A Hierarchical Approach to Autonomic Network Management

Jeroen Famaey*, Steven Latré*, John Strassner†, and Filip De Turck*

* Department of Information Technology
Ghent University – IBBT
Gaston Crommenlaan 8/201, B-9050 Gent, Belgium
Email: jeroen.famaey@intec.ugent.be

† Division of IT Convergence Engineering
Pohang University of Science and Technology
Pohang, Korea

***Abstract*—Recently, the autonomic communication networks paradigm has been introduced as a solution to the increasing management complexity of communication networks in the Future Internet. In order to encompass the large-scale nature of these networks, a general consensus has been reached that the supporting autonomic management architectures should be distributed for scalability reasons. However, several open issues related to the distribution of autonomic components remain to be solved.**

In this paper, we propose a novel approach to structuring distributed autonomic components in large-scale communication networks. The approach is generic and can be applied to many existing autonomic architectures and control loops. The autonomic components are structured in a hierarchy, which simplifies the interaction between components, and allows them to manage resources and govern child components in a more scalable manner. In addition to giving a detailed description of the hierarchical architecture, the advantages of the proposed approach are validated through analytical evaluation results.

I. Introduction

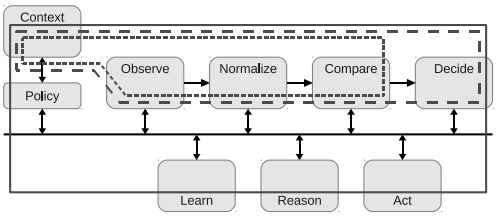
In recent years, communication networks have greatly increased in size, complexity, and heterogeneity. Additionally, the end-user and service requirements have become drastically more diverse and stringent. Hence, managing these complex and large-scale systems is proving increasingly difficult and this complexity is likely to increase in the Future Internet. To alleviate the problems associated with managing current and future communication networks, the autonomic communication networks paradigm has been introduced [1, 2].

The ultimate goal of autonomic network management systems is to automatically adapt the network's services and resources in accordance with changing environmental conditions and user needs [3]. Policy-Based Network Management [4, 5, 6] gives these systems the ability to automatically perform low-level configurations in compliance with high-level business goals. This will allow human administrators to focus on high-level tasks. Consequently, the increasing management complexity will be handled by the system itself.

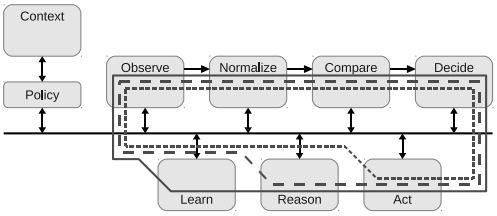
It has been generally agreed upon that autonomic architectures for managing current and future networks and services should be distributed for scalability reasons [7, 8, 9]. Distribution of autonomic components provides a means to keep up with the exploding growth of the number of network devices, services, and end-users. However, little research has been performed on how exactly these distributed autonomic components should collaborate and communicate. As a first step, a solution has been proposed in the form of combining autonomic components in a hierarchical structure [8, 10]. In this paper we build upon these first ideas, and give a detailed description of the interactions between autonomic components in a hierarchical autonomic management architecture. We argue that by grouping autonomic components into a hierarchy, the network overhead associated with managing network devices and other resources can be greatly reduced. Additionally, dissemination of context, propagation of policies, and collaboration between autonomic components can be more efficiently orchestrated, resulting in a more scalable architecture. Finally, the hierarchical structure can be logically mapped to the structure of the organization and their infrastructure, simplifying configuration and management at all layers of the organization.

The contributions of our work are threefold. First, we propose a novel hierarchical approach to structuring autonomic components. Second, the interactions in this hierarchically structured autonomic network are identified and discussed in detail. This includes the propagation of context and policies, and governance of child autonomic components. Third, an analytical study that evaluates the scalability of this new approach has been performed. The results are discussed in the second part of this paper.

This paper is organized as follows. A brief overview of existing autonomic management architectures, and more specifically FOCALE, is given in Section II. Subsequently, Section III further explores the proposed hierarchical architecture. The introduced concepts are evaluated in Section IV. Finally, Section V concludes this paper.



(a) outer loop view



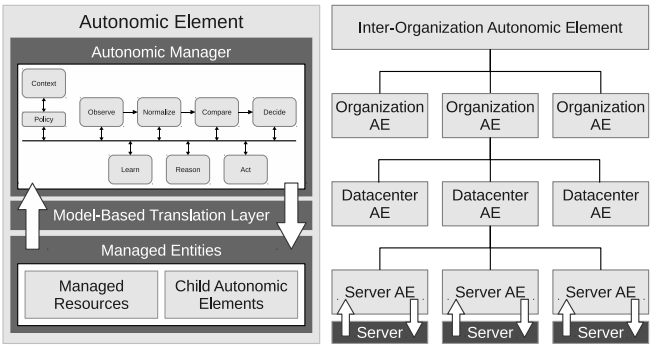
(b) inner loop view

Figure 1: The FOCALE control loops [8]

II. FOCALE Autonomic Management Architecture

Since the conception of autonomic computing and communications, many autonomic control loops and architectures have been proposed [8, 11, 12]. They share the common goal of autonomically adapting the behavior of managed resources if their state differs from the desired state. However, the term "autonomic" is often interpreted in different ways, which is reflected in the various approaches used to implement autonomic control loops. The hierarchical autonomic architecture proposed in this paper is based on the FOCALE architecture and control loops [8, 10]. However, the ideas presented in this paper can conceptually be applied to other autonomic architectures as well, as these architectures face the same challenges regarding distribution. Furthermore, in describing the hierarchical autonomic architecture, we do not introduce specific FOCALE components, but merely use FOCALE as an example throughout the paper. We have chosen FOCALE because it aims to free network administrators from performing low-level configuration tasks, allowing them to focus on high-level network planning and optimization. These low-level tasks are performed by the network itself, which uses reasoning and learning components to adapt its behavior to context changes. This adaptive behavior is governed by policies, representing the high-level business goals. Additionally, the FOCALE Cognitive Model [8] supports collaboration between Autonomic Elements (AE) by grouping them into communities, and providing hooks that support centralized and decentralized governance. Cooperation between AEs is fundamental towards achieving the hierarchical autonomic architecture proposed in this paper.

The FOCALE architecture provides a set of outer and inner control loops, as shown in Figure 1. The outer control loops perform large-scale adjustments by reacting to context changes. On the other hand, the inner control loops make more detailed adjustments of functionality within a specific context. Both outer and inner loops come in three types:



(a) hierarchical autonomic element **(b)** example hierarchy tree

Figure 2: A hierarchical autonomic element manages a set of managed resources and child autonomic elements, together forming the set of *managed entities*; The autonomic elements are structured in a logical tree topology

reactive, deliberative, and reflective. The reactive path is taken when adapting to a previously analyzed context change. In such a case, a previously inferred behavior change can be performed, without the need for complex reasoning. The deliberative control loops are used when context changes that are not sufficiently well understood take place. Finally, the reflective loops provide a means to better understand how context changes affect the goals of AEs. Note that the three types of outer and inner loops of FOCALE were inspired by congnitive psychology, and correspond to modeling how a human makes decisions using short- and long-term memory.

In addition to its advanced control loops, FOCALE introduces the notion of the enhanced Autonomic Element. It is an abstraction that allows FOCALE to provide distributed functions such as communication, learning, reasoning, and management. Each AE provides a set of services to perform knowledge management, composition, business-enabling, and orchestration. Additionally, AEs can cooperate and collaborate in communities, by sharing functionality and information.

The next section gives a detailed overview of the concepts we devised to augment existing autonomic management architectures in order to support hierarchical collaboration between autonomic components.

III. Policy-Based Hierarchical Autonomic Management Architecture

In our proposed hierarchical autonomic network management architecture, AEs are grouped together in cooperating communities, or *clusters*. Each AE is composed of a set of *managed entities*, which are either *managed resources* or AEs themselves. Managed resources are oblivious to the autonomic management capabilities of the network, and fully depend on the parent AE to govern their management decisions. Child AEs are guided by their parent, but also have autonomic decision-making capabilities of their own. Figure 2 shows the structure of a hierarchical AE and gives an example of how the hierarchy can be mapped to the physical infrastructure. Figure 2a shows a simplified view of the FOCALE AE. The heart of our enhancement lies in the managed entities

container, which replaces the FOCALE managed resource. The managed entities container consists of managed resources and/or child AEs. In the example shown in Figure 2b each server is managed by its own AE. Additionally, all servers within a datacenter are grouped together in a cluster. At the top layer, several organizations cooperate via an inter-organization AE. Note that combining different organizations into a single AE introduces additional difficulties. This, and an alternative method for collaboration between organizations is further discussed in Section III-D.

By introducing parent-child relationships, the AEs in the network will form a logical tree. At the bottom layer AEs only manage a set of managed resources, while the root of the tree effectively governs the entire network. This approach introduces a hybrid management scheme, where AEs within a cluster are managed in a centralized way by the parent AE, while management across the tree is distributed. Note that although logically the parent AE is a single entity, it may be physically distributed across multiple devices to improve scalability and robustness.

The hierarchical autonomic approach has several advantages over a flat autonomic architecture. First, scalability is improved, as context no longer needs to be exchanged between every pair of cooperating AEs, but only needs to be sent to the parent AE. This greatly reduces management overhead. Additionally, by aggregating and filtering the exchanged context, overhead can be even further reduced. AEs at higher levels in the hierarchy thus have a broader, but less detailed, view on the managed resources. This allows them to perform large-scale reasoning and decision-making in a scalable manner. On the other hand, AEs at the bottom of the hierarchy can use more detailed information to react faster and more precise, but on a smaller scale. This approach also mirrors the design of FOCALE's outer and inner control loops, with the outer loops defining the coarse context for governance, and the inner loops defining the finer-grained management within that context. Second, AEs that need to cooperate or share common goals can be grouped together in a cluster. The hierarchical structure greatly simplifies governing the interactions between them, and aligning their behavior. Additionally, this can be exploited to facilitate the business interactions between organizations. Finally, the layers of the tree can be more easily mapped to the hierarchical structure of organizations and infrastructure. This facilitates the translation and mapping of business goals and policies to the actual network configurations.

The rest of this section elaborates upon the different types of interaction between AEs in the hierarchy tree.

A. Cluster Management

A cluster is defined as the group of AEs that share the same parent AE. It is necessary to determine which AE in the cluster will act as the parent. In a stable network, where devices stay online for prolonged periods, the parent AE can be statically determined. In the example shown in Figure 2b, a datacenter AE can be chosen in advance, as datacenters are mostly static environments. However, in a more dynamic network, devices and thus AEs might randomly go offline and online. In such a case, a robust leader election protocol can be used to dynamically determine cluster parents [13, 14].

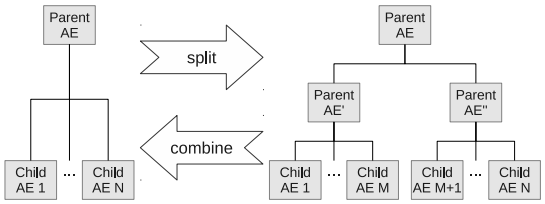


Figure 3: A cluster can be dynamically split into several sub-clusters to improve scalability

In addition to parent selection, an AE must be assigned to a specific cluster. As the general structure of a network is static, even over longer periods of time, we believe an AE would not often change its position in the hierarchy. The cluster of an AE can thus be statically specified using policies. However, for scalability reasons, clusters becoming too large could be split into sub-clusters, by introducing an additional layer in the hierarchy. Figure 3 shows this process by way of an example.

Policies can be used to define a maximum threshold for overhead generated by intra- and inter-cluster communication. If this threshold is exceeded, the autonomic manager detects the invalid state and executes the cluster splitting algorithm. Analogous to splitting, peer clusters with small populations can be recombined. An example policy for splitting clusters is shown below (using the Ponder policy specification language [15]):

```
inst oblig splitCluster {
    on      overhead(cluster) > X %
    subject p = parent(cluster)
    do      p.splitChildCluster()
}
```

In the example, the parent AE of a cluster is asked to split its children into multiple sub-clusters if the overhead generated by the cluster for management communications is greater than X% of the consumed bandwidth. More details on determining the splitting threshold are given in Section IV-B.

B. Context Dissemination

Context is a vital part of any autonomic system. It is used to model the current state of the managed entities, which in turn allows the system to adapt to changes when necessary. Context of managed resources can be obtained by way of active or passive monitoring using standard protocols or techniques, such as SNMP [16]. Child AEs, on the other hand, cannot be directly monitored. They make parts of their own context available to the parent AE, for example through a publish-subscribe mechanism. Using policies, context can be flagged private or public. Public context is made available to the parent, while private context is not disseminated. This allows for enforcing privacy and ensures that contextual data can be exchanged between organizations. For example, if an inter-organization AE groups together several cooperating organizations, they do

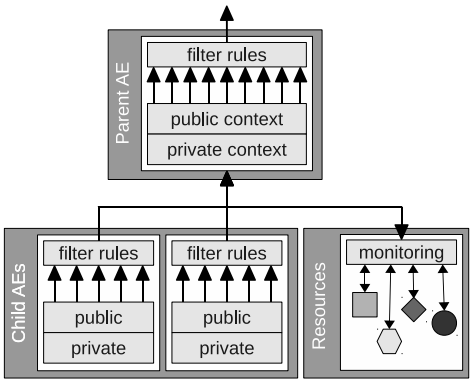


Figure 4: Public context of an AE is aggregated and filtered before it is disseminated to the parent AE; Private context is not made available to the parent

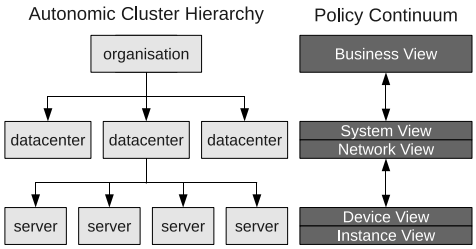


Figure 5: High-level policies are propagated down the cluster hierarchy; The AEs can be mapped to the views of the Policy Continuum

not want all information about their infrastructure and services to be made public, but only specific types of information that are needed to support the collaboration. Figure 4 shows the process of context dissemination throughout the hierarchy in more detail.

Although all public context is available to the parent AE, not all data is unconditionally sent to the parent. In a typical publish-subscribe mechanism, which can form the basis for the context dissemination process, filter rules are used that allow parent AEs to inform their children about the context in which they are interested [17]. Additionally, filter rules can be used to define the aggregation of detailed information before it is disseminated to the parent. Filtering and aggregation allow the overhead, in terms of bandwidth consumption and reasoning time, to be greatly reduced. Aggregation gives the parent AE a broader, but less detailed, view on the managed entities, allowing it to reason and take decisions on a larger scale, without greatly increasing execution time of reasoning algorithms.

Policies also play a vital role in combination with filter rules: they are used in two distinct situations. First, similar to flagging context public or private, policies are also used as a means to limit the visibility of contextual data to parent AEs. For example, the AE managing a datacenter might have detailed information on the resource consumption of all its servers. However, an AE at a higher level might only be allowed to view more general statistics showing average or maximum consumption over all servers in the datacenter. Policies allow tuning the amount of aggregation that is needed between different AE levels, and thus tune the view a parent AE gets on its children. These policies can be specified both by human operators as well as AEs; in the first case an operator will restrict the view because of trust issues between parent and child AE (e.g. they belong to different organizations), in the latter case policies can be automatically specified when the overhead exceeds a threshold. Second, policies can be used to enable the dynamic composition of filter rules. For example, a parent AE managing a video delivery service might only request a general Quality of Experience score from its children during normal operation. However, when this score becomes too low, more detailed information could be requested, in order to diagnose and resolve the problems. In Latré et al. [17], an ontology based approach is suggested that uses semantic reasoning to dynamically change the set of filter rules. Here, policies can be specified by the network operator in the form of ontology rules.

C. Policy Interaction

Although administrators of an autonomic system are not directly involved in configuring management algorithms and managed entities, they do control the entire process by adding policies to the policy repository. At the higher levels of the hierarchy, these policies correspond to the business goals of the organization. At lower levels they become more specific, defining the desired state of AEs in more technical ways. The approach of translating policies from general business goals to more implementation-specific technical rules has already been proposed as the Policy Continuum [5]. Figure 5 shows an example of how the views of the Policy Continuum could be mapped to the cluster hierarchy. However, this should be viewed as just an example. In reality, many possible ways of mapping the continuum views to the hierarchy exist. As shown in the figure, a one-to-one mapping between the Policy Continuum and the cluster hierarchy is not always possible. However, multiple continuum views can be combined within a single AE layer, or views can stretch over multiple AEs.

The Policy Continuum paradigm provides a clear abstraction of the complexity present in the different levels. By linking policies at different levels, management problems can be split more easily into smaller, and hopefully easier to tackle, problems. If the process of linking policies can be automated, the policy translation process can be automated as well. In such an approach, changing a policy at the higher level can immediately trigger the change of one or more policies at the lower levels. If policies can be changed in multiple ways, strategies can aid in determining which path to take. The automation of policy translation requires at least a policy authoring infrastructure that is able to detect and tackle policy conflicts, as argued in Davy et al. [18]. In this approach, a policy conflict analysis algorithm is used, which can form the basis for policy translation.

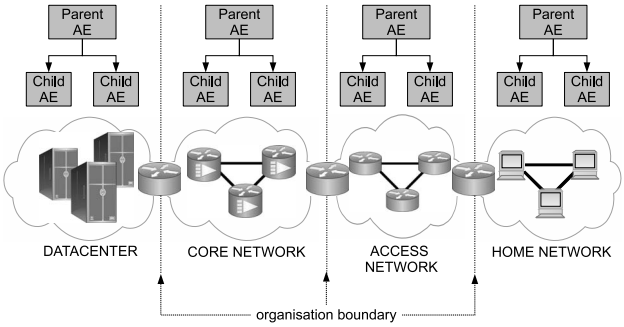


Figure 6: Mapping of different hierarchical structures onto an exemplary network topology, spanning multiple domains.

D. Autonomic Element Collaboration

In the previous sections we discussed how the exchange of policies and context on one hand and the organization of autonomic elements themselves on the other hand can be handled. Another important communication aspect in a distributed management environment is the behavioral orchestration between AEs. Such orchestration is both needed in an inter- and intra-domain management scenario.

An inter-domain management scenario occurs when two or more organizations collaborate to offer composite services. For example, a cloud computing provider, wanting to offer an end-to-end Quality of Service guarantee for a service, needs to collaborate with the network providers that manage the networks where his data transits to ensure that the management actions he/she undertakes are supported or, even better, reenforced by those network providers. In the strict hierarchical approach, the solution to this problem would be to instantiate a new inter-organization AE that orchestrates the QoS guarantees. However, in an inter-organizational scenario, such an approach is often not feasible due to privacy issues or a lack of a shared infrastructure. We argue that, between organizations, interactions between parent AEs belonging to different organizations should be maintained instead of enforcing the strict parent child interaction. This is illustrated in Figure 6, which depicts the mapping of several management hierarchies onto an exemplary network topology.

This parent-to-parent communication can only work if there is an unambiguous interaction agreement on (1) what each party can expect from each other, or in other words, what management functions are made available to each organization and (2) what the effect is of each management function on the network context. While crucial in an inter-domain scenario, such agreements can also play an important part in regular parent-child AE interactions. For a parent AE, a good management strategy is to assign his child AEs with specific management tasks that have a smaller scope and require a smaller reaction time. The assigned child AEs are then responsible for independently complying with the assigned task. Without an interaction agreement, the parent AE has no formal guarantee that the assigned task will be executed. Through interaction agreements, one party can delegate management authority to another party in both an inter- and intra-domain scenario.

As proposed in van der Meer et al. [19], such an interaction agreement can be established by using the Design by Contract paradigm as originally presented by Bertrand Meyer [20]. A contract enables the formal specification of the functional and non-functional characteristics of a distributed artefact such as a management function. In van der Meer et al. [19] the L-ADS language is presented, which allows contracts to be defined for a distributed management environment.

E. Management Algorithms

The management algorithms are responsible for configuring the managed entities, to make sure their state reflects a desired state of the system. They are guided by the reasoning and learning components, which may change the algorithms' parameters to adjust their behavior. Managed resources are directly configured by the algorithms, but child AEs are not. However, as discussed earlier, the algorithms are capable of influencing child behavior through policies or contracts.

An algorithm often exists at many layers in the hierarchy. However, its behavior will differ based on its location. For example, at an organization level, a resource reservation algorithm can insert policies into specific datacenter AEs specifying the amount of resources to reserve for each service. At the datacenter level this same algorithm would select specific servers on which to execute the services. Finally, at the server level, the service would be executed and a specific amount of resources would be reserved for it.

IV. Evaluation

The hierarchical approach of structuring AEs in an autonomic communication network has both qualitative and quantitative advantages compared to classical flat architectures. The qualitative advantages have already been clarified in previous sections. Quantitatively, the hierarchical approach is expected to greatly reduce overhead and increase the view of the autonomic managers on the network and its resources. Existing flat architectures often sacrifice accuracy of context information in order to reduce management overhead, and thus increase scalability. This often leads to suboptimal decision-making. The hierarchical approach alleviates this problem by giving selected autonomic managers a broader view on the network and its resources, and letting them govern components with more narrow but detailed context information.

In this section, the overhead introduced by exchanging context information in both flat and hierarchical autonomic architectures is evaluated, by way of an analytical model. First, a generic model is given. Subsequently, it is applied to a specific scenario in order to obtain more concrete results.

A. Analytical Model

In flat distributed management architectures, management components enter into peer relationships with other components [21]. These relationships are then used to exchange context information. The number of neighbors of a component thus controls the size of its view on the network, but also the

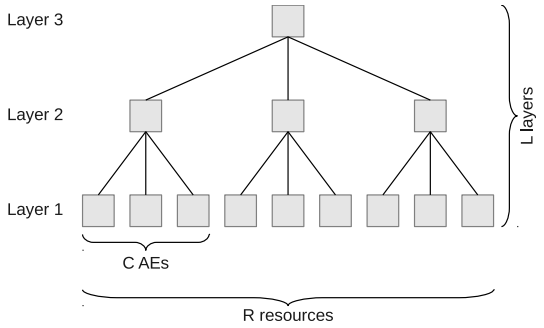


Figure 7: An example hierarchical AE topology, with $R = 9$, $C = 3$, and $L = 3$

generated overhead. Increasing the number of neighbors of a component will thus increase the accuracy of the available context information, but also the generated network overhead.

Before quantifying this overhead and the size of the network view, we introduce some notations. Assume the network consists of R network resources, each governed by an AE. Additionally, each AE has N neighbors, with $N < R$. Let B represent the size in bytes of one context exchange between two AEs. Information is exchanged between neighbor AEs every I seconds.

The generated overhead per second (in bytes) for a flat architecture can then be calculated as follows

$$\mathcal{O}_{flat} = \frac{R \times N \times B}{I} \tag{1}$$

In addition, the size of the view of an AE (as a fraction of the full view) is calculated as follows

$$\mathcal{V}_{flat} = \frac{N+1}{R} \tag{2}$$

In the hierarchical architecture, every AE (except the root) is governed by its parent. Therefore, it only sends context information to the parent AE. The parent aggregates and filters the context received by its children before propagating it to its own parent. The equations for the hierarchical architecture thus become more complex, and some additional notations are needed. The architecture consists of L layers. The bottom layer, layer 1, consists of R AEs (one for each network resource). At all layers, AEs are grouped in clusters of at most size C, except the top layer, layer L, which consists of a single root AE. At layer l (with $1 \leq l < L$), a context exchange with the parent is of size B_l bytes. The exchanged context may be different in size at every layer, as it is filtered and aggregated before being propagated. Figure 7 shows an example hierarchical AE topology, with $R = 9$, $C = 3$, and $L = 3$.

The generated overhead per second for hierarchical architectures can now be calculated by determining the number of AEs per layer. At layer 1, there is 1 AE per resource, which equals R. At layer 2, there is 1 AE per layer 1 cluster, which equals $\lceil R/C \rceil$. This can be generalized to the number of AEs at layer l (with $1 \leq l < L$) being equal to $\lceil R/C^{l-1} \rceil$. As each AE at layer l sends B_l bytes of context information to its parent every I seconds, the generated overhead (in bytes) can be calculated as follows

$$\mathcal{O}_{hier} = \frac{1}{I} \times \sum_{l=1}^{L-1} \left(B_l \times \left\lceil \frac{R}{C^{l-1}} \right\rceil \right) \tag{3}$$

The size of the view is different at every layer. At layer 1, an AE only has context information about its own resource, which results in an $1/R$ fraction of the entire network view. This can be generalized to an arbitrary layer l, resulting in the fraction

$$\mathcal{V}_{hier}(l) = \frac{C^{l-1}}{R} \tag{4}$$

Note that if R is not divisible by C, one layer 1 cluster will contain less than C resources, and Equation 4 is thus only an upper boundary for that cluster and its parent clusters.

In addition to the total management overhead, the context information that needs to be processed by a single AE also plays an important role in the performance. If the load is not properly balanced, heavily loaded AEs might not be able to process and reason on the large amounts of context they receive. In the flat architecture, under the assumption that all AEs have the same number of neighbors, the load is equally spread across all AEs. The maximum context per second (in bytes) that an AE receives can be calculated as follows

$$\mathcal{R}_{flat} = \frac{N \times B}{I} \tag{5}$$

For the hierarchical architecture, the calculation is again less straight forward. At each layer, except the top, an AE has at most C children. However, at the top layer, the root AE has $\lceil R/C^{L-2} \rceil$ children. This results in the following equation for calculating the maximum AE load in bytes per second

$$\mathcal{R}_{hier} = \frac{1}{I} \times \max \left(\max_{1 \leq l \leq L-2} (B_l) \times C, B_{L-1} \times \left\lceil \frac{R}{C^{L-2}} \right\rceil \right) \tag{6}$$

In the rest of this section, the analytical model is applied to a specific service delivery middleware scenario, and more concrete results are derived from the equations.

B. Results

In order to derive more tangible results from the analytical model, we apply it to a *service delivery middleware* scenario [21, 22, 23]. A service delivery middleware is a middleware substrate responsible for managing a large set of application services. The tasks performed by the middleware include: allocation of resources to service instances, selection and composition of services, and request admission control. In this section we focus on the resource allocation component. It is responsible for deciding which application services to execute on every server in the datacenter. It is often a distributed component that is executed on every server separately, using context information from other servers. The resource allocation component, as described in [21], uses the following context information:

- The available and used resources per resource type on the server

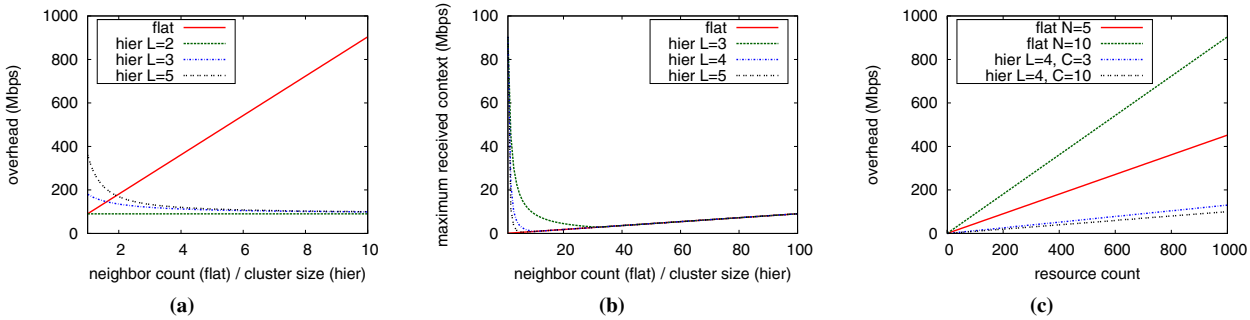


Figure 8: Quantitative results for a service delivery middleware scenario. The graphs show the total network overhead and maximum received context per AE (both in megabits per second) as a function of the size of the network R, the neighbor count N (for the flat architecture), and the cluster size S (for the hierarchical architecture)

- The number of requests received and satisfied for every application service
- The amount of required and supplied resources per resource type for every application service

Assume the number of different resource types equals T, and each server executes on average S application services. Additionally, associated with each resource type and application service is a 4 byte identifier. A resource amount is represented using 8 bytes, while request counts are represented using 4 byte integers. The information about a resource type requires $8+8+4 = 20$ bytes, and about an application service $4+4+4+(8+8+4)\times T = 12+20\times T$ bytes. Therefore, the size in bytes of a context exchange message can be calculated as follows

$$B = 20 \times T + S \times (12 + 20 \times T) \tag{7}$$

In the rest of this section we assume: a datacenter with 1000 servers (R), 5 resource types (T), on average 100 application services per server (S), and a context exchange interval of 1 second (I). Using these values, $B = 11300$ bytes per second can be derived. Note that the actual values of these parameters do not influence the relative performance of the architectures, but merely the absolute values.

In the hierarchical architecture, context information is averaged over all servers, and averaged per service when it is propagated to the parent AE. This means we can assume the size of a context exchange message is the same at all layers in the hierarchy, or $B = B_l$ for $1 \leq l < L$.

Some specific results are shown in Figure 8. The total overhead generated by context exchange (in megabits per second) is shown in Figures 8a and 8c. The maximum context received by any AE (also in megabits per second) is shown in Figure 8b. The first two graphs depict the evaluation metrics as a function of the number of neighbors N, for the flat architecture, and cluster size C, for the hierarchical architecture. The third graph is depicted as a function of the network size N.

As shown in Figure 8a, the hierarchical architecture outperforms the classical flat approach in terms of overhead for almost every combination of N and C. The flat architecture only generates less overhead for a cluster size C of less than 2 AEs per cluster, which is not a suitable cluster size for a network with 1000 resources anyway. Additionally, the performance of the flat approach clearly degenerates quickly as the number of neighbors increases. Even for $N = 10$, which is a view of only about 1% of the entire network, the flat approach generates up to 10 times as much overhead as the hierarchical architecture. More specifically, for 1000 servers, the flat architecture generates 452 and 904 Mbps overhead for 5 and 10 neighbors respectively. On the other hand, the overhead is limited to 130 and 100 Mbps for respectively a cluster size of 3 and 10 in the hierarchical case. Finally, the performance of the hierarchical architecture quickly approaches the optimum, even for a topology with 5 layers and a cluster size of only 10 AEs per cluster.

As previously shown, increasing the cluster size decreases the total generated overhead of the hierarchical architecture. However, as shown in Figure 8b, it also negatively influences the maximum AE load. Therefore, it is necessary to leverage this trade-off by selecting a cluster size C that gives good results for both metrics. As shown in the figure, the maximum load decreases up to a certain cluster size, after which it grows again. The location of this optimum depends on the number of layers L in the topology, and can be derived from Equation 6. From this equation, it can be derived that the optimal maximum load is reached when the following equality holds

$$\max_{1 \leq l \leq L-2} (B_l) \times C = B_{L-1} \times \left\lceil \frac{R}{C^{L-2}} \right\rceil \tag{8}$$

As stated earlier, for the considered scenario we assume that $B = B_l$ for $1 \leq l < L$. If furthermore the special case where R is divisible by C is considered, this equation can be rewritten as follows

$$C = \sqrt[L-1]{R} \tag{9}$$

Although this equation does not return the exact optimum if R is not divisible by C, it does remain a good approximation. The cluster management algorithm (cf. Section III-A) can exploit Equation 9 to help determine the optimal cluster size.

The overall scalability of the approaches is compared in Figure 8c. It depicts the increase in network overhead for context dissemination as a function of increasing number of

managed resources, and thus AEs. In order to keep the same relative view size of the network, in the flat architecture, N should grow with the network size. However, as shown in the figure, doubling the size of N also doubles the overhead. This introduces a scalability bottleneck, which can only be circumvented by reducing the relative view size as the network size grows. This in turn will cause the optimality of management algorithms to decrease as the network grows in size. On the other hand, increasing the cluster size in the hierarchical architecture, actually decreases total overhead. This can be exploited to improve scalability. As the network size grows, cluster size can be increased proportionally in order to contain the increase in overhead.

V. Conclusion & Future Work

This work extends existing architectures for autonomic network and service management with several relevant contributions. A novel approach to collaboration and interaction between autonomic elements is introduced. By structuring them in a hierarchical manner, the overhead generated by exchanging context can be greatly reduced. This greatly improves scalability compared to flat decentralized architectures. Additionally, the hierarchical approach provides a more intuitive mapping between the Policy Continuum and the autonomic components. This eases human intervention at all layers of the infrastructure and organization. By way of aggregating and filtering context information, autonomic elements can be provided with a more fine-grained but narrow, or less detailed but wide view on the network and its resources. This can be exploited by management algorithms to operate at different layers of the hierarchy, taking more general decisions at the top and more specific ones at the bottom layers.

The quantitative advantages, such as a reduction in generated network overhead and improved scalability, were studied in more detail using an analytical model. First, a general model was constructed. Second, this model was applied to a specific scenario, which resulted in several concrete conclusions. The results were used to study the trade-off between scalability in terms of generated network overhead, and the size of the view on the network and its resources.

In future work, we plan to focus on devising service management algorithms that exploit the hierarchical structure of the proposed architecture. Additionally, the hierarchical architecture will be extended to a hybrid federation-based model.

Acknowledgment

Jeroen Famaey is funded by the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT). Steven Latré is funded by the Fund for Scientific Research Flanders (FWO). This work is sponsored in part by the WCU (World Class University) program through the Korea Science and Engineering Foundation funded by the Ministry of Education.

References

[1] S. Dobson, S. Denazis, A. Fernandez, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli, "A survey of autonomic communications," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 1, no. 2, pp. 223–259, 2006.

[2] B. Jennings, S. van der Meer, S. Balasubramaniam, D. Botvich, M. O. Foghlu, W. Donnelly, and J. Strassner, "Towards autonomic management of communications networks," *IEEE Communications Magazine*, vol. 45, no. 10, pp. 112–121, 2007.

[3] N. Agoulmine, S. Balasubramaniam, D. Botvitch, J. Strassner, E. Lehtihet, and W. Donnelly, "Challenges for autonomic network management," in *1st IEEE International Workshop on Modeling Autonomic Communications Environments (MACE)*, 2006.

[4] M. Sloman, "Policy driven management for distributed systems," *Journal of Network and Systems Management*, vol. 2, pp. 333–360, 1994.

[5] J. Strassner, *Policy-Based Network Management – Solutions for the Next Generation*. Morgan Kaufman, 2004.

[6] D. Agrawal, K.-W. Lee, and J. Lobo, "Policy-based management of networked computing systems," *IEEE Communications Magazine*, vol. 43, no. 10, pp. 69–75, 2005.

[7] Y. Cheng, R. Farha, M. S. Kim, A. L. Garcia, and J. W. K. Hong, "A generic architecture for autonomic service and network management," *Computer Communications*, vol. 29, no. 18, pp. 3691–3709, 2006.

[8] J. Strassner, J. Won-Ki Hong, and S. van der Meer, "The design of an autonomic element for managing emerging networks and services," in *IEEE International Conference on Ultra Modern Telecommunications (ICUMT)*, 2009.

[9] L. Baresi, A. D. Ferdinando, A. Manzalini, and F. Zambonelli, "The cascadas framework for autonomic communications," in *Autonomic Communication*, 2009, ch. 6, pp. 147–168.

[10] J. Strassner, N. Agoulmine, and E. Lehtihet, "FOCALE – a novel autonomic networking architecture," *ITSSA Journal*, vol. 3, no. 1, pp. 64–79, 2006.

[11] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[12] J. Boyd, *The Essence of Winning and Losing*, 1995.

[13] O. Dagdeviren and K. Erciyes, "A hierarchical leader election protocol for mobile ad hoc networks," in *Computational Science – ICCS*, 2008, pp. 509–518.

[14] N. Schiper and S. Toueg, "A robust and lightweight stable leader election service for dynamic systems," in *IEEE International Conference on Dependable Systems and Networks (DSN)*, 2008, pp. 207–216.

[15] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The Ponder policy specification language," in *Policies for Distributed Systems and Networks*, 2001, pp. 18–38.

[16] J. D. Case, M. Fedor, M. L. Schoffstall, and J. Davin, "Simple network management protocol (snmp)," United States, 1990.

[17] S. Latré, S. van der Meer, F. De Turck, J. Strassner, and J. Won-Ki Hong, "Ontological generation of filter rules for context exchange in autonomic multimedia networks," in *12th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2010.

[18] S. Davy, B. Jennings, and J. Strassner, "The policy continuum-policy authoring and conflict analysis," *Comput. Commun.*, vol. 31, no. 13, pp. 2981–2995, 2008.

[19] S. van der Meer, "Architectural artefacts for autonomic distributed systems- contract language," in *Sixth IEEE Conference and Workshops on Engineering of Autonomic and Autonomous Systems (EASe)*, 2009, pp. 99–108.

[20] B. Meyer, "Applying "design by contract"," *Computer*, vol. 25, no. 10, pp. 40–51, 1992.

[21] C. Adam and R. Stadler, "Service middleware for self-managing large-scale systems," *IEEE Transactions on Network and Service Management*, vol. 4, no. 3, pp. 50–64, 2008.

[22] C. Reich, K. Bubendorfer, M. Banholzer, and R. Buyya, "A SLA-oriented management of containers for hosting stateful web services," in *Proceedings of the Third IEEE International Conference on e-Science and Grid Computing (E-SCIENCE)*, 2007, pp. 85–92.

[23] Y. Cheng, A. Leon-Garcia, and I. Foster, "Toward an autonomic service management framework: A holistic vision of soa, aon, and autonomic computing," *IEEE Communications Magazine*, vol. 46, no. 5, pp. 138–146, 2008.