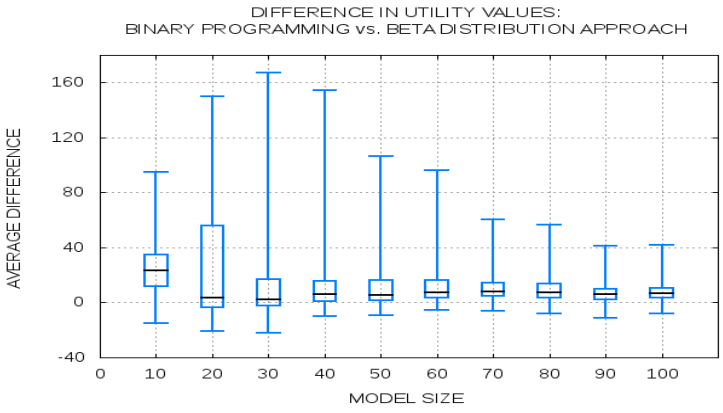

Fig. 7. The Average Difference between the Utility Values obtained by optimizing RASP combined with applying Thresholds based on the Beta-distribution Approach, and solving directly the ASP Problem

Figure 6 and figure 7 illustrate the empirical distributions of the differences between the averaged utility values of the involved approaches for each model size. The averaged utility values were obtained based on the belonging test dataset. That is, we averaged the utility values obtained for each action request in the corresponding test data, and present the empirical distribution of the difference between these averaged utility values for each of the 1000 model instances of the size in question. By difference we mean *"averaged utility value of RASP and (11)" - "averaged utility of ASP"* for figure 6, and *"averaged utility value of RASP and (13)" - "averaged utility of ASP"* for figure 7. Indeed, a positive value within the empirical distribution means that the corresponding RASP based technique performed better than straight binary optimization. The box plots visualize the median, the 25% quantile, the 75% quantile, the minimum and maximum value of the experienced sample. Figure 6 shows that solving RASP, and applying thresholds obtained by (11), performed slightly worse than binary optimization of ASP for smaller model instances. However, with growing model sizes, the RASP based method started performing better even though there were still some outliers (in terms of models) for which binary optimization is better. On the other hand, figure 7 shows that solving RASP, and applying thresholds obtained through the beta-distribution approach, resulted in better utility values for all evaluated model sizes. Again, there were some outliers (in terms of models) by which the binary optimization seems to be the better approach to obtaining optimal system fitness.

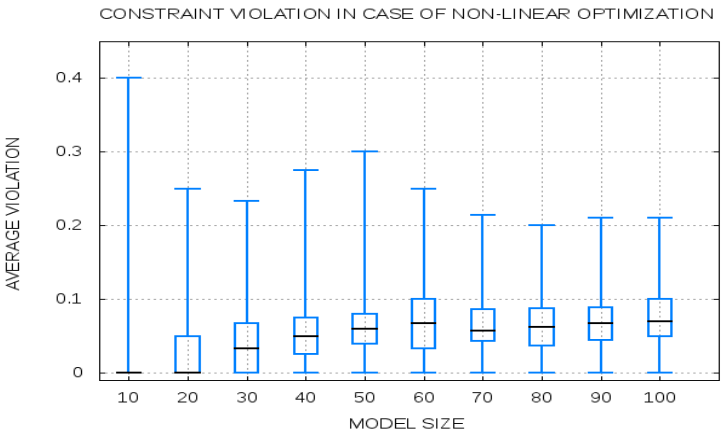

Fig. 8. The Average Violation in case of optimizing RASP and applying Thresholds obtained through (11)

The average numbers of constraint violations while performing RASP based optimization are shown in figure 8 and figure 9. The average value was built based on the test data set for each model instance. Figure 8 and figure 9 clearly indicate that even tough the violations in the case of the beta-distribution approach were minor, the non-linear threshold optimization (11) clearly outperformed the beta approach when it comes to constraint violations. This is not surprising, since (11) reflects to a large extent the initial ASP problem regarding the constraints, while the probability based approach (13) relaxes them by assuring compliance only on average.

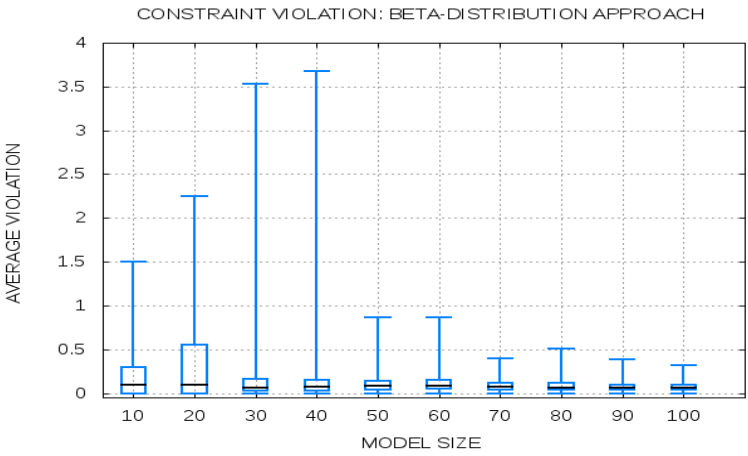

Fig. 9. The Average Violation in case of optimizing RASP and applying Thresholds obtained through the Beta-distribution based Approach

In general, we see that the two discussed RASP based techniques performed reasonably as compared to straight ASP binary optimization when it comes to achieved system fitness, and to satisfying the original constraints, which influence the simultaneous execution of actions.

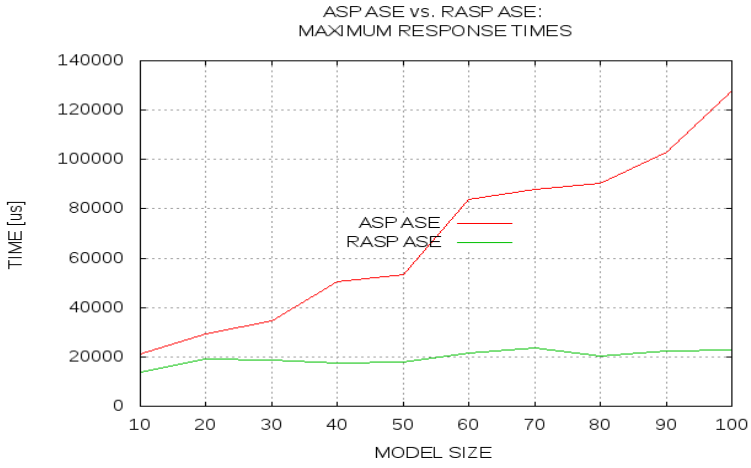

Fig. 10. Maximum Response Time of an ASE Component when solving ASP and when solving RASP

7.2 Scalability and Overhead Measurements

Since ASE entities are meant to facilitate self-management for a large diversity of systems, including network switches and routers, embedded systems, multimedia gateways, and server systems, the issue of scalability and overhead produced by such an agent is of paramount importance. Therefore, we use the previously described prototype to conduct a number of performance and overhead measurements of our proposed approach. In the course of that, we would also like to evaluate the performance characteristics related to the proposed optimization problems, i.e. ASP and the corresponding relaxation RASP.

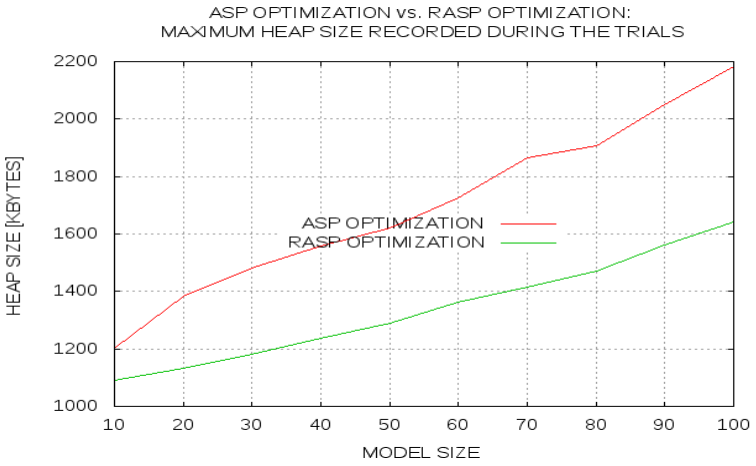

Fig. 11. Memory Consumption of an ASE Component when solving ASP and when solving RASP

The simulations were conducted on a Linux machine with the following hardware parameters: *Intel Pentium 4, 3.00 GHz, RAM 2 GB*. We simulated two requesting agents which were simultaneously issuing a total number of 100 action synchronization requests to an ASE entity. The size of the requests was set to 20 actions requiring synchronization. The last of the requests was sent as *highly important* (see figure 3), such that synchronization is issued immediately for the overall set of actions submitted by both agents. This setup allows also to judge on the scalability of the approach in case a large number of control loops need to be synchronized, since it pushes to the extreme the size of the models and the number of actions to synchronize. The only aspect that is not covered is given by the capability of the underlying hardware and operating system to serve a large number of connections and threads. Potential issues there can be remediated by additional hardware or by increasing the use of self-organization techniques (e.g. self-organizing maps) among the autonomic agents in question.

The maximum of the response times for the two requesting entities and the model size in question is plotted in figure 10. The trend clearly shows that a RASP solving ASE is the better choice in situations where fast responses are vital. Additionally, figure 11 compares the heap memory consumption of an ASP and a RASP solving ASE that can be seen as indicative for the amount of memory consumed while solving the underlying optimization problems. The trend in figure 11 indicates that in case memory is scarce resource, a RASP based approach to action synchronization would be the most suitable one.

8 Case Study: Autonomic Fault-Management and Resilience in Self-managing Networks

Our illustrative case study comes from the domain of autonomic networking and deals with the functions of Autonomic Fault-Management (AFM) and Resilience in self-managing IP networks. Autonomic Fault-Management is understood [28] as an automation of the tasks comprising traditional Fault-Management as indicated by the ITU-T TMN [29] (Telecommunication Management Network) standard. These tasks are: Fault-Detection – *"detect the presence of fault"*, Fault-Isolation - *"find the root cause for the observed faulty condition"*, and finally Fault-Removal - *"remove the identified root cause"*. On the other hand, the resilience of the network depends also on an immediate reaction to an observed faulty condition. This means that an action is required in order to mask the erroneous state until the Autonomic Fault-Management mechanisms have managed to remove the underlying root cause(s). Such an immediate action could be for example the result of a regulative mechanism based on a transfer function as implemented in traditional control theory [38]. Architecturally, these aspects have been addressed in publications such as [43]. The idea is to have two different autonomic entities: one implementing the immediate reaction based on a policy model – resilience agent, and a second one realizing the Autonomic Fault-Management tasks – AFM agent. These entities are introduced in each node of the network. Indeed, when it comes to executing the resulting actions, the two entities need to negotiate in order to resolve potential conflicts due to multiple AFM or fault-masking processes (threads) running in parallel. This negotiation is facilitated by an Action Synchronization Engine within each device. Hence, the case study shows how an ASE can be used to synchronize the tentative actions of autonomic entities within an architecture for Autonomic Fault-Management and Resilience as the one presented in [43].

Figure 12 presents the reference network for our case study. Each of the routers (R1 to R5) is equipped with a resilience agent, an AFM agent, and an ASE agent. Furthermore, each of the routers is running an OSPF routing daemon, e.g. as the one provided by the Quagga routing platform. We focus on potential faulty conditions around R2, and on potential reactions to these faulty conditions issued by the autonomic entities on R2. However, presenting the models and mechanisms that drive the reactions of the autonomic entities is beyond the scope of this paper. For more information, we refer the reader to [37]. In figure 12, R2 is a critical point in the

network, since it is a router having different (Gigabit and Fast Ethernet) types of links with different characteristics, as for example MTU (Maximum Transmission Unit) size – 1500 bytes for Fast Ethernet and maximum 9000 bytes for Gigabit Ethernet links. This creates the potential for black hole problems as described in [RFC2923]. These problems are made possible by the suppression of ICMP messages on the router in question having links with different MTUs attached. This is especially critical in IPv6 networks where there is no packet fragmentation on intermediate routers. On the other hand, the (dynamic) suppression of ICMP messages is required in case the network is under attack, since ICMP responses were often used to realize flooding and Deny of Service (DOS) attacks in the past. Hence, it is a good idea to also dynamically regulate the level of ICMP suppression as a reaction to an anticipated attack on the network infrastructure. Thus, the following potential actions, to be issued on R2 by the AFM agent, are introduced: *set-icmp-suppression-(low/medium/high)*. These actions are expected to differ with regard to the subsets of ICMP message types being suppressed on R2. Furthermore, the problem of link flapping can potentially occur on any of the routers including R2. It is constituted by a link that is continuously going up and down thereby affecting the OSPF routing in the network [40], such that the routers have difficulties to converge with respect to available routes. An immediate reaction of the resilience agent to this phenomenon could be given by adjusting the rate of the OSPF Hello timer on the affected routers [40]. Thus, the following potential actions, to be issued on R2 by the resilience agent, are considered: *set-hello-rate-(low/medium/high)*. As a result of the belonging Autonomic Fault-Management processes, the AFM agent can decide to either restart a Network Interface Cards (NIC) on R2, or to restart R2 as whole. This results in the following potential actions *restart-node* and *restart-NIC*. Regarding the KPIs to optimize for the case study, we consider the following QoS metrics from the telecommunications domain: *delay*, *jitter*, *packet loss*, *out of order packets*, and *throughput*, as well as the KPIs of *overall security level*, *CPU utilization* and *memory consumption* on R2.

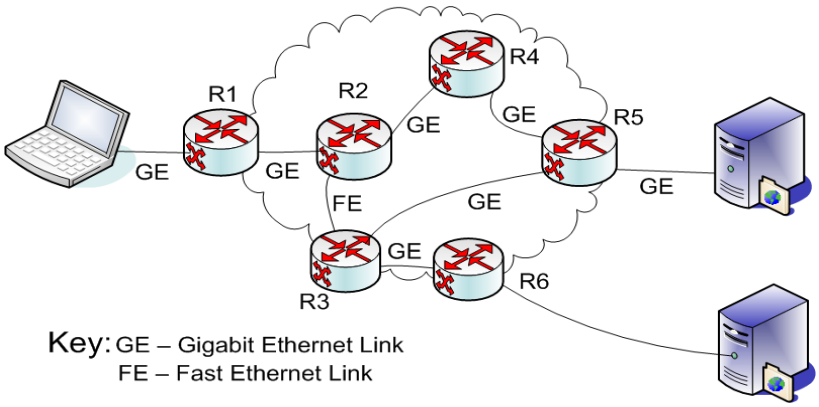

Fig. 12. The Reference Network for our Case Study

Table 1. Model Parameters for the Case Study

	RESTART-NODE	*RESTART-NIC*	*SET-HELLO-RATE-HIGH*	*SET-HELLO-RATE-MEDIUM*	*SET-HELLO-RATE-RATE-LOW*	*SET-ICMP-SUPPRESSION- HIGH*	*SET-ICMP-SUPPRESSION- MEDIUM*	*SET-ICMP-SUPPRESSION-LOW*	*CONSTRAINTS*	*WEIGHTS*
DELAY	5	4	5	1	-2	-3	1	3	2	15
JITTER	4	3	5	1	-2	-3	1	3	2	10
OUT OF ORDER	4	3	5	1	-2	-3	1	3	2	15
PACKET LOSS	5	3	5	1	-2	-4	2	4	2	20
THROUGHPUT	5	4	5	1	-2	-4	2	4	2	20
SECURITY LEVEL	-1	0	-1	1	2	5	1	-5	2	20
CPU	5	0	-4	1	3	0	0	0	2	10
MEM	5	0	-4	1	3	0	0	0	2	10
DISALLOW SOME OF THE ACTIONS TO EXECUTE SIMULTANEOUSLY										
	0	0	0	0	0	1	1	1	1	0
	1	1	0	0	0	0	0	0	1	0
	0	0	1	1	1	0	0	0	1	0
PRIORITIZE SPECIFIC ACTIONS										
	2	1	0	0	0	0	0	0	1	1
	2	0	1	0	0	0	0	0	1	1
	2	0	0	1	0	0	0	0	1	1
	2	0	0	0	1	0	0	0	1	1
	2	0	0	0	0	1	0	0	1	1
	2	0	0	0	0	0	1	0	1	1
	2	0	0	0	0	0	0	1	1	1

Table 1 presents the parameters of the model created based on the case study described hitherto. The first eight columns show the *impact matrix* of the model. The last two columns show correspondingly the constraints and the weights of the model determining the importance of the KPIs to optimize. For the impact of a certain action on the KPIs we take integer values from the interval [-5,5] meaning that a negative degrades the KPI, and correspondingly a positive value improves the KPI. For the weights, values from the interval [0,20] were considered. A number of interesting aspects are given by the lines, not assigned to a KPI. They demonstrate how model parameters can be extended beyond the modeling notions described in section 3, such that *particular* actions do never get executed simultaneously, and *certain* actions are prioritized. In the current case study, it is logically required that only one action at a time is executed from the sets *set-hello-rate-(low/medium/high)*, *restart-(node/nic),* and *set-icmp-suppression-(low/medium/high)* in case multiple actions from these sets are requested for synchronization. This is modeled by the second part of table 1, and is achieved through creating a relationship between these actions, and restricting this

Table 2. Illustrative Action Synchronization Requests

REQUESTED	*RESTART-NODE*	*RESTART-NIC*	*SET-ICMP-SUPPRESSION-LOW*	
SELECTED	*YES*	*NO*	*NO*	
REQUESTED	*RESTART-NIC*	*SET-ICMP-SUPPRESSION-LOW*	*SET-ICMP-SUPPRESSION-MEDIUM*	
SELECTED	*YES*	*YES*	*NO*	
REQUESTED	*RESTART-NODE*	*SET-ICMP-SUPPRESSION-HIGH*	*SET-ICMP-SUPPRESSION-LOW*	*SET-ICMP-SUPPRESSION-MEDIUM*
SELECTED	*YES*	*NO*	*NO*	*NO*
REQUESTED	*RESTART-NODE*	*SET-HELLO- RATE-LOW*	*SET-HELLO- RATE-MEDIUM*	*SET-ICMP-SUPPRESSION-HIGH*
SELECTED	*YES*	*NO*	*NO*	*NO*
REQUESTED	*RESTART-NIC*	*SET-HELLO-RATE-MEDIUM*	*SET-HELLO-RATE-HIGH*	*SET-ICMP-SUPPRESSION-MEDIUM*
SELECTED	*YES*	*NO*	*YES*	*NO*

relationship by extending the impact matrix and the constraints vector. Through this, only one action of each set is selected for execution after a synchronization process. The last segment of table 1 illustrates how an action can be set to have a higher priority than another one. In that case we prioritize the action *restart-node* over each of the other potential actions. This is achieved by an additional row in the *impact matrix* for each intended prioritization. Indeed, *restart-node* is favored based on the new *"impact values"*, and only one of the two actions in question, i.e. *restart-node* and the action with respect to which it is being prioritized, can be selected according to the new *constraints* (refer to the belonging entries in the constraints vector).

To illustrate the operation of an ASE component based on the case study and belonging model, we used our prototype, in order to issue different requests for action synchronization, and observe the resulting selected subsets of actions. Thereby, the ASE agent was operating on the binary program constituting the ASP version of the above described model. Table 2 summarizes some of the results from our experiments. It can be observed that indeed only one action from the sets *set-hello-rate-(low/medium/high)*, *restart-(node/nic),*and *set-icmp-suppression-(low/medium/high)* has been selected for execution, in case multiple actions from theses sets were requested for synchronization. Moreover, we can also see how the action *restart-node,* when requested, is always prioritized over the others. For the rest, the selection is performed based on the impact of the tentative actions on the fitness of the network as defined in (1) and (5). The results in table 2 show how such an ASE component can resolve emergent conflicts between parallel autonomic control loops, thereby always trying to achieve an optimal improvement in the operation of the network.

9 Conclusions and Future Research Directions

This article introduced a framework for ensuring the conflict-free and synchronized operation of multiple parallel autonomic control loops. Such a framework is useful in cases when the agents implementing the parallel control loops are not designed with the explicit awareness of each other, but instead operate in a way as to achieve their own goal and optimize the performance of the resources they were assigned to manage. By introducing components called Action Synchronization Engines (ASE), we propose to have an arbiter that enables the negotiation over tentative actions which are about to be executed in the scope of the parallel running control loops. An ASE component needs a model that allows it to reason about the impact of different actions such that a synchronization procedure can be performed. Therefore, we developed a mathematical model called ASP (Action Synchronization Problem), instances of which can be given to an ASE agent in order to facilitate its operation in a particular context. This model is based on the optimization of some key performance indicators (KPIs). It allows for: 1) specifying the importance of single KPIs, 2) specifying the impact of the potential actions on the KPIs in question, and 3) influencing the execution of actions with overlapping impact domains (in terms of KPIs). Moreover, in the course of the presented case study, we saw how the model can be used to: 4)

explicitly disallow the simultaneous execution of some actions, and 5) prioritize some actions over others. Unfortunately, this model results in a problem which is of very hard computational complexity. For that reason, an additional effort was presented that aims at relaxing the original problem in a way that it can be efficiently solved. The resulting formulation delegates some degree of decision to the requesting agents. These decisions are based on parameters (thresholds) for which we proposed a machine learning approach for their configuration. A comparison between the diverse techniques – direct binary optimization and machine learning based decisions – was also conducted. This comparison shows that the proposed reformulation does not degrade the quality of the action synchronization procedure, while at the same time reducing the memory consumption and improving the response time of an ASE component.

The approach proposed in this article opens a number of exciting research and development challenges. For example, the investigation of further ways for modeling runtime action synchronization and belonging efficient algorithms might be an interesting research topic. Besides, there is a need for sophisticated tooling that allows to easily create ASP models as proposed within this article. This could potentially be done in a way that the system's operator does not get involved into the mathematical details presented in this paper, but rather relies on easy to understand atomic artifacts, which can be efficiently combined to enable the self-managing system coping with various emergent situations. Finally, one might also consider the mapping of multiple parameters of the ASP model to business goals, and use this to achieve revenue by for example adapting the behavior of an autonomic network to specific type of expected traffic.

Acknowledgement. This work has been partially supported by EC FP7 EFIPSANS project (INFSO-ICT-215549).

References

[1] Autonomic Computing: An architectural blueprint for autonomic computing, IBM White Paper (2006)

[2] GLPK, GNU Linear Programming Kit, `http://www.gnu.org/software/glpk/` (as of date March 18, 2012)

[3] Chaparadza, R.: Requirements for a Generic Autonomic Network Architecture (GANA), suitable for Standardizable Autonomic Behavior Specifications for Diverse Networking Environments. IEC Annual Review of Communications 61 (December 2008)

[4] Bai, J., University, V., Abdelwahed, S.: A Model Integrated Framework for Designing Self-managing Computing Systems. In: The Proceedings of FeBid 2008, Annapolis, Maryland, USA, June 6 (2008)

[5] Debbabi, B., Diaconescu, A., Lalanda, P.: Controlling Self-Organising Software Applications with Archetypes. In: 2012 IEEE Sixth International Conference on Self-Adaptive and Self-Organizing Systems, pp. 69–78 (September 2012)

[6] EC funded- FP7-EFIPSANS Project, `http://efipsans.org/` (as of date March 18, 2013)

[7] Thorncraft, S.R.: Evaluation of Open-Source LP Optimization Codes in Solving Electricity Spot Market Optimization Problems. In: 19th Mini-Euro Conference on Operation Research Models and Methods in the Energy Sector, Coimbra, Portugal, September 6-8 (2006)
[8] Coin-OR, http://www.coin-or.org/ (as of date March 18, 2013)
[9] Derbel, H., Agoulmine, N.: ANEMA: Autonomic network management architecture to support self-configuration and self-optimization in IP networks. Published in the Elsevier Journal of Computer Networks (November 2008)
[10] Kephart, J.O., Das, R.: Achieving Self-Management via Utility Functions. IEEE Internet Computing 11(1), 40–48 (2007)
[11] Tesauro, G., Kephart, J.O.: Utility Functions in Autonomic Systems. In: Proceedings of the First International Conference on Autonomic Computing, May 17-18, pp. 70–77 (2004)
[12] Lehtihet, E., Derbel, H., Agoulmine, N., Ghamri-Doudane, Y.M., van der Meer, S.: Initial approach toward self-configuration and self-optimization in IP networks. In: Dalmau Royo, J., Hasegawa, G. (eds.) MMNS 2005. LNCS, vol. 3754, pp. 371–382. Springer, Heidelberg (2005)
[13] CASCADAS project, http://acetoolkit.sourceforge.net/cascadas/ (as of date March 18, 2012)
[14] Strassner, J., Agoulmine, N., Lehtihet, E.: FOCALE: A Novel Autonomic Networking Architecture. In: Latin American Autonomic Computing Symposium (LAACS), Campo Grande, MS, Brazil (2006)
[15] Famaey, J., Latre, S., Strassner, J., De Turck, F.: A hierarchical approach to autonomic network management. In: 2010 IEEE/IFIP Network Operations and Management Symposium Workshops, pp. 225–232 (2010)
[16] Höfig, E., et al.: On Concepts for Autonomic Communication Elements. In: Proc. of 1st IEEE International Workshop on Modelling Autonomic Communication Environments (2006)
[17] Prakash, A., et al.: "Formal Methods for Modeling, Refining and Verifying Autonomic Components of Computer Networks. Transactions on Computational Science 15, 1–48 (2012)
[18] Rodriguez-Fernández, C., et al.: Self-management capability requirements with SelfMML & INGENIAS to attain self-organising behaviours. In: Proceeding of the Second International Workshop on Self-organizing Architectures, SOAR 2010 (2010)
[19] Vassev, E.: ASSL: Autonomic System Specification Language - A Framework for Specification and Code Generation of Autonomic Systems. LAP Lambert Academic Publishing, Germany (November 2009)
[20] Vassev, E., Hinchey, M.: The ASSL approach to specifying self-managing embedded systems. Concurrency and Computation: Practice and Experience 24(16), 1860–1878 (2012)
[21] Vassev, E., Mokhov, S.A.: Developing Autonomic Properties for Distributed Pattern-Recognition Systems with ASSL - A Distributed MARF Case Study. Transactions on Computational Science 15, 130–157 (2012)
[22] NLOPT library, http://ab-initio.mit.edu/wiki/index.php/NLopt (as of date March 18, 2012)
[23] Runarsson, T.P., Yao, X.: Search biases in constrained evolutionary optimization. IEEE Trans. on Systems, Man, and Cybernetics Part C: Applications and Reviews 35(2), 233–243 (2005)

[24] Powell, M.J.D.: Direct search algorithms for optimization calculations. Acta Numerica 7, 287–336 (1998)
[25] Birgin, E.G., Martínez, J.M.: Improving ultimate convergence of an augmented Lagrangian method. Optimization Methods and Software 23(2), 177–195 (2008)
[26] Tcholtchev, N., Chaparadza, R., Prakash, A.: Addressing Stability of Control-Loops in the Context of the GANA Architecture: Synchronization of Actions and Policies. In: Spyropoulos, T., Hummel, K.A. (eds.) IWSOS 2009. LNCS, vol. 5918, pp. 262–268. Springer, Heidelberg (2009)
[27] Kastrinogiannis, T., Tcholtchev, N., Prakash, A., Chaparadza, R., Kaldanis, V., Coskun, H., Papavassiliou, S.: Addressing Stability in Future Autonomic Networking. In: Pentikousis, K., Agüero, R., García-Arranz, M., Papavassiliou, S. (eds.) MONAMI 2010. LNICST, vol. 68, pp. 50–61. Springer, Heidelberg (2011)
[28] Li, N., Chen, G., Zhao, M.: Autonomic Fault Management for Wireless Mesh Networks. Electronic Journal for E-Commence Tools and Applicatoins (eJETA) (January 2009)
[29] The FCAPS management framework: ITU-T Rec. M. 3400
[30] Fréville, A.: The multidimensional 0-1 knapsack problem: An overview. European Journal of Operational Research 155, 1–21 (2004)
[31] Mak-Karé Gueye, S., de Palma, N., Rutten, E.: Coordinating energy-aware administration loops using discrete control. In: Proc. of the 8th International Conference on Autonomic and Autonomous Systems, ICAS 2012 (March 2012)
[32] Gorski, J., Paquete, L., Pedrosa, F.: Greedy algorithms for a class of knapsack problems with binary weights. Computers & Operations Research 39, 498–511
[33] Gueye, S.M.-K., Rutten, E., Tchana, A.: Discrete Control for the Coordination of Administration Loops. In: 2012 IEEE Fifth International Conference on Utility and Cloud Computing (UCC), November 5-8 (2012)
[34] Gilmore, P.C., Gomory, R.E.: The theory and computation of knapsack functions. Operations Research 14, 1045–1075 (1966)
[35] EMF, `http://www.eclipse.org/modeling/emf/` (as of date March 18, 2012)
[36] Hessel, A., Pettersson, P.: Model-Based Testing of a WAP Gateway: An Industrial Case-Study. In: Brim, L., Haverkort, B.R., Leucker, M., van de Pol, J. (eds.) FMICS 2006 and PDMC 2006. LNCS, vol. 4346, pp. 116–131. Springer, Heidelberg (2007)
[37] Tcholtchev, N., et al.: Scalable Markov Chain Based Algorithm for Fault-Isolation in Autonomic Networks. In: 2010 IEEE Global Telecommunications Conference, GLOBECOM 2010, pp. 1–6 (2010)
[38] Hellerstein, J.L., et al.: Feedback Control of Computing Systems. Wiley-IEEE Press (September 2004) ISBN: 978-0-471-26637-2
[39] Schlittgen, R.: Einführung in die Statistik. 9. Auflage. Oldenbourg Wissenschaftsverlag, Oldenbourg (2000) ISBN 3-486-27446-5
[40] Wang, F., et al.: A Route Flap Suppression Mechanism Based on Dynamic Timers in OSPF Network. In: Proceedings of ICYCS 2008, pp. 2154–2159 (2008)
[41] Farnum, N.R.: Some results concerning the estimation of beta distribution parameters. J. Oper. Res. 38(3), 287–290 (198)
[42] de Oliveira Jr., F.A., Sharrock, R., Ledoux, T.: Synchronization of multiple autonomic control loops: Application to cloud computing. In: Sirjani, M. (ed.) COORDINATION 2012. LNCS, vol. 7274, pp. 29–43. Springer, Heidelberg (2012)
[43] Tcholtchev, N., Grajzer, M., Vidalenc, B.: Towards a Unified Architecture for Resilience, Survivability and Autonomic Fault-Management for Self-Managing Networks. In: Dan, A., Gittler, F., Toumani, F. (eds.) ICSOC/ServiceWave 2009. LNCS, vol. 6275, pp. 335–344. Springer, Heidelberg (2010)

[44] Charalambous, T., Kalyvianaki, E.: A min-max framework for CPU resource provisioning in virtualized servers using H-infinity Filters. In: 2010 49th IEEE Conference on Decision and Control (CDC), December 15-17, pp. 3778–3783 (2010)
[45] Kalyvianaki, E., Charalambous, T., Hand, S.: Resource Provisioning for Multi-Tier Virtualized Server Applications. Computer Measurement Group Journal (CMG Journal 2010) (126), 6–17 (2010)
[46] Kalyvianaki, E., Charalambous, T., Hand, S.: Self-Adaptive and Self-Configured CPU Resource Provisioning for Virtualized Servers Using Kalman Filters. In: 6th Int. Conference on Autonomic Computing andCommunications, ICAC 2009 (2009)
[47] ETSI AFI, http://portal.etsi.org/portal/server.pt/community/afi/344 (as of date March 18, 2012)
[48] Frey, S., Diaconescu, A., Demeure, I.: Architectural Integration Patterns for Autonomic Management Systems. In: 9th IEEE International Conference and Workshops on the Engineering of Autonomic and Autonomous Systems (EASe 2012), Novi Sad, Serbia, April 11-13 (2012)
[49] Agoulmine, N.: Autonomic Network Management Principles. Academic Press (December 2010) ISBN: 0123821908