Evaluation issues in Autonomic Computing

Julie A. McCann, Markus Huebscher
Department Of Computing, Imperial College London
{jamm, mch1} @doc.ic.ac.uk

Abstract

Autonomic Computing is a concept that brings together many fields of computing with the purpose of creating computing systems that are reflective and self-adaptive. In this paper we draw upon our experience of this field to discuss how we can attempt to evaluate autonomic systems. By looking at the diverse systems that describe themselves as autonomic, we provide an introduction to the concepts of Autonomic Computing and describe some achievements that have already been made. We then discuss this work in terms of what is necessary to evaluate and compare such systems. We conclude with a definitive set of metrics, which we believe are useful to evaluate autonomicity.

1. Introduction

Autonomic computing is generally considered to be a term first used by IBM in 2001 to describe computing systems that are said to be self-managing [1]. However the concept of self-management and adaptation in computing systems has been around for some time. The event of the combination of object-oriented programming paralleled with component-based software engineering essentially paved the way toward autonomic computing [49]. That is, it can be argued that without the concepts of dynamic re-binding of components a system cannot effectively reconfigure itself to adapt to improve its service.

When reviewing the current state-of-the art in autonomic systems, the concept of self-management usually groups into having four basic properties: self-configuration, self-optimization, self-healing and self-protection. Here is a brief description of these properties (for more information, see [1] and [2]):

- *Self-configuration*: an autonomic computing system configures itself according to high-level goals, i.e. by specifying what is desired, not necessarily how to accomplish it. This can mean being able to install itself based on the needs of a given platform and the user.
- *Self-optimization*: an autonomic computing system optimises its use of resources. It may decide to initiate a change to the system *proactively* (as opposed to reactive behaviour) in an attempt to improve performance.
- *Self-healing*: an autonomic computing system detects and diagnoses problems. What kinds of problems are detected can be interpreted broadly: they can be as low-level as a bit-error in a memory chip (hardware failure) or as high-level as an erroneous entry in a directory service (software problem) [3]. If possible, it should attempt to fix the problem, for example by switching to a redundant component or by downloading and installing software updates. However, it is important that as a result of the healing process the system is not further harmed, for example by the introduction of new bugs or the loss of vital system settings. Fault-tolerance is an important aspect of self-healing. Typically, an autonomic system is said to be *reactive* to failures or early signs of a possible failure.
- *Self-protection*: an autonomic system protects itself from malicious attacks but also from end users who inadvertently make software changes, e.g. by deleting an important file. The system autonomously tunes itself to achieve security, privacy and data protection. Thus, security is an important aspect of self-protection, not just in software, but also in hardware (e.g. TCPA [23]). A system may also be able to anticipate security breaches and prevent them from occurring in the first place.

Self-management requires that a system monitor its components (internal knowledge) and its environment (external knowledge), so that it can adapt to changes that may occur, which may be known changes or unexpected changes where a certain amount of artificial intelligence may be required.

However, there is no agreed definition of what an Autonomic system is, their evaluation and moreover comparison, is difficult. Furthermore the very emergent nature of such systems adds further complexity to the evaluation of such systems. This paper is an attempt

to look at Autonomic computing and try to highlight areas, which can be used to compare performance and derive some form of metrics.

The structure of this paper is as follows. Initially, in section 2 we survey the area of Autonomic computing to attempt to build a map of the subject. To this end we provide an introduction to the concepts of Autonomic Computing and describe some research that is taking place in various fields of computing and some achievements that have already been made, section 3. After that in section 4, we concentrate on research in the field of software engineering and describe projects that focus on adding autonomic behaviour to software systems. Finally, in sections 5 and 6 we combine this work together with a discussion on performance evaluation and benchmarking, taking into account our experiences of measuring autonomic systems and provide some initial ideas on how such systems can be compared.

Company	Title	Salient feature
IBM	eLiza	servers and mainframes respond to unexpected changes and system components' failure autonomously [3].
	LEO	autonomic query optimiser for IBM's DB2 database systems [38]. LEO monitors queries as they execute and compares its estimates of the cost for each step in the query execution plan with actual results. The detection of estimation errors can also trigger reoptimisation of a query in mid-execution.
	Blue Gene/L (BG/L),	IBM's a supercomputer to be released in 2005, will implement self-healing and self-management [3] using application programmers placing checkpoints in code.
	IceCube	The server tracks its components' health and maximum capacity. It can then autonomously distribute data among the available nodes. Failed nodes (bricks) are worked around automatically, [17]
Sun	Jini network	allow Systems experiencing failure of a component could find components on the network with the required functionality that are still functioning and then reallocate resources to them autonomously
	Grid Engine Enterprise Edition	Deadline policies can be used to make sure that projects nearing the deadline receive a greater share of resources. Then share based policies consider the accumulated resource usage of a user, so that if a user "over-uses" resources, then the grid will lower the entitlement of that user to resources for a certain period of time [39].
HP	Superdome	adding self-healing properties to replace a processor if there are too many errors to automatically fix

Figure 1. Sample Commercial Autonomic Effort

2. Autonomic computing today

The ideas behind autonomic computing are not new. In fact, it is possible to find some aspects of autonomic computing already in today's software products [2]. For instance, Windows XP optimises its user interface (UI) by creating a list of most often used programs in the start menu. Thus, it is self-configuring in that it adapts the UI to the behaviour of the user, although in a fairly basic way, by monitoring what programs are called most often. It can also download and install new critical updates without user intervention, sometimes without restarting the system. Therefore, it also exhibits its basic self-healing properties. DHCP and DNS services allow devices to self-configure to access a TCP/IP network (albeit in a limited way, as many settings such as web proxy server addresses or VPN security settings must still be entered manually). PCs on a LAN can discover other devices, such as printers, and install their drivers automatically (under the right conditions). Further many consider database query optimisers an early form of autonomic computing [4].

However our definition of what we mean by autonomic computing is that of a self-adaptive system as opposed to an adaptive system. Therefore, here standard query optimisers would not be considered as providing autonomicity. However if while a query was running and the DBMS was monitoring the query's execution and deciding on a different query plan, then we would consider that autonomic. Nevertheless, we realise that the boundary from adaptive to self-adaptive systems is fuzzy.

3. Why Autonomic computing

In trying to understand how to evaluate an autonomic system one much understand the reason we would want such a system. This allows us to compare whether or not the objective has been met.

The main reason for large blue-chip companies, like IBM, being interested in autonomic computing is the need to reduce the cost and complexity of owning and operating an IT infrastructure [4]. In particular, there is a need to alleviate the complexity with which system administrators of IT services are faced today. The aim is to allow administrators to specify high-level policies that define the goals of the autonomic system, and let the system manage itself to accomplish these goals. At present, system administrators must tweak hundreds of

settings and often spend weeks before getting a system to run optimally. Autonomic systems are also faster at adapting to changes to the environment, e.g. by distributing its resources differently when a critical-project requires more CPU processing power. Furthermore, as information systems in enterprises grow larger, it is becoming increasing difficult to identify a failure in the system and repair the affected component quickly, as large systems are heterogeneous and no single person knows the entire system. Examples of such systems can be found in Figure 1.

Autonomic behaviour is a topic that has found its way in many other computing fields in particular adhoc networking [52]. For example, Liu and Martonosi [22] discuss the problem of propagating software updates in a wireless network of devices that are spread over a large area and are not all reachable from a base station. Sensors co-operate to propagate software updates to the entire network of sensors, but at the same time they must optimise energy consumption, because of tight energy constraints. Further due to the autonomous nature of NASA's DS1 (Deep Space 1) mission and the Mars Pathfinder [24],[18] some self-adaptation is required. That is, as mission control cannot rapidly send new commands to a probe, it must quickly adapt to extraordinary

Group	Domain	Main characteristics	Refs
Multi-agent systems			
Kuo-Ming, James, Norman	Framework for multi-agent systems	Communication middleware based on CORBA for monitoring and cooperation	[19]
Sterritt, Bustard	Autonomic components	Heartbeat or pulse monitor for monitoring	[26]
Georgiadis, Magee, Kramer	Architectural constraints for self-organising components	Self-organising components with a global view expressed as architecture description	[25]
Kumar, Cohen	Adaptive Agent Architecture	Broker agents used as to provide fault-tolerance to overlying problem-solving agents.	[27]
Bigus et al.	ABLE agent toolkit	Framework for building multi-agent systems. Working on including autonomic agents.	[36]
Architecture design-based autonomic systems			
Garlan, Schmerl	Architecture model-based adaptation for autonomic systems	Probes, gauges for monitoring running system, architecture manager implements adaptive behaviour, based on architecture-model of system.	[7] [8][9] [10]
de Lemos, Fiadeiro	Architecture for fault-tolerance in adaptive systems	Components considered as black-boxes.	[15]
Dashofy, van der Hoek, Taylor	Framework for architecture-based adaptive systems	xADL 2.0 architecture description language, c2.fw development framework.	[11] [12]
Valetto, Kaiser	Adding autonomic behaviour to existing systems	Autonomic behaviour as a distributed multi-agent infrastructure called Workflakes.	[28] [29]
Hot swapping components			
Rutherford et al.	Reconfiguration in EJB model	BARK tool as an extension to EJB to support component replacement.	[33]
Whisnant, Kalbarczyk, Iyer	Model for reconfigurable software	Adaptivity through replacement of bindings between operations and invoked code blocks.	[34]
Appavoo et al.	Hot-swapping at OS level	High-performance hot-swapping of fine-grained components in K42 OS.	[35]
Kon, Campbell et al.	Reflective middleware	DynamicTAO, a middleware for dynamically reconfigurable software.	[40]
G. S. Blair et al.	Reflective middleware	OpenORB, reflective middleware for self-healing systems.	[41]

Table 1: Summary of reviewed research in software architectures for autonomic computing.

situations, therefore it is important that a probe be able to make decisions and carry them out on its own.

4. Software architectures for autonomic computing

The autonomic research activities in software systems can broadly be categorised into four areas: monitoring of components, interpretation of monitored data, creation of a repair plan (i.e. an adaptation of the system), and execution of a repair plan. Based on this, we choose to group the approaches to autonomic computing systems orthogonally into two categories: intelligent multi-agent systems and architecture design-based autonomic systems. However, the two approaches have common concepts it is sometimes difficult to place a research project in one particular category.

Table 1 shows a summary of the research reviewed in this section.

4.1. Multi-agent systems

Complex autonomic systems that are not composed of a single self-managing component can be built using intelligent agents (for information on multi-agent systems, see [5]). Every agent has its own goals, which drive its decisions. An agent in an autonomic system is *proactive, and* possesses *social ability*. The latter can potentially lead to instabilities of the overall system due to the chain reaction of agents instructing other agents to change behaviour[1]. A difficult talk is also to define the individual goals of the agents such that the desired global goal is accomplished [1]. In an autonomic system, we want to be able to provide goals in the form of high-level notions, and expect the agents themselves to determine what behaviour is necessary to reach them.

Wise et al. [37] propose a top-down hierarchical coordination model for agent applications, in the form of their visual process language Little-JIL. A task is divided into steps, and each step can further be divided into sub steps. A step can then be assigned to an execution agent, which keeps an agenda of tasks to complete.

Although in multi-agent systems each component exhibits its own autonomic behaviour, there is usually a clean separation between the conventional component that performs a task and the autonomic manager which implements self-management around it. Figure 2 (based on a figure from [26]) shows a general diagram for an autonomic agent. One example is [20] known as the BDI methodology. However, in some systems the autonomic

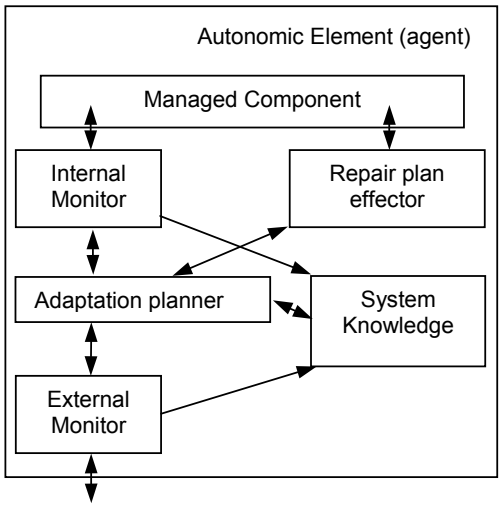

Figure 2: An autonomic agent

components are inseparable from the main application logic in the agent.

Compared to the architecture design-based approach, adaptive multi-agent systems have an innate distributed architecture. With no centralised monitoring infrastructure, agents must monitor themselves (*internal monitor*) but also other agents (*external monitor*). External monitoring can be achieved proactively by having each agent send its *heartbeat* or *pulse* regularly on an autonomic signal channel that other agents send and listen on [26]. The heartbeat provides a summary of the state of an agent to other agents responsible for monitoring that state. Because the autonomic signal channel connects agents that are not necessarily on the same physical system, but could very well be peer-to-peer or networked, there is the danger here that external monitoring activity may flood the autonomic signal channel with lots of traffic. Thus, care must be taken in designing the monitoring protocol. The heartbeat approach has already been used by the Open Grid Services Architecture (OGSA) [31] and by NASA on its Deep Space 1 (DS1) mission [32] (although in NASA's case it is used in a different context). In both cases the heartbeat is a compact message that provides very limited information about the monitored component.

Georgiadis et al show how components can self-configure their interactions in compliance with an overall architectural specification expressed using the Alloy language [25],[16]. Each component has a view of the entire system maintained by the component manager. They measure the elapsed time from the moment a new component is inserted into the system until the moment when all components have the new consistent view of the system. Because of message broadcasting, this time increasing roughly linearly in the number of nodes in the system.

Kumar and Cohen [27] show with experimental data how a team of broker agents can

recover when a broker agent gets disconnected from the rest of the system. Again Broker agents share the same global knowledge of the system, and therefore when a broker agent discovers that another agent has been disconnected, it shares this information with the rest of the team.

Bigus et al. [36] are extending their ABLE agent platform to support autonomic agents to reduce the system administrator workload. ABLE agents are built on top of Enterprise JavaBeans. They use sensors to collect monitoring data and effectors to perform resulting actions on an application. An important aspect thereof is the ability of an intelligent autonomic agent to maintain a model of the external environment and its own components.

4.2. Architecture design-based autonomic systems

In the architecture-based approach, the individual components are not per se autonomic. Instead, the infrastructure that handles the autonomic behaviour of the system uses an architectural description model of the running system (which is not autonomic itself) to monitor the running system, reason about it and determine appropriate adaptive actions. The adaptivity infrastructure is typically clearly separated from the running system.

First of all, an architecture model is used to design the system (as is often the case with software development). In essence, an architecture model can be considered a graph of interacting components [6]. The nodes of a graph are called *components*, a general concept, and what a component actually is depends on the application. Often in research, the example of a web server application is used, in which the components are web servers and clients, and possibly databases. It may be desirable, however, to have a finer level of componentisation. For instance, user interfaces could be considered components. The arcs in the graph are called *connectors* and they represent the interaction paths between components (again, a broad notion). Georgiadis et al [25] also used a graph of components and connectors, but there they are clearly mapped to self-configuring software components and their communication connections, respectively. Here, the granularity level of the notion of components in the model is not necessarily the same for all architecture descriptions, but is determined be the designer of the architectural model of a specific system.

Many systems allow components and connectors to be annotated with a property list [6] [12] and constraints [46],[48]. These properties are updated during monitoring of the running system and the constraints on them are used to decide when an adaptation is necessary. The autonomic infrastructure is loosely coupled with the running system. In fact, it can run on a different machine, so as not to hinder the running system [6]. In some the code of components is augmented with checkpoints for example to allow reporting of the occurrence of specific method calls, thereby making monitoring more straightforward.

4.2.1. Monitoring

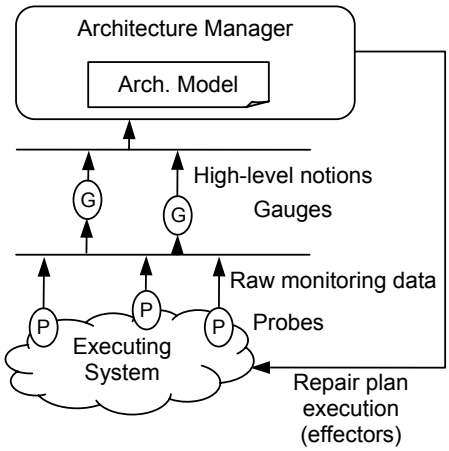

Figure 2: Architecture model-based systems

Figure 2 shows a diagram of an architecture model-based autonomic system illustrating the monitoring infrastructure (it is based on figures from [6],[28], [47] and [48].

Probes can be inserted into the running system to monitor it. These probes are usually localised and deliver system-specific observations. For example, a probe might be deployed to report the size of files that are loaded into a system, in which case the appropriate system call in the OS would be instrumented to allow this type of monitoring by a probe [8]. The raw monitoring data provided by the probes must be aggregated and mapped to high-level notions in the architecture model. This job is performed by so-called *gauges*, which are intermediary components between the probes in the running system and the architecture manager, which controls adaptation of the system at the architecture model level. Gauges may need to collect data from various probes to be able to compute high-level observations. These high-level data allow the architecture model to be updated based on the current state of the executing system. When a property in the architecture model is updated through monitoring, the architecture model is analysed to determine whether the system is still per-

forming adequately. If not, a repair plan is created. The repair plan describes which components or connectors are to be removed or adjusted and which ones are to be inserted. The repair plan is created based on repair strategies that are defined in advance. For many of the architectures the adaptive strategy is closed in the future we may see the knowledge of the success of past repair plans used to determine the best strategy [7]. This can be determined 'of-line' on another machine, however a considerable amount of bandwidth may be required for monitoring, and this can become a problem if the monitoring data travels on the same network interface as application data as seen in Patia[48] and [6].

An advantage of the complete separation between autonomic behaviour and the running system is that software adaptation can be "plugged into" a pre-existing system [28]. The only direct interaction with the target system is through the probes that monitor the system and the effectors that make adjustments and reconfigurations. Changes can be coarse-grained, such as replacing entire components or rearranging the connections among components, but they can also be fine-grained, such as changing the operational parameters, internal state or functioning logic of individual components [29]. Naturally, the architecture model used for adaptation decisions has to be created after the existing system. Valetto and Kaiser [28] tested this concept on a server farm delivering instant messaging (IM) services. monitor the load of IM servers, and if a threshold is exceeded, another IM server is started. This may also happen when an existing server fails. The infrastructure implementing the adaptivity of the system, e.g. the deployment of Worklets, gauges and probes, is called Workflakes. The workflow describing the adaptation process was initially expressed as a set of coding patterns in Java.

Because of the centralised nature of the architecture-based approach to self-healing systems, deployment of such a system in a truly distributed environment, where unplanned failures, such as that of the central architecture manager, can occur requires particular attention to fault tolerance in the infrastructure and the individual components. For example [28], use a decentralised workflow system that uses agents that are part of the adaptivity infrastructure, which is cleanly, separated from the running system.

4.3. Hot swapping components

In this section, research projects are described that focus on how to effect adaptation within the system. This usually involves replacing components in the system with new ones that possess similar functionality but with a different implementation and which are more appropriate to the current condition of the system and its environment.

Much research has been carried out with regard to the hot swapping of components to reconfigure the system [11],[12],[13],[14] and [51]. Typically this involves various stages: terminating a component that is to be replaced and suspending any components and connectors bordering the affected area; removing components and connectors and adding new ones as defined by the repair plan; and resuming components and connectors affected.

Rutherford et al. [33] show how an Enterprise JavaBeans system can be extended such that components can be replaced with new versions. In particular, they must implement the reloading of parameters and refreshing its bindings, two activities that are not part of the standard JavaBean interface. Preliminary experiments show that loading a new component and binding it in the system takes on the order of a few seconds.

Whisnant, Kalbarczyk and Iyer [34] describe a system model in which operational elements can be reconfigured by changing the bindings between operations and the code blocks invoked through the operation. Interestingly, this approach does not require entire components to be terminated, removed and replaced. Modelling at the level of code blocks allows efficient adaptation of component behaviour. Here adaptivity is fine-grained in that it permeates the design of the system down to the code such as at tree data structure.

Appavoo et al. [35] show how the component-based operating system K42 has been improved to support hot swapping of components in the OS and support of applications. This is carried out by the transfer of the component state from the old component to the new. This can be freely chosen by the component, and need not follow a canonical form. New components can be downloaded and plugged into the running system at any time after the deployment of the original system. Their experiments showed that overhead was negligible compared to the performance gains achieved. This was possible because they implemented hot swapping at the OS level. That is, the notion of a component here is fine-grained: a component is for example the File Cache Manager (FCM)

5. Metrics and evaluation

With modern computing, consisting of new paradigms such as planetary-wide computing,

pervasive, and ubiquitous computing, systems are more complex than before. Interestingly, when chip design became more complex we employed computers to design them. Today we are now at the point where humans have limited input to chip design. With systems becoming more complex it is a natural progression to have the system to not only automatically generate code but build systems, and carryout the day-to-day running and configuration of the live system. Therefore autonomic computing has become inevitable and therefore will become more prevalent. Hence their evaluation is increasingly important. This section lists sets of metrics and means by which we can compare such systems.

5.1. Quality of Service (QoS)

QoS is possibly the top-level means to compare modern systems – it should reflect the degree to which the system is reaching its primary goal. It is typically composed of a number of metrics, e.g. data delivery turnaround time over cost. It is a highly important metric in autonomic systems as they are typically designed to improve some aspect of a service. Most of the research in this field is looking at using autonomicity to improve performance (usually speed or efficiency). However other systems wish to improve the user's experience with the system in self-adaptive or personalised GUI design for disabled people etc. Overall this metric is tightly coupled to the application area or service that is expected of the system. It can be measured as a global goal metric or at the sub-service or component level where each unit's ability to met its local goal is measured.

5.2. Cost

Autonomicity costs, the degree of this cost and its measurement is not clear-cut. Currently most performance studies of architecture-based autonomic systems have measured its ability to reach its goal. However agent-based systems typically compare the amount of communication, actions performed, and cost of actions required to reach the goal.

For many commercial systems the aim is to improve the cost of running an infrastructure, which includes primarily people costs in terms of systems administrators and maintenance. This means that the reduction in cost for such systems cannot be measured immediately but over time and as the system becomes more and more self-managing. Further many commercial companies have had difficulties in sharing the vision of autonomic computing with their shareholders for this reason [53]. It should however be noted that current *de facto* commercial performance comparison has borne cost in mind for some time. For example TPC benchmarks have always taken cost per performance into account [54].

Cost comparison is further complicated by the fact that adding autonomicity means adding intelligence, monitors and adaptation mechanisms – and these cost. In Patia we aimed to measure the cost of adding autonomic features to a webserver, which can cope with fluctuating demand and sudden high demand (flash crowds) [48]. We found that the costs of adding monitors and monitor traffic were only just outweighed by the benefits they provided under the normal operation of the server specifically. As this was fairly predictable it was hardly worth-wile. However under duress the system would not work without the autonomic features. Therefore would a comparative characteristic be to do with added functionality achievable that would otherwise not be achieved in a non-autonomic system? As this might be found in a serendipitous fashion, it could be difficult to predict what to test for in advance.

The actual architecture can also impact in the measurement of the cost of a self-adaptive system. For example most architecture-based solutions consist of a service that has autonomic features added. For many of these architectures the intelligence to run the system is separate and centralised, the monitors or gages are external to what they are measuring and the decision to adapt and its supervision is external to the component. Here the question is do we compare systems that use other computing nodes to run the autonomic services with those who run the autonomic services on the same system? With the former, costs could be in terms of extra hardware and communications to that hardware node, and the saving is that it lessens the impact on the running of the main system. Extra nodes dedicated to the autonomic services means that they can be more intelligent, checking the validity of a given reconfiguration or if it is an optimum configuration of many candidates. Further extra nodes can allow resources for open intelligence where the autonomic decisions themselves are fed into the autonomic system for it to self-evaluate and learn.

In AI and agent-based autonomic systems, the intelligence is highly distributed and usually contained within the component or agent. The latter type of system can have the intelligence to carry out its service tightly coupled to the self-management intelligence continued within its component. Therefore the self-management overhead is perhaps indistinguishable from the agent's core function and

therefore it is more difficult to separate out the costs – if sensible at all.

Further, a class of application very fitting to autonomic computing is that of Ubiquitous computing which typically consists of networks of sensors working together to create intelligent homes, monitor the environment etc [47]. This sort of application relies on self-reliance, distributed self-configuration intelligence and monitoring. However many of the nodes in such a system are limited in resources and can be wireless, which means that the cost of autonomous computing involves resource consumption such as battery power.

5.3. Granularity/Flexibility

The granularity of autonomicity is an important issue when comparing autonomic systems. Fine grained components with specific adaptation rules will be highly flexible and perhaps adapt to situations better, however this may cause more overhead in terms of the global system. That is, if we assume that each finer-grained component requires environmental data and is providing some form of feedback on its performance then potentially there is more monitoring data or at least environmental information flowing around the global system. Of course may not be the case in systems where the intelligence is more centralised. Many current commercial autonomic endeavours are at the thicker grained service level.

Granularity is important for eg in [33], where unbinding, loading and rebinding a component took a few seconds. These few seconds are tolerable in a thick-grained component based architecture where the overheads can be hidden in the system's overall operation and potentially change is not that regular. However in finer-grained architectures, such as an Operating System or Ubiquitous computing where change is either more regular or the components smaller, the hot swap time is potentially too much.

One question we may ask is, can systems that provide the same service be compared with each other if the granularity of autonomicity is different? Perhaps at a high level yes.

5.4. Failure avoidance (Robustness)

Typically many autonomic systems are designed to avoid failure at some level. Many are designed to cope with hardware failure such as a node in a cluster system or a component that is no longer responding. Some avoid failure by retrieving a missing component. Either way the predictability of failure is an aspect in comparing such systems. Some systems will be designed for their ability to cope with predicted failure e.g. using a mean time before failure metric of hardware and others to cope with unpredicted environments. To measure this, the nature of the failure and how predictable that failure is, needs to be varied and the systems' ability to cope measured. Ability to cope could be in terms of a Quality of Service metric that pertains to the application domain.

For example in our Kendra[1] audio server, which is a closed self-adaptive system, we would test Kendra's failure avoidance abilities by varying the bandwidth in terms of available bandwidth and how quickly that bandwidth varied. This would test its ability to avoid periods of silence given certain environmental circumstances. That is, in a network, who's bandwidth only varied slightly or in a predictable way, we observed that Kendra would adapt more gracefully than in a bursty network which saw Kendra adapt up and down the codecs sometimes even missing an opportunity to adapt as it did not notice environmental change as it was handling the previous adaptation [44].

5.5. Degree of Autonomy

Related to failure avoidance, we can compare how autonomous a system is. This would relate to AI and agent-based autonomic systems primarily as their autonomic process is usually to provide an autonomous activity. For example the NASA pathfinder must cope with unpredicted problems and learn to overcome them without external help. Decreasing the degree of predictability in the environment and seeing how the system copes could measure this. Lower predictability could even reach it having to cope with things it was not designed to. A degree of proactivity could also compare these features.

5.6. Adaptivity

We separate out the act of adaptation form the monitoring and intelligence that causes the system to adapt. Adaptivity can be something simple as a parameter begin changed in for example self-configuration systems. Here the adaptation does not impact the performance so much as a component-based reconfiguration. In the latter a component may need to be hot-swapped where state is saved, the new component located and then bound into the system. Some systems are designed to continue execution whilst reconfiguring, while others cannot.

[1] Kendra is a self-adaptive audio player that was developed in 1995 and adapted the delivery of the audio codec to best suit the available bandwidth between a client and the audio server. It monitored audio delivery and if bandwidth changed another codec was chosen. The aim was to keep the audio quality as best as possible and avoid periods of silence [43,44,45,50].

Furthermore the location of such components again impacts the performance of the adaptivity process. That is, a component object, which is currently local to the system verses a component (such as a printer driver for example), having to be retrieved over the Internet, will have significantly differing performance. Perhaps more future systems will have the equivalent of a pre-fetch of components that are likely to be of use and are preloaded to speed up the re-configuration process.

5.7. Time to adapt and Reaction Time

Related to cost and sensitivity, these are measurements concerned with the system reconfiguration and adaptation. The time to adapt is the measurement of the time a system takes to adapt to a change in the environment. That is, the time taken between the identification that a change is required until the change has been effected safely and the system moves to a continue state. Reaction time can be seen to partly envelop the adaptation time. This is the time between when an environmental element has changed and the system recognises that change, decides on what reconfiguration is necessary to react to the environmental change and get the system ready to adapt. Further the reaction time affects the sensitivity of the autonomic system to its environment (see below).

5.8. Sensitivity

This is a measurement of how well the self-adaptive system fits with the environment it is sitting in. At one extreme a highly sensitive system will notice a subtle change as it happens and adapt (perhaps subtly) to improve itself based on that change. However in reality, depending on the nature of the activity, there is usually some form of delay in the feedback that some part of the environment has changed effecting a change in the autonomic system. Further the changeover takes time. Therefore if a system is highly sensitive to its environment potentially it can cause the system to be constantly changing configuration etc and not getting on with the job itself.

In measuring Kendra we made the parameters such that the system became more sensitive to the fluctuations in bandwidth to see if it would improve the reaction and ultimately have the delivery of the audio better match the bandwidth available to it. As mentioned in section 5.4, Kendra is a relatively simple self-adaptation system, yet the numbers of parameters, which affected the sensitivity of the adaptation mechanism, were many. For example we could vary the buffer size (which is the data area used to buffer audio), disaster horizon (how close the system thinks it is to a disaster situation), monitoring sample rates (how much environmental data to monitor and store to use predict change in bandwidth). We found that in a generally low bandwidth link it is better that the system is not sensitive as that adaptation process impeded too much on the delivery of the sound. However in good network conditions it is better to be more sensitive as this delivers the best all round quality of sound [48].

5.9. Stabilisation

Another metric related to sensitivity is stabilisation. That is the time taken for the system to learn its environment and stabilise its operation. This is particularly interesting for open adaptive systems that learn how to best reconfigure the system. For closed autonomic systems the sensitivity would be a product of the static rule/constraint base and the stability of the underlying environment the system must adapt to.

5.10. Benchmarking

Finally, it will become necessary to bring these metrics together to form some sort of benchmarking tool. There are two approaches this can take; either we can derive new autonomic systems benchmarks or we can augment current benchmarks to incorporate metrics, which measure autonomic characteristics. Our initial attempt to do with was with the Patia project [Patia]. This project required we test our autonomic webserver and compare its performance with current webservers. We soon found that current webserver benchmarks would not only be able to test the autonomic aspects of Patia, but actually did not measure how traditional webservers were being used. It soon became apparent that we would have to design and build a new webserver benchmark, namely Aeolus [42]. This took research, which describes modern web access and data, characteristics, and built a benchmark based on this. Further, we wished to test the robustness of our Patia webserver under extreme conditions where we simulated a flash crowd that would test the autonomic features of Patia to the extreme. Using many of the metrics we have mentioned in this section, we extended the Aeolus webserver benchmark accordingly. However, we do not believe that deriving benchmarks that measure autonomic systems is the way forward. Instead, due to the diverse application of autonomic systems, it seems better to augment application specific benchmarks to include metrics which evaluate autonomic features of that system e.g. robustness, reaction speed, stability etc. In particular the

Quality of Service benchmark, which we believe is the top-level measurement of how well the system is meeting its goals, is specific to the application in question. Therefore we see traditional benchmarks such as the TPC benchmarks being used to measure autonomic DBMSs but perhaps extended to test the autonomous nature of the system.

6. Conclusions

Autonomic computing is an engineering concept that has found its way in a myriad of computing fields. This paper is a review of some typical examples of autonomic computing attempting to give the reader a feel for the nature of these types of systems and in doing so illustrate the complexities in trying to measure the performance of such systems and compare them. We have presented two major types of architecture that exhibit autonomic properties and describe these as AI (agent-based) and architecture based. We have presented the common components found in each of these types of system, and from this derived a set of metrics and methods which we believe can be used to compare autonomic computing systems. These are:

- Quality of Service
- Cost
- Granularity/Flexibility
- Failure avoidance (Robustness)
- Degree of Autonomy
- Adaptivity
- Time to adapt and Reaction Time
- Sensitivity
- Stabilisation

We realise that some of these metrics are more general than others and some pertain to some autonomic systems and not to others. However we believe that the next step is to take this information and derive a more formal method to compare performance of autonomic systems.

A final note regarding our experience of evaluating the Kendra architecture. Here we set the top-level QoS goal to be that the audio quality was as high as possible while avoiding periods of silence. When testing the system we measured general quality levels, unnecessary adaptation, missed opportunities to adapt, sensitivity to environment etc. Kendra is a relatively simple system with closed self-adaptation (i.e. the autonomic intelligence does not grow), yet the performance statistics were of a large volume and difficult to interpret - especially in terms of relating behaviour to varying the many tuning parameters and differing environment (networking) conditions. We felt that no concrete quantifiable conclusions were really made other than to say that over sensitivity in bursty networks is bad which we possibly would have guessed.

Therefore, finally, it is interesting that alleviate the maintenance and operation of our modern more complex systems we require that addition of even more complexity. It is our argument that this complexity makes such systems much more difficult to evaluate than before and therefore the need to derive metrics and benchmarks is a highly important and interesting area.

References

[1] Kephart J. O., Chess D.M.. *The Vision of Autonomic Computing.* Computer, IEEE, Volume 36, Issue 1, January 2003, Pages 41-50

[2] Bantz D. F. et al. *Autonomic personal computing.* IBM Systems Journal, Vol 42, No 1, 2003

[3] Dailey Paulson L.. *Computer System, Heal Thyself.* Computer, IEEE, Volume 35, Issue 8, August 2002, Pages 20-22

[4] Pescovitz D.. *Helping computers help themselves*. Spectrum, IEEE, Volume 39, Issue 9, September 2002, Pages 49-53

[5] Wooldridge M., *An Introduction to MultiAgent Systems.* John Wiley & Sons Ltd

[6] Garlan D., B. Schmerl. *Exploiting Architectural Design Knowledge to Support Self-Repairing Systems*. Proceedings of the 14th international conference on Software engineering and knowledge engineering. July 2002

[7] Garlan D., B. Schmerl. *Model-based Adaptation for Self-Healing Systems*. Proceedings of the first workshop on Self-healing systems, November 2002

[8] Garlan D., Schmerl B., Chan J.. *Using Gauges for Architecture-Based Monitoring and Adaptation*. Working Conference on Complex and Dynamic Systems Architecture, Brisbane, Australia, December 2001.

[9] Cheng S-W., Garlan D., Schmerl B., Sousa J. P., Spitznagel B., Steenkiste P.. *Using Architectural Syle as a Basis for System Self-repair*. Software Architecture: System Design, Development, and Maintenance (Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture) Bosch J., Gentleman M., Hofmeister C. Kuusela , J. (Eds), Kluwer Academic Publishers, August 25-31, 2002. Pages 45-59

[10] Cheng S-W., Garlan D., Schmerl B., Steenkiste P., Ningning Hu. *Software Architecture-based Adaptation for Grid Computing.* 11th IEEE Conference on High Performance Distributed Computing (HPDC'02), Edinburgh, Scotland, July 2002.

[11] Dashofy E. M., van der Hoek A., Taylor R. N.. *Towards Architecture-based Self-Healing Systems*. Proceedings of the first workshop on Self-healing systems, November 2002

[12] Dashofy E. M., van der Hoek A., Taylor R.N.. *An Infrastructure for the Rapid Development of*

XML-based Architecture Description Languages. Proceedings Of the 24th International Conference on Software Engineering (ICSE2002), Orlando, Florida, May 2002.

[13] xADL 2.0 Homepage. URL: http://www.isr.uci.edu/projects/xarchuci/

[14] ArchStudio 3 Foundations – c2.fw. URL: http://www.isr.uci.edu/projects/archstudio/c2fw.html

[15] de Lemos R., Fiadeiro J. L.. *An Architectural Support for Self-Adaptive Software for Treating Faults*

[16] Kramer J. and Magee J.. *The Evolving Philosopher Problem: Dynamic Change Management*. IEEE Transactions on Software Engineering, Vol. 16, No. 11, November 1990.

[17] BlueGene/L Team at IBM and Lawrence Livermore National Laboratory, IBM Research and IBM Rochester. *An Overview of the BlueGene/L Supercomputer*.

[18] Wolpert D. H., Wheeler K. R., Tumer K., *Collective Intelligence for Control of Distributed Dynamical Systems*. NASA Ames Research Center, Technical Report No NASA-ARC-IC-99-44.

[19] Kuo-Ming C., James A., Norman P.. *A Framework for Intelligent Agents within Effective Concurrent Design*. The Sixth International Conference on Computer Supported Cooperative Work in Design, 12-14 July 2001, Pages 338–343

[20] Gamma E. et al. *Design patterns: elements of reusable object-oriented software*. Addison Wesley.

[21] Object Management Group. URL: http://www.omg.org/

[22] Liu T., Martonosi M.. *Impala: A Middleware System for Managing Autonomic Parallel Sensor Systems*. Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming, June 2003.

[23] TCPA – The Trusted Computing Platform Alliance. URL: http://www.trustedcomputing.org/

[24] Muscettola N., Nayak P. P., Pell B., Williams B. C.. *Remote Agent: To Boldly Go Where No AI System Has Gone Before.* NASA Ames Research Center.

[25] Georgiadis I., Magee J., Kramer J.. *Self-Organising Software Architectures for Distributed Systems*. ACM, Proceedings of the first workshop on Self-healing systems, November 2002.

[26] Sterritt R., Bustard D.. *Towards an Autonomic Computing Environment*. University of Ulster, Northern Ireland.

[27] Kumar S.,. Cohen P. R. *Towards a Fault-Tolerant Multi-Agent System Architecture*.

[28] Valetto G., Kaiser G.. *A Case Study in Software Adaptation*. ACM, Proceedings of the first workshop on Self-healing systems. November 2002.

[29] Valetto G., Kaiser G.. *Combining Mobile Agents and Process-based Coordination to Achieve Software Adaptation.* Columbia University.

[30] Cougaar – Cognitive Agent Architecture. URL: http://www.cougaar.org/

[31] The Globus Heartbeat Monitor Specification v. 1.0. URL: http://www-fp.globus.org/hbm/heartbeat_spec.html

[32] Wyatt J., Hotz, H. Sherwood R., Szijjaro J., Sue M., *Beacon Monitor Operations on the Deep Space One Mission.* 5th Int. Sym. AI, Robotics and Automation in Space, Tokyo, Japan, 1998.

[33] Rutherford M. J., Anderson K., Carzaniga A., Heimbigner D., Wolf A. L., *Reconfiguration in the Enterprise Javabean Component Model.* Proceedings of the IFIP/ACM Working Conference on Component Deployment, Berlin, 2002, Pages 67-81.

[34] Whisnant K., Kalbarczyk Z. T., Iyer R. K., *A system model for dynamically reconfigurable software*. IBM Systems Journal, Vol. 42, No. 1, 2003.

[35] Appavoo J. et al. *Enabling autonomic behaviour in systems software with hot swapping*. IBM Systems Journal, Vol. 42, No. 1, 2003.

[36] Bigus J. P. et al. *ABLE: A toolkit for building multiagent autonomic systems*. IBM Systems Journal, Vol. 41, No. 3, 2002.

[37] Wise A. et al. *Using Little-JIL to Coordinate Agents in Software Engineering*. In Automated Software Engineering Conference (ASE 2000), September 2000.

[38] Markl V., Lohman G. M., Raman V.. *LEO: An autonomic query optimizer for DB2*. IBM Systems Journal, Vol. 42, No. 1, 2003.

[39] *How Sun™ Grid Engine, Enterprise Edition 5.3 Works*. URL: http://wwws.sun.com/software/gridware/sgeee53/wp-sgeee/wp-sgeee.pdf

[40] Kon F., Campbell R. H., Mickunas M. D.,. Nahrstedt, K Ballesteros F. J.. *2K: A Distributed Operating System for Dynamic Heterogeneous Environments*. IEEE, Proceedings of The Ninth International Symposium on High-Performance Distributed Computing, 1-4 Aug. 2000, Pages 201-208.

[41] Blair G. S. et al. *Reflection, Self-Awareness and Self-Healing in OpenORB*. ACM, Proceedings of the first workshop on Self-healing systems, November 2002. Pages 9-14.

[42] Bletsas E. N., McCann, J. A AEOLUS: An Extensible Webserver Benchmarking Tool submitted to 13th IW3C2 and ACM World Wide Web Conference (WWW04), New York City, 17-22 May 2004.

[43] McCann J. A,. Crane J.S.,'Kendra: Internet Distribution & Delivery System an introductory paper', Proc. SCS EuroMedia Conference, Leicester, UK, Ed. Verbraeck A., Al-Akaidi M., Society for Computer Simulation International, January 1998. pp 134-140

[44] McCann J.A., Howlett P., Crane J.S., 'Kendra: Adaptive Internet System', Journal of Systems and Software, Elsevier Science, Volume 55, Issue 1, 5 November 2000 .pp 3-17.

[45] McCann J.A., 'The Kendra Cache Replacement Policy and its Distribution', published in World Wide Web An International Journal, Volume 3, Number 4, Baltzer Science Publishers, ISSN 1386-145X, December 2000.pp231-240.

[46] McCann J .A. , Jawaheer G., Sun L., 'Patia: Adaptive Distributed Webserver (a Position Paper)' International Symposium on Autonomous Decentralized Systems (ISADS). April 2003

[47] McCann J .A., 'ANS (Autonomic Networked System): A Position Paper', 1st UK-UbiNet Workshop,25-26th September 2003

[48] McCann J .A., Jawaheer G. 'Experiences in Building the Patia Autonomic Webserver' 1st International Workshop "Autonomic Computing Systems", DEXA 2003

[49] McCann J .A. 'The Database Machine: Old Story, New Slant?' Proceedings of the first Biennial Conference on Innovative Data Systems Research, VLDB, January 5-8 2003

[50] McCann J .A., 'Adaptivity for Improving Web Streaming Application Performance', chapter in Adaptive Evolutionary Information Systems. ed. N. V. Patel, 2002

[51] Kostkova P., McCann J .A., ' Support for Dynamic Trading and Runtime Adaptability in Mobile Environments' chapter in Adaptive Evolutionary Information Systems. ed. N. V. Patel, 2002.

[52] Barr R., Bicket J. C., Dantas D.S., Du B., T. W. D. Kim, B. Zhou, E. Gün Sirer ‘On the need for system-level support for ad hoc and sensor networks. ACM SIGOPS Operating Systems Review, Volume 36 Issue 2

[53] Nussey I and Telford R presentation part of the IBM Academic Autonomic Computing Day, London, 29th October 2003.

[54] The Transaction and Performance Processing Council, PC http://www.tpc.org/