

# Evaluation and Comparison of Decentralised Autonomic Computing Solutions

*Tom De Wolf*  
*Tom Holvoet*

*Report CW437, March 2006*



Katholieke Universiteit Leuven  
Department of Computer Science  
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

# Evaluation and Comparison of Decentralised Autonomic Computing Solutions

*Tom De Wolf*

*Tom Holvoet*

*Report CW 437, March 2006*

Department of Computer Science, K.U.Leuven

## **Abstract**

When engineering different decentralised autonomic computing solutions for a certain application there is a lack of a common and usable evaluation and comparison approach. Useful metrics and measurements are mostly domain-specific and are hard to generalise. This paper proposes to use radar charts which allow to apply these domain-specific metrics in an overall evaluation and comparison approach. As such, multiple solutions can be compared with each other and evaluated along multiple dimensions that are important for evaluating decentralised autonomic systems. The approach is illustrated in mobile ad-hoc network management.

**Keywords :** decentralised autonomic computing, radar charts, performance evaluation, comparison of performance.

**AMS(MOS) Classification :** Primary : D.2.8., Secondary : C.4, D.4.8., I.2.11.

# Evaluation and Comparison of Decentralised Autonomous Computing Solutions

Tom De Wolf and Tom Holvoet  
Department of Computer Science, KULeuven  
Celestijnenlaan 200A, 3001 Leuven, Belgium  
{Tom.DeWolf, Tom.Holvoet}@cs.kuleuven.ac.be  
<http://www.cs.kuleuven.be/~tomdw/>

## Abstract

*When engineering different decentralised autonomous computing solutions for a certain application there is a lack of a common and usable evaluation and comparison approach. Useful metrics and measurements are mostly domain-specific and are hard to generalise. This paper proposes to use radar charts which allow to apply these domain-specific metrics in an overall evaluation and comparison approach. As such, multiple solutions can be compared with each other and evaluated along multiple dimensions that are important for evaluating decentralised autonomous systems. The approach is illustrated in mobile ad-hoc network management.*

## 1. Introduction

In autonomous computing, as in any other domain, it is important to be able to evaluate and compare different solutions thoroughly. However, today there is no common and generally usable approach. We need metrics and an approach that allows to evaluate and compare the performance of solutions on multiple characteristics related to autonomous computing. Useful metrics are mostly domain-specific and hard to generalise. However, there is a need for an overall approach to compare and evaluate multiple solutions, possibly in different domains. This is a general problem for autonomous computing and in particular for decentralised autonomous computing, which is the focus of this paper.

Decentralisation is a characteristic of many modern computing systems and implies an increased complexity in managing the system. Autonomous Computing (AC) [10] is essential to keep such systems manageable. The problem is that many decentralised systems make central or global control impossible. For example, the information needed to make decisions cannot be gathered centrally (e.g. ad-hoc networks). As such, AC is only possible when decentralised entities autonomously coordinate with each other to main-

tain the self-\* properties. We denote this kind of AC as *Decentralised Autonomous Computing* (DAC).

This paper proposes an evaluation and comparison approach based on radar charts which allows to apply domain-specific metrics to compare and evaluate multiple DAC solutions in multiple dimensions which are important for (decentralised) autonomous computing. To illustrate this, the approach is applied in detail to a case study of mobile ad-hoc networks. Note that many of the issues and techniques discussed in what follows are not restricted to decentralised autonomous computing but may also be applicable to autonomous computing in general. Previous work on metrics or evaluation approaches for autonomous computing were [1, 11] which only described the challenges and requirements for such approaches and no concrete solution.

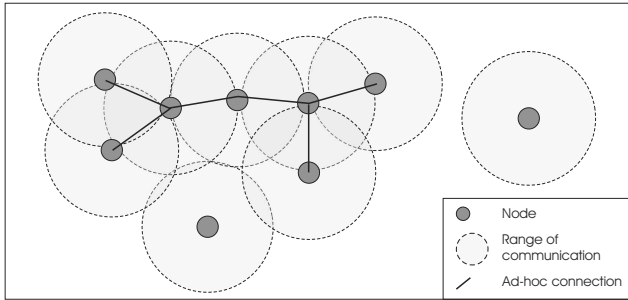
The paper is structured as follows. Section 2 introduces the term Decentralised Autonomous Computing (DAC), the case study used throughout the paper, and the implications of DAC on self-\* properties such as self-healing, self-configuring, self-optimising, and self-protecting. Then, section 3 describes how radar charts allow to evaluate and compare DAC solutions and applies it to the case study. Finally, sections 4 and 5 discuss open issues and conclude.

## 2. Decentralised Autonomous Computing

In this section we introduce the term Decentralised Autonomous Computing, and the implications of decentralisation on self-\* properties such as self-healing, self-configuring, self-optimising, and self-protecting.

### 2.1. What is Decentralised Autonomous Computing?

Decentralised systems are systems for which the control is decentralised of nature. The system behaviour cannot be controlled by a central entity and the system is commonly composed of many entities working together to form a stable structure. Autonomous computing is essential to keep such systems manageable.



**Figure 1. A mobile ad-hoc network where two nodes are not connected to the network.**

Typically [9], each self-\* property in AC is the responsibility of a single autonomous entity (or manager) which possibly controls a hierarchy of autonomic managers to achieve the self-\* property. However, in DAC, the main implication of decentralisation is that no central or global control is possible. Many self-\* properties (e.g. self-configuration, self-optimisation, self-healing, and self-protecting) are achieved by a group of autonomous entities that coordinate in a peer-to-peer fashion. Therefore we introduce the term Decentralised Autonomic Computing:

*Decentralised Autonomic Computing (DAC)* is achieved when a system is constructed as a group of locally interacting autonomous entities that cooperate in order to adaptively maintain the desired system-wide self-\* properties without any external or central control.

Such a system is also called a self-organising emergent system [3]. Self-organisation is achieved when the behaviour is constructed and maintained adaptively without external control. A system exhibits emergence when there is coherent system-wide or macroscopic behaviour that dynamically arises from the local interactions between the individual entities at the microscopic level. The individual entities are not explicitly aware of the resulting macroscopic behaviour; they only follow their local rules. The term ‘macroscopic’ refers to the dynamics of the system as a whole and the term ‘microscopic’ refers to the dynamics of the individual entities within the system. In other words, the self-\* properties are so called macroscopic properties. It is however not mandatory that all self-\* properties become macroscopic properties. For example, self-protection can still be achieved locally by a single entity if the protection of the system is controlled by a single entry point. If however an attack on the system has to be countered by a defensive group-attack on the intruders then self-protection becomes a macroscopic property.

For example, consider mobile ad-hoc networks as a case study that is used throughout the paper. A mobile ad-hoc

network is a local area network or other small network of portable and mobile devices, in which some of the network devices are connected to the network only while in some close proximity to the rest of the network (see Figure 1). The devices or nodes are free to move randomly and organise themselves arbitrarily; thus, the network’s wireless topology may change rapidly and unpredictably.

A goal of managing such a network is the self-\* property to adaptively maintain a certain degree of connectivity within the network so that communication between different nodes remains possible. This management task cannot be achieved by introducing a central controlling entity because the information needed for making management decisions is inherently distributed and decentralised. Also, nodes can leave, join, and fail at every moment. In such a dynamic context it is impossible to aggregate enough information about a network centrally. By the time the information would have been aggregated something has already changed which makes the information obsolete. Therefore a decentralised solution is needed in which a group of autonomous entities residing on the nodes coordinate their local movement in order to maintain an acceptable global connectivity. An individual node is not aware of the overall connectivity of the network. Still, this connectivity emerges from the interactions between the nodes.

## 2.2. Implications on Self-\* Properties

Engineering a decentralised solution to achieve self-\* properties as macroscopic properties has a number of implications on how to interpret each self-\* dimension such as self-healing, self-configuring, self-optimising, and self-protecting. Table 1 gives an overview of the most common self-\* dimensions or properties and indicates which dynamics each one emphasises. The definitions are a combined result from descriptions given in [10, 8, 5, 9]. A prerequisite to achieve each self-\* property is that a system will need advanced feedback control mechanisms to monitor its state and metrics and take appropriate action. Such adaptive actions could mean adjusting parameters, reallocating resources to improve overall utilisation, or complete restructuring of the system configuration.

For DAC, in general, interpretation of these self-\* properties puts more emphasis on decentralisation and the problem-solving power resulting from the interactions in a group of autonomous entities instead of on the internal reasoning of each autonomous entity:

- *Decentralised Self-Healing*: next to healing of individual parts, DAC implies healing of structures that are maintained by the system as a whole. For example, in ad-hoc networks routing of packets can be achieved by a collaborative effort of agents on the nodes which results in the construction of efficient paths through the

**Table 1. The different self-\* dimensions or properties in autonomic computing**

Self-*	Definition	Dynamics	Consequences
Self-Healing	A system is self-healing when it autonomously detects, diagnoses, and repairs problems that occur such as failures of parts of the system and disruptions of the wanted behaviour. A policy-based corrective action will be taken in order to keep the system functioning smoothly.	<i>Failure Dynamics</i> , e.g. agent or subsystem fails, action fails, etc.	As such, the system becomes more reliable and available because day-to-day operations are less likely to fail.
Self-Configuring	A system is self-configuring when system configuration or “setup” occurs autonomously and the system adapts to varying and unpredictable changes by (re-)configuring itself following high-level policies and without disruption of the system behaviour.	<i>Setup-Structural Dynamics</i> , e.g. new policy or goal behaviour, adding/removing components, changes in the environment, new features, software, and hardware, dramatic changes in system characteristics, etc.	As such the service is not disrupted when changes occur and dynamic adaptation helps ensure continuous strength and productivity of the system.
Self-Optimising	A system is self-optimising when it never settles for status quo but continuously looks for ways to optimise its efficiency and performance by autonomously monitoring and tuning the system behaviour.	<i>Normal Dynamics that signals to optimise</i> , e.g. obstacles on routes, ever-changing needs, dynamically changing workloads, priorities of the customer and/or supplier change constantly, etc.	As such, a constant drive to optimality improves efficiency of the system.
Self-Protecting	A system is self-protecting when it autonomously anticipates, detects, identifies, and protects against malicious attacks or cascading failures to maintain overall system security and integrity. It uses early warnings to anticipate and prevent system-wide failures.	<i>Malicious Dynamics</i> , e.g. attacks, symptoms, threats, intrusions, hostile behaviour, viruses, unauthorised access and usage, attack-related component failures, etc.	As such, the system is made less vulnerable and businesses can consistently enforce security and privacy policies.

network [2]. When a certain network connection on a path fails the path has to be healed, again by a collaborative effort of a group of agents.

- *Decentralised Self-Configuring*: for DAC an important mechanism to achieve a desired macroscopic behaviour is the decentralised coordination mechanism used [4] such as digital pheromones, gradient fields, market-based, contract-net, etc. As such, when new agents are added, the system should self-configure so that the new agent is seamlessly incorporated into the coordination process. For example, when a new node is in reach of an ad-hoc network this one should be seamlessly integrated in the process of constructing and maintaining routing paths through the network and the coordinated movement of those nodes to optimise the network connectivity.
- *Decentralised Self-Optimising*: also in DAC the group

of coordinating entities should constantly maintain and seek to optimise their behaviour as a whole. For example, in ad-hoc networks routing paths through the network should never become fixed. The coordination to maintain those paths has to constantly consider better alternatives and optimisations of those paths.

- *Decentralised Self-Protecting*: protection in DAC where the group dynamics is the most important aspect mainly involves protection against malicious agents that want to intrude and participate in the coordination processes. Instead of cooperating they'll try to sabotage the coordination. To achieve this in a decentralised manner often the group of agents also has to coordinate to identify such malicious agents (e.g. trust mechanisms such as tags [4]).

When evaluating and comparing DAC solutions these implications also have an impact on how to evaluate the

performance with respect to autonomic computing. The focus of metrics used has to include the aspect of a group of autonomous entities interacting to reach the required performance as a whole and in a decentralised way.

### 3. Evaluating Decentralised Autonomic Computing Systems

An important step in a new research discipline such as autonomic computing is to *be able to compare different solutions and systems*. Knowing what the self-\* dimensions are for decentralised autonomic computing, this section describes an approach to evaluate and compare how well each dimension is accomplished by different systems.

To capture the essence of autonomic computing - adaptation - we must *introduce change* into the evaluation [1]. Each self-\* dimension is concerned with reacting to change. For example, how does the system cope with faults that are injected (self-healing), with injecting configuration change requests (self-configuration), or with injecting attacks (self-protection). As such, according to [1], the goal is to supplement traditional performance metrics with quantitative and qualitative evaluation results of how well the system adapts to the injected changes. A second issue to incorporate into the evaluation is concerned with the essence of DAC, i.e. *decentralisation of control*. Decentralisation is a number of advantages such as scalability, flexibility, and robustness. Evaluating this is also important.

To capture all different aspects of decentralised autonomic computing the approach in this paper evaluates each one as a different point of view, i.e. *evaluation views*. These are described and applied in section 3.3. First, section 3.1 discusses general issues on evaluating systems and section 3.2 introduces radar charts which are used to visually capture the evaluation views.

#### 3.1. General Evaluation Issues

**Quantitative versus Qualitative.** When capturing the performance of a system with respect to autonomic computing this can be done in two different ways:

- *Qualitative comparison*, is comparing different systems' characteristics or properties on a non-numeric and discrete scale (e.g. categories, levels, etc.). This is often a subjective approach but can have more meaning than a single number.
- *Quantitative comparison*, is comparing different systems' numerical measurements and quantities (hence the name). This is mostly an objective approach but a single number is not expressive enough to evaluate every characteristic or property of the system.

Both complement each other. Therefore, in what follows, both are used according to what seems most appropriate to capture a certain characteristic of a system.

**Starting from the Requirements.** Evaluation of the performance of a system or solution is always related to the requirements for the problem which is solved (e.g. Quality-of-Service metric in [11]). For example, in ad-hoc networks, the throughput of packets is required to be acceptable. Therefore, measuring the throughput is needed when evaluating the performance of the network.

Therefore, each characteristic of a decentralised autonomic system is evaluated with respect to the requirements that apply to that characteristic. For example, consider self-optimisation. In an ad-hoc network the following requirements have to be incorporated to evaluate self-optimisation:

- *Efficient and adaptive routing of packets.* In a network the communication packets have to be routed from source to destination as efficiently as possible. Therefore, self-optimisation of routes is required.
- *Optimise fraction of nodes in largest network component.* At each moment in time the system has to maintain a network structure in which the number of nodes that are part of the largest connected network component is as high as possible.
- *Maintaining a degree of connectivity among nodes.* Related to the previous requirement, the number of connections that each node has to other nodes has to reach a certain degree. Optimisation towards the (configured) ideal number of connections is needed.
- *Optimise when network usage load changes.* Most of the performance related requirements have to be optimised when the network usage load changes dynamically and constantly over time.

Some other example requirements to which the evaluation that follows refers are:

- *Self-configure when new nodes enter or leave the network.* Those new nodes have to get linked to their neighbours and be incorporated into the coordination processes that achieve routing and connectivity.
- *Self-configure when usage patterns change dramatically.* In contrast to optimisation for constant changes in the usage load, a dramatic change in the usage pattern may require a whole reconfiguration of the network to be able to handle this.
- *Self-configure to enforce a new policy.* For example, instead of allowing nodes to use all available bandwidth, a new policy could restrict the bandwidth usage per node to 10 percent of the available bandwidth.

- *Self-heal routing paths* when they are destructed due to failure of one or more nodes/connections.
- *Self-heal failing re-configurations*. For example, when the introduction of a new node fails this should be anticipated autonomously.
- *Self-protect against denial-of-service attacks* on nodes in the network.
- *Self-protect against malicious nodes*. The network should identify and exclude maliciously behaving nodes that want to enter the network. It is possible that such protection can be handled at a single entry point which gives each node authorisation. However, in a decentralised situation, such as ad-hoc networks, it is also possible that malicious nodes can only be identified and eliminated by a coordinated defensive group-attack of the other nodes.
- *High degree of decentralisation*. In ad-hoc networks, the management control should be decentralised as much as possible due to the high dynamics of the topology and distributed nature of control information.

**Finding Concrete Metrics.** Metrics or measurements to evaluate the performance of a solution are always domain or problem specific. The reason is that an evaluation of a system is always related to the problem specific requirements which is discussed earlier. As such, it is hard or even impossible to generalise to an overall usable metric for autonomic computing. This paper accepts this reality and proposes to apply the domain-specific metrics and integrate them into an evaluation and comparison approach based on radar charts. As a consequence, multiple solutions, possibly in multiple problem domains can be evaluated in a similar and comparable manner. In what follows, the approach is applied to mobile ad-hoc networks and many metrics are specific for that problem domain.

For decentralised autonomic computing, metrics have to capture the performance of so called macroscopic properties. With respect to that context some guidelines can be given on what are promising metrics to look at:

- *Averages*: An average is in many cases a metric worth considering for properties of the system as a whole. Many macroscopic properties are only the average of a local property of entities in the system. The system is structured as a group of entities. This often implies that measuring something of the group as a whole involves integration of properties of the individual entities in a single measurement. For example, in ad-hoc networks the degree of connectivity can be measured by taking the average over all nodes of the number of connections to other nodes.

- *Entropy*: A metric that is often used for macroscopic properties is the so called Entropy [6]. This metric originates from Shannon Entropy [13] in information theory, and is defined as follows:

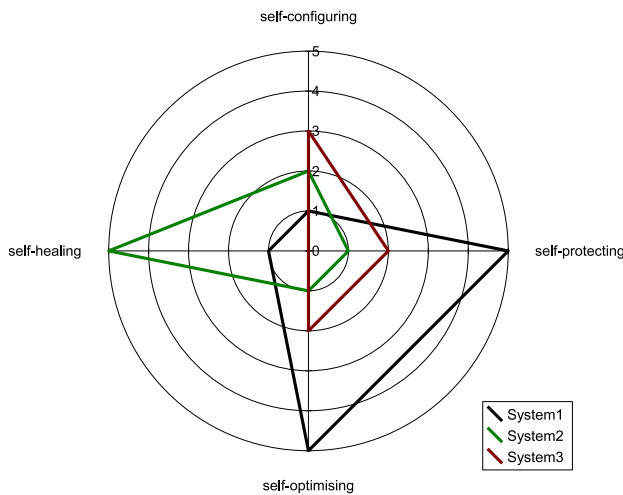
$$E = \frac{-\sum_n^N p_n \cdot \log p_n}{\log N} \quad (1)$$

where  $p_n$  is the probability that state  $n$  out of all  $N$  states occurs. The denominator  $\log N$  normalises to the range  $[0, 1]$ . An entropy close to 1 indicates equal probabilities, close to 0 indicates a non-equal situation. Entropy can especially be used for spatial distribution-like properties. For example, consider an ad-hoc network on unmanned air vehicles for a surveillance system with a number of equally important regions in space to cover. An equal spatial distribution of the nodes (i.e. UAVs) has to be maintained between those regions in order to cover and observe each region equally well. In terms of an Entropy measure, for each region there is the state that a node is in that region. Then  $p_n$  is the probability of finding a node in a region  $n$ .  $E$  is close to 1 indicates an equal distribution, otherwise  $E$  is closer to 0. In general, Entropy is useful for properties related to a number of states with probabilities and where a measure of equality or inequality of probabilities is useful. As such, measuring how focussed a group of agents move in space can be done by considering for each agent the different locations to choose from as the states in the entropy. The probabilities for choosing each location are considered as the probabilities in the entropy. An entropy of 1 indicates a random choice and 0 indicates a strong preference for some locations.

- *Distances*: To measure if a self-\* property is achieved, the difference between the current state and the desired state can be measured, i.e. a ‘distance’-metric. Consider the self-\* property to achieve a certain level of performance, a specific spatial shape, or other structures. Measuring this by calculating the distance between the current state and the desired state can be a good metric. However, how this distance metric is constructed remains a challenge.

### 3.2. Visualisation with Radar Charts

Radar charts [12, 7] (also called spider charts, polar charts, or kiviart charts) are a form of a graph that allows a visual comparison between several quantitative or qualitative aspects of a situation, or when charts are drawn for several situations using the same axes, a visual comparison between the situations may be made. A radar chart shows one axis for each aspect of a situation. Close to the center are the low values for the axis, and near the edge of the



**Figure 2. Sample Radar Chart**

graph the high values are located. Such charts often show a current situation compared to some target.

In the context of software systems, a certain solution or system can be considered as one situation and as such a radar chart offers a graphical display of the differences between actual and ideal performance and is useful for defining performance and identifying strengths and weaknesses of different solutions. Performance can depend on several aspects each represented in the radar chart.

For example, consider autonomic computing. If self-healing, self-configuring, self-optimising, and self-protecting are considered as the dimensions on which one measures the autonomic performance of a system then a possible radar chart could be the one shown in figure 2. Each self-\* dimension is an axis on the chart and the rating on each axis depends on how good the system is evaluated for that dimension. As such multiple systems can be compared with each other as shown in figure 2: system1 is clearly better in self-protection and self-optimisation, while system2 is an expert in self-healing and system3 does not do self-healing but is best at self-configuring.

There are a number of advantages of radar charts:

- They make good use of the human ability to spot symmetry (or rather un-symmetry). As such easy and quick evaluation is possible.
- They are very useful when a relatively small number of samples need to be compared and the number of variables or factors to look at is large.
- In a gap analysis situation, the ‘desirable state’ and the ‘present state’ data can be plotted on the same chart to demonstrate graphically the gap between them.

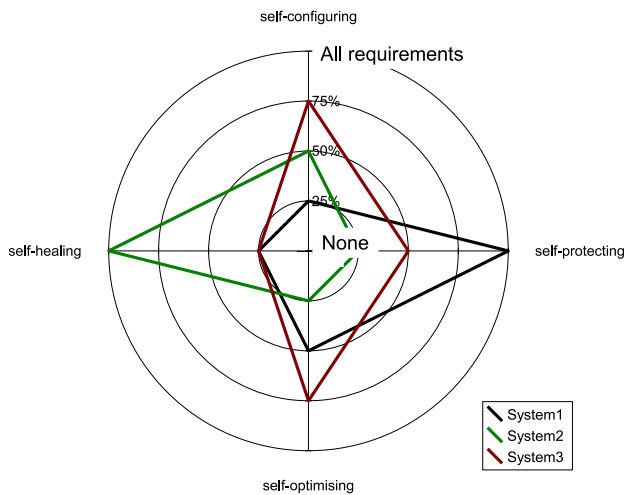
- In a change analysis situation, the ‘before’ and ‘after’ results can be graphically compared.
- The (numerical) scales can differ between two axes.

### 3.3. Evaluation Views

From the beginning of section 3 and from the discussion in section 3.1 it is clear that there are different aspects or characteristics to be included in the evaluation and comparison of decentralised autonomic computing systems or solutions. Also, [1] and [11] give a number of dimensions, issues or characteristics that are important in evaluation. To accommodate this, this section introduces multiple *evaluation views*, visualised by radar charts, that each capture an evaluation of the system from a specific point of view. The different characteristics, dimensions, and requirements determine which views are useful for a system and how the views are constructed. Domain-specific metrics are to be used in these evaluation views. In what follows each view is described and applied to the ad-hoc network case.

**Level/Degree of Autonomy.** In autonomic computing it is important to be able to state how autonomic a system is, i.e. the degree of autonomy [11]. There are two approaches to evaluating how autonomic a system is:

- *Coverage of Self-\* Requirements.* For each self-\* property such as self-healing, self-configuring, self-protecting, and self-optimising there are a number of related requirements (see section 3.1 for an overview in the case of ad-hoc networks). Requirements are independent of the solution so multiple solutions or systems can cover the requirements for a problem in different ways. The more self-\* requirements covered, the more autonomic the solution is. As such this implies a first evaluation view shown in figure 3. The radar charts show that System1 achieves all self-protecting requirements and only 25 percent of the self-configuring requirements, while System3 achieves 75 percent of the self-configuring requirements and self-optimising requirements and only 50 percent of the self-protecting requirements.
- *Steps towards Full Autonomic Computing.* As stated in [1], to handle partially autonomic systems an autonomic computing evaluation should provide useful metrics that can quantify the steps towards a fully autonomic system. However, it is probably hard to quantify the degree of autonomy. The steps towards a fully autonomic system are typically first monitoring autonomously, then analysing what is monitored, after that planning of suitable actions, and finally executing

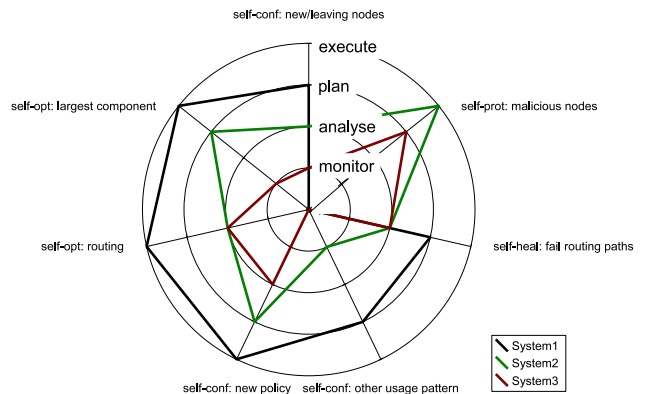


**Figure 3. Coverage of Self-\* Requirements**

the actions autonomously (i.e. MAPE: Monitor, Analyse, Plan, Execute). Determining on what autonomic level a system is requires a qualitative assessment of human involvement [1].

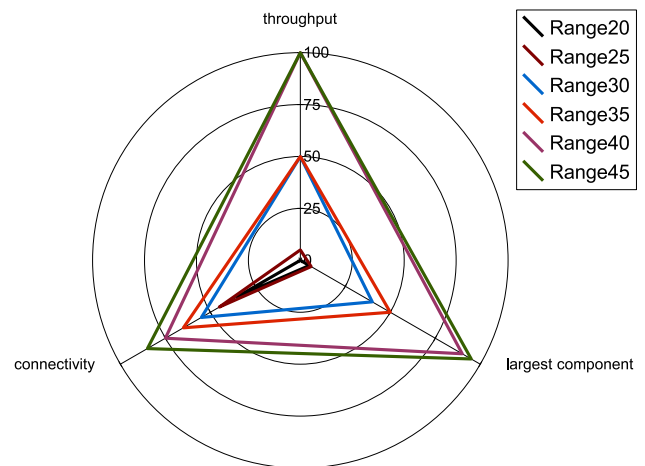
Similar to figure 3, each self-\* property is situated at a MAPE-level. However, because there are multiple requirements concerning each self-\* property, it is better to put an evaluation of each of those requirements as a dimension on the radar chart and determine if for that requirement the solution has reached the monitoring, analysing, planning, or executing level. This avoids problems such as on which level is self-configuring if only one out of two self-configuring requirements has reached the executing level and the other the monitoring level. Figure 4 shows that system1 is more autonomic than system2 and system3. System1 is already autonomously executing actions for a number of self-\* requirements where system2 and system3 only monitor, analyse, or plan. On the other hand system1 completely fails to self-protect against malicious nodes.

**Performance of Autonomic Response.** In any case one can use the domain-specific performance metrics as dimensions on a radar chart and as such compare different solutions quantitatively with each other. For example, in ad-hoc networks, the average throughput, the average largest component size, and the average degree of connectivity, could be performance measures to put on a radar chart as shown on figure 5. Figure 5 compares the same solution for an ad-hoc network but with different values for the communication range of the nodes. This shows that for ranges lower than 30-35 the performance is very low and above 30-35



**Figure 4. Level of AC for ad-hoc networks**

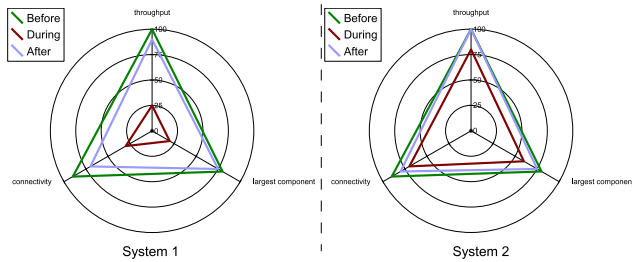
the performance is good. A similar radar chart can be made comparing different systems instead of parameter values.



**Figure 5. Parameter Dependent Performance**

However, with respect to autonomic computing another kind of comparison is more interesting, shown in figure 6: the quality of the response (how well it accomplishes the necessary adaptation) [1]. Instead of plotting different systems on one radar chart, one can plot the performance of one system (which is domain-specific) at three different moments in time: before an autonomic response is started, during the execution of the autonomic response, and after the autonomic response has completed. As such, a system can be evaluated based on how well it returns to the starting performance and/or how the performance degrades during adaptation. Also, comparing multiple of such autonomic performance views of different systems is possible. For example, figure 6 shows that system 1 and 2 almost

equally adapt their performance back to the situation before the change. But system 2 is better in maintaining the performance as much as possible during the adaptation.



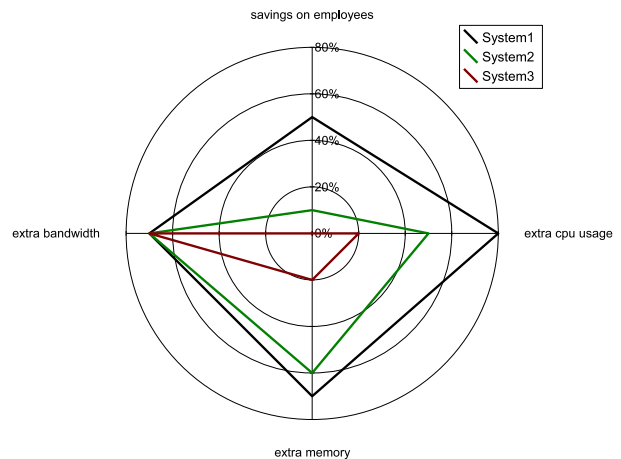
**Figure 6. Autonomic Response Performance**

**Cost of Autonomic Response.** Autonomic computing also demands extra resources to support the autonomic responses of the system. Calculating this overhead of extra autonomic activity is the cost of autonomic computing [11, 1]. There are a number of cost-related metrics such as business cost, i.e. savings on administrator personnel; or overhead cost, i.e. cost of extra autonomic activity inside the system (cpu usage, storage, ...). Again, it depends on the problem domain what is considered a “cost” in the system.

The cost view on evaluating autonomic computing can be a radar chart such as the one shown in figure 4 except there are no MAPE-levels on the axes but for each requirement the value of a metric that calculates the total cost of achieving that requirement autonomously is plotted. Another possibility is to use the cost metrics themselves as dimensions or axes on the radar chart as shown in figure 7. Each axis depicts the extra percentage needed (or saved in the case of personnel savings) compared to the same system without the autonomic capabilities. Especially the extra bandwidth cost is interesting for decentralised autonomic computing because achieving something in a decentralised way requires overhead in the form of coordination and thus information exchange and bandwidth usage.

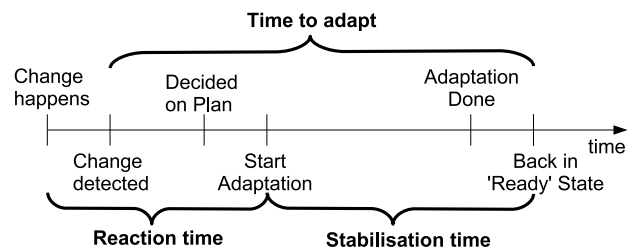
**Speed of Autonomic Response.** Another evaluation view considers how fast the system adapts, i.e. the speed of the system’s autonomic response. Figure 8 shows different time measurements that are interesting to look at:

- *Time to adapt* [11]: time taken between identification that a change is required until the change has been effected safely and the system moves to a ready state.
- *Reaction time* [11]: time between when an environmental element has changed and the system recognises that change, decides on what reconfiguration is necessary and gets ready to adapt.



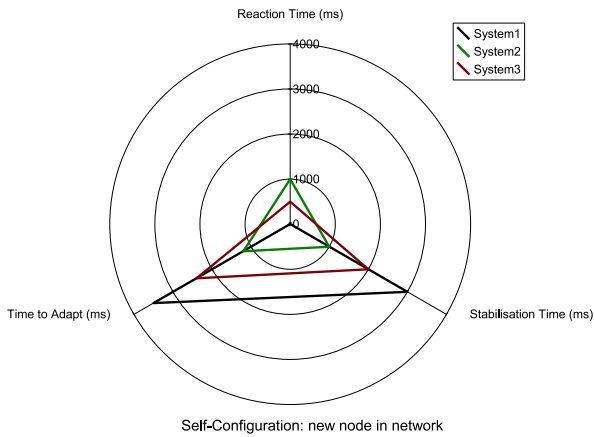
**Figure 7. Cost of Autonomic Response**

- *Stabilisation Time*: time between when the system starts adaptation and the moment where the system is back in a ready state or in other words a stable state as envisioned by the adaptation. Note that in decentralised systems an adaptation often has to propagate through the system to a large part of the autonomous entities before one can speak of a ready state.



**Figure 8. Different Time or Speed Measures**

Again, a radar chart can be made with the self-\* requirements as axes (similar to figure 4) and one of the time measurements then supplies the values on the axes. Or the time measurements can be considered as axes themselves and as such a radar chart can be given for each self-\* requirement comparing multiple systems, e.g. figure 9: for re-configuring autonomously when a new node is added System2 needs less time to adapt and stabilise but takes longer before it reacts to a change compared with System1 and System3. A radar chart for each system comparing the times for multiple self-\* requirements is also possible.

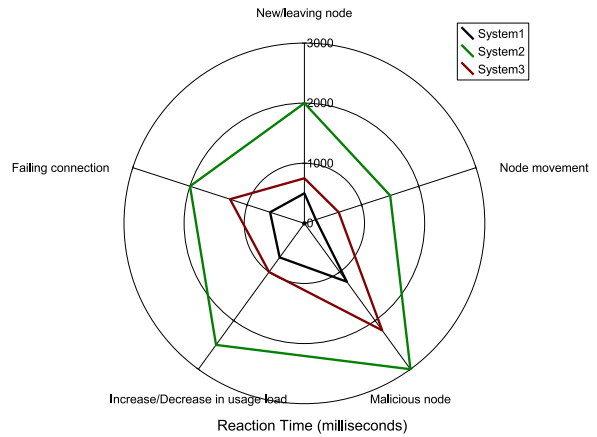


**Figure 9. Speed of Autonomic Response**

**Sensitivity to Changes.** A long reaction time, as mentioned earlier, is not necessarily worse. If a system reacts too fast to a change, meaning it is highly sensitive to such changes in its environment, it can potentially cause the system to be constantly changing configuration (oscillating) and not getting on with the job it has been assigned [11]. In decentralised autonomic computing it is important to note that a single entity in the system can react very fast but the group as a whole may react very slow or even not at all. As such, a system is considered to react only if the group as a whole gets ready to coordinate an adaptation.

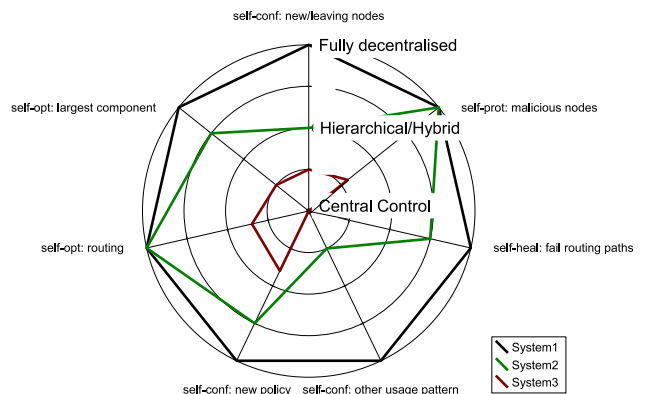
Constructing a radar chart where there is an axis or dimension for each possible change in the system and on which the reaction time for that change is plotted allows to evaluate the sensitivity of a system to changes in more detail. An example is shown in figure 10. Note that reaction time may not be the best measure of sensitivity. Other measures can be used in figure 10 as well. The figure shows that System1 is very sensitive to all changes in the network which might imply that it is not working properly because it is constantly reconfiguring itself. On the other hand, System2 is very insensitive which might imply that it reacts too late on changes. The intermediate solution of System3 might be the better one because it waits just long enough to allow a change to disappear automatically before starting an adaptation. As such, the system oscillates less often.

**Degree of Decentralisation of Autonomic Response.** As stated in [11], the granularity of the autonomic response is an important issue when comparing autonomic systems and especially when comparing decentralised autonomic systems. In other words, the degree of decentralisation of the autonomic control is important. Fine-grained components with specific adaptation rules (i.e. decentralised control) will be highly flexible and perhaps adapt to situations



**Figure 10. Sensitivity to Changes**

better, although this may cause more overhead in terms of the global system (especially coordination overhead). On the other hand, ticker-grained components do not have this overhead but due to the centralisation of control it can be less flexible, robust (single point of failure) and scalable. As such in decentralised autonomic computing it is important to know to what degree a system is controlled locally in an decentralised way or globally by a central controller.



**Figure 11. Degree of Decentraliation**

Figure 11 shows an evaluation view that evaluates this. Each self-\* requirement is typically achieved by a different control approach and therefore they are a different axis on the radar chart. System1 is clearly a fully decentralised solution for all requirements. Which is in strong contrast with System3 using almost complete central control. System2 has centrally solved properties, some are solved decentralised, and some employ a hybrid/hierarchical form of control. Decentralisation is hard to capture quantitatively.

Therefore a qualitative assessment of the system determines its position on the decentralisation scale.

#### 4. Discussion and Open Issues

As the previous sections show the semantically very different evaluation views are represented by one charting approach, i.e. radar charts. As such, radar charts are a very flexible way of visualising an evaluation of multiple properties for multiple systems, multiple domain-specific metrics, multiple conditions (e.g. parameter values), etc in decentralised autonomous computing. However, there are a number of open issues. First of all, in many of the proposed evaluation views there are still concrete measurements to be found. However, this will always be the case because, as mentioned before, in our opinion measurements of performance will always be domain-specific and as such no general and usable metrics exist. As shown, radar charts allow to use those measurements as an integral part of a more general evaluation approach. A second issue or remark is that not all possible evaluation views are given here. More are possible due to the flexibility of the radar chart approach. As such, everyone can adapt this approach to suit their evaluation or comparison task. And finally, a third open issue concerns a quantitative metric or measurement that states how autonomous a system is. To get a global metric on how autonomous a system is, the challenge is to synthesise multiple sub-metrics into an overall measure of autonomy, and calibrate scores across different systems [1]. Considering a more general evaluation view (e.g. degree of autonomy) and calculating the size of the surface between the lines plotted for each system is a possibility. But even then a lot of other characteristics are not captured in this metric. In our opinion it is infeasible to find an overall metric. Such a metric would be less expressive compared to a detailed evaluation with different evaluation views.

#### 5. Conclusion

In autonomous computing, as in any other domain, it is important to be able to evaluate and compare different solutions thoroughly. However, today there is no common and generally usable way to achieve this. Especially because metrics are always domain-specific and related to the requirements of the system. An approach to evaluate and compare (decentralised) autonomous computing systems by visualisation via radar charts is proposed.

Using a visualisation in different evaluation views that each capture an important aspect of decentralised autonomous computing allows easy comparison of multiple systems, for multiple requirements, with multiple metrics, and from multiple points of view. Evaluation views for the degree of autonomy, the performance, the cost, the speed, the

sensitivity of the autonomous response, and degree of decentralisation are described in detail. However, concrete measurements still have to be found for each application domain in which decentralised autonomous computing is applied.

**Acknowledgements.** This work is supported by the K.U.Leuven research council as part of the concerted research action on Autonomous Computing for Decentralised Production Systems (project 3E040752)

#### References

- [1] A. B. Brown, J. Hellerstein, M. Hogstrom, and T. Lau. Benchmarking autonomous capabilities: Promises and pitfalls. In *Proceedings of the International Conference on Autonomous Computing (ICAC'04)*. IEEE Press, 2004. (poster).
- [2] G. D. Caro and M. Dorigo. Two ant colony algorithms for best-effort routing in datagram networks. In *Proceedings of the Tenth IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'98)*, pages 541–546. IASTED/ACTA Press, 1998.
- [3] T. De Wolf and T. Holvoet. Emergence and Self-Organisation: a statement of similarities and differences. In *Proceedings of the Second International Workshop on Engineering Self-Organising Applications*, pages 96–110, July 2004.
- [4] T. De Wolf and T. Holvoet. *Autonomous Computing: Concepts, Infrastructure, and Applications*, chapter A Taxonomy for Self-\* Properties in Decentralised Autonomous Computing. CRC Press, to appear in 2006.
- [5] A. G. Ganek and T. A. Corbi. The dawning of the autonomous computing era. *IBM Systems Journal*, 42(1), 2003.
- [6] S. Guerin and D. Kunkle. Emergence of constraint in self-organizing systems. *Nonlinear Dynamics, Psychology, and Life Sciences*, 8(2):131–146, April 2004. (available at <http://www.redfish.com/>).
- [7] R. L. Harris. *Information Graphics: A Comprehensive Illustrated Reference*. Oxford University Press, 2000.
- [8] IBM. *Autonomous Computing Manifesto: IBM's Perspective on the State of Information Technology*. IBM, 2001. (online at <http://www.research.ibm.com/autonomic/manifesto/>).
- [9] IBM. An architectural blue-print for autonomous computing. White Paper, Third Edition, 2005.
- [10] J. O. Kephart and D. M. Chess. The vision of autonomous computing. *IEEE Computer Magazine*, 36(1):41–50, January 2003.
- [11] J. A. McCann and M. C. Huebscher. Evaluation issues in autonomous computing. In H. Jin, Y. Pan, N. Xiao, and J. Sun, editors, *Proceedings of the GCC 2004 Workshops*, volume 3252 of *Lecture Notes in Computer Science*, pages 597–608, Berlin, 2004. Springer Verlag.
- [12] N. B. Robbins. *Creating More Effective Graphs*. Wiley, 2005.
- [13] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, 1948.