PACT: Personal Autonomic Computing Tools

Roy Sterritt, Barry Smyth, Martin Bradley
School of Computing and Mathematics,
University of Ulster
Northern Ireland
r.sterritt@ulster.ac.uk

Abstract

Personal Computing offers unique challenges for self-management due to its multi-equipment, multi-situation, and multi-user nature. The aim of this research is to improve the autonomicity within personal computing; the evolution towards Personal Autonomic Computing. In this paper proof of concept self-managing tools are presented and lessons learned derived. At this stage of the autonomic evolution the need of new software development processes to provide the self- properties is becoming very evident.*

Keywords: Autonomic Computing, Personal Autonomic Computing, Self-Monitoring

1. Introduction

Personal Autonomic Computing is Autonomic Computing [1] in a personal computing environment [2]. Personal computing has evolved substantially becoming a consumer product. Its scope now extends from end user computing in the office, to home PCs, wireless laptops, palm tops and next generation mobile phones. In the near future these will be leaf nodes in the self-managing ubiquitous and pervasive computing environments incorporating next generation internet.

Personal computing is an area that can benefit substantially from autonomic principles. Examples of current difficult experiences that can be overcome by such an approach include [2]: (i) trouble connecting to a wired or a wireless network at a conference, hotel or other work location; (ii) switching between home and work; (iii) losing a working connection (and shouting across the office to see if anyone else has had the same problem!); (iv) going into the IP settings area in Windows and being unsure about the correct values to use; (v) having a PC which stops booting and needs major repair or re-installation of the operating system; (vi) recovering from a hard-disk crash; and (vii) migrating efficiently to a new PC. Coping with these situations should be routine and straightforward but in practice such incidents are typically stressful and often waste a considerable amount of productive time.

Autonomic Personal Computing shares the goals of personal computing – responsiveness, ease of use and flexibility – with those of autonomic computing – simplicity, availability and security [3].

Personal computing also creates some problems for the implementation of autonomic principles. In particular [2], personal computing users are often, of necessity, system administrators for the equipment they use. Most are amateurs without formal training, who perform system operations infrequently. This reduces their effectiveness and typically requires them to consult with others to resolve difficulties.

To clarify the basic requirements and activities of a personal autonomic computer-based system environment this paper presents and considers several developed proof of concepts, establishing lessons learnt for further planned developments. It first recaps the personal Autonomic Element establishing the need for self-monitoring, vital health signs and reflex communications and an application of a personal web server. It then concludes with a discussion on lessons learnt.

2. Personal Autonomic Computing Architecture

Achieving high usability and security for personal systems requires rapid responses to changing circumstances. The PAC architecture incorporates a mechanism equivalent to the biological reflex reactions, to alert members of the peer group to situations requiring urgent attention. Human reflex reactions enable a rapid response to pain, such as when a hot object is touched. In computing terms, it is assumed that a system will have to reconfigure itself to avoid a detected threat, while maintaining its operation as far as possible. This may result in the system operating with a reduced set of resources [4]. Like the body, a system can then address the problem causing the reaction with less urgency; this may involve some damage repair.

Proceedings of the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05)
0-7695-2308-0/05 $20.00 © 2005 **IEEE**

Figure 1 shows an abstract view of a system architecture to support this model [5]. This is similar in nature to the architecture proposed in the IBM blueprint where an autonomic manager consists of monitor, analyse, plan and execute (MAPE) components [6].

An autonomic element is made up of a managed component and an autonomic manager. The self-monitor actively observes the state of the component and its external environment, drawing conclusions using information in the system knowledge base. If necessary, this can lead to adjustments to the managed component. One additional feature is the use of a heartbeat monitor (HBM) extended to a pulse monitor (PBM) [7] to summarize the state of the managed component for other connected autonomic elements. Essentially it provides an indication of the health of the managed component or external environment as viewed by that manager, with the absence of a signal (heartbeat) indicating a specific problem with the manger itself. The signal itself, like a pulse, can provide additional information to further explain the state of the element and trigger reflex actions [8].

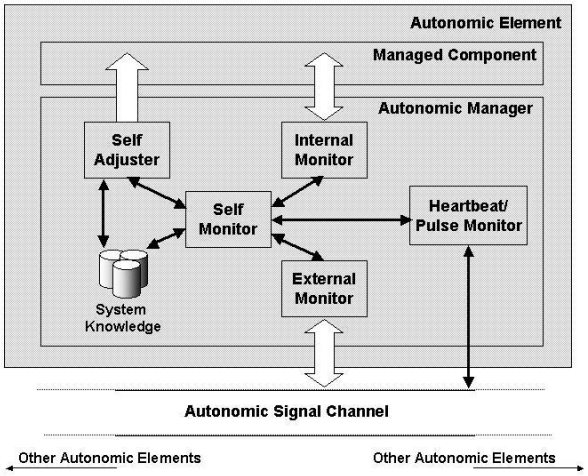

Figure 1 Architecture of an Autonomic Element [5]

NASA has a similar construct, the *Beacon monitor* [9]. A spacecraft sends a signal to the ground that indicates how urgent it is to track it for telemetry data (the beacon states are *nominal*, *interesting*, *important*, *urgent*, and *no tone*). This concept involved a paradigm shift for NASA from routine telemetry downlink and ground analysis to onboard health determination and autonomous data summarisation.

Key within this AE is the ability to provide self-awareness through a combination of a control loop (self-monitor and self-adjuster) and the system knowledge.

3 Self-Monitoring: Systems Monitor Utility Prototype

Figure 1 highlights the importance of the ability to self-monitor. An internal monitor utilizes sensors to detect changing conditions on the managed component – in this case the personal computing system, completing a control loop with the ability to self-adjust through effectors. As has been highlighted, in personal autonomic computing it is particularly important to provide a view into the mechanics for the user to the personal system [10],[11] and allow them to take over the management function if so desired.

3.1 Self-Monitoring for Potential Personal Systems Failures

Today we have many more emerging technologies that rely on integration with standard PC's. For example:

- PDA's – handheld computing devices
- Mobile Phones
- Wireless connectivity through Bluetooth [12] devices and IEEE 802.11 [13] standard devices

These devices work independently but most commonly integrate with PC's or rely on the PC to be the central repository for information from which they may update themselves. Hence if the PC is not a stable and well managed system, the others that rely so heavily on it may also fail or at least may not function at their full level of potential.

There are a variety of ways in which a personal system failure may be detected. Mainly it requires monitoring active process status, Windows services, drivers, processor work rate, memory page file rate, etc. These are the components that the operating systems rely on to function. If one more of these components behave in a certain way, it is possible for a monitor utility to recognize these patterns and possibly perform actions to either prevent system failure, or limit the damage caused by it. For example:

- *Processor work rate* – if the work rate is in excess of 90% for long periods of time, it may cause a crash, this can be monitored and alerts can then ensue.
- *Memory usage (RAM)* – if the system memory is repeatedly being filled and the excess spilled out in to the hard disk (virtual memory) then this in itself may cause problems. This is mainly because of the arrival of real-time and multimedia applications that need work to be performed within allocated time periods.
- *Services/Processes* – these are often interconnected components that rely on one or more related services/processes. When one

fails, unless the software using it is designed to handle the exception it may cause a chain reaction. But, if the failure of the first one can be detected immediately, it may be possible to limit the damage caused, or halt the process of the chain reaction.

- *Thread monitoring* – every process/service can have multiple threads, which are functions being performed simultaneously by that process/service. It is possible to monitor these, but it is not easy to do so dynamically. It is possible to monitor all the threads for a given application (static monitoring) if the designer knew the application to be monitored in advance. The current state of affairs would result in a monitor needing to be designed for each and every application.

In terms of monitoring processes and their threads, as autonomicity evolves, integration and communication concerning autonomicity between components would facilitate self-monitor program updates much the way anti-virus software currently works. This would allow the monitor to self-update by communicating with the new application to understand its tasks, its monitoring needs, its interconnections to the system and adapting to it to ensure complete system coverage. This highlights the need for applications to declare their intended use of resources and management needs.

3.2 Related Work

Currently in Windows there are two obvious means for fault diagnosis provided:

- Windows Event Viewer
- Services.msc

The main issue with both of these systems is that when Windows is installed, there are no direct ways for users to access them, as by default the Administrative tools section is not installed in the Start menu. This must be turned on manually and most personal computing users would not know the services.msc /eventvwr run commands.

3.2.1 Windows Event Viewer

The Windows event viewer is currently found on the Windows start button under the directory: Programs>Administrative Tools> Event Viewer or by entering eventvwr in the run command.

"An event is some action performed by or recognized by the computer that has an effect on the machine as a whole or any application running therein." – as defined by Windows help.

The event viewer typically shows a serially ordered list of actions that your computer takes. The actions are categorised into three sections: Application, Security and System actions. Although this certainly does allow an element of traceability for the actions a PC performs in the background, it normally is difficult for inexperienced home users to understand. These are more aimed at experienced ICT technicians to diagnose faults. For example they show error codes, which can then be looked up on Microsoft Technet [14] for possible solutions.

3.2.2 Services.msc

You can run Services.msc through the run command or by traversing the Start Menu and selecting:

"Programs>Administrative Tools>Services"

A service is an application type that runs in the background and is similar to UNIX daemon applications. Service applications typically provide features such as client / server applications, Web servers, database servers, and other server-based applications to users, both locally and across the network.

You can use Services.msc to:

- Start, stop, pause, resume, or disable services on remote and local computers. You must have the appropriate permissions to start, stop, pause, restart, and disable services.
- Manage services on local and remote computers
- Set-up recovery actions to take place if a service fails, for example, restarting the service automatically or restarting the computer
- Enable or disable services for a particular hardware profile.
- View the status and description of each service.

Typically the services application shows a list of the services that are available through Windows. It shows their status (i.e. started, starting, stopping, stopped) and their start-up type (manual, automatic – when windows starts, or disabled.)

This is a slightly less technical than the event viewer as a lot of the services are recognizable to mainstream users.

3.2.3 Other systems available

There are other systems that are commercially available to provide fault tolerance but most are designed to be used in conjunction with large networks, utilizing environments configurations to achieve redundancy, used in business environments and hence are too complex and too expensive for mainstream personal users. For instance: GRID, HIVE, Windows 2000/2003 Clustering and P2P (Peer-to-peer) networks.

3.3 Window's System Self-Monitor Utility

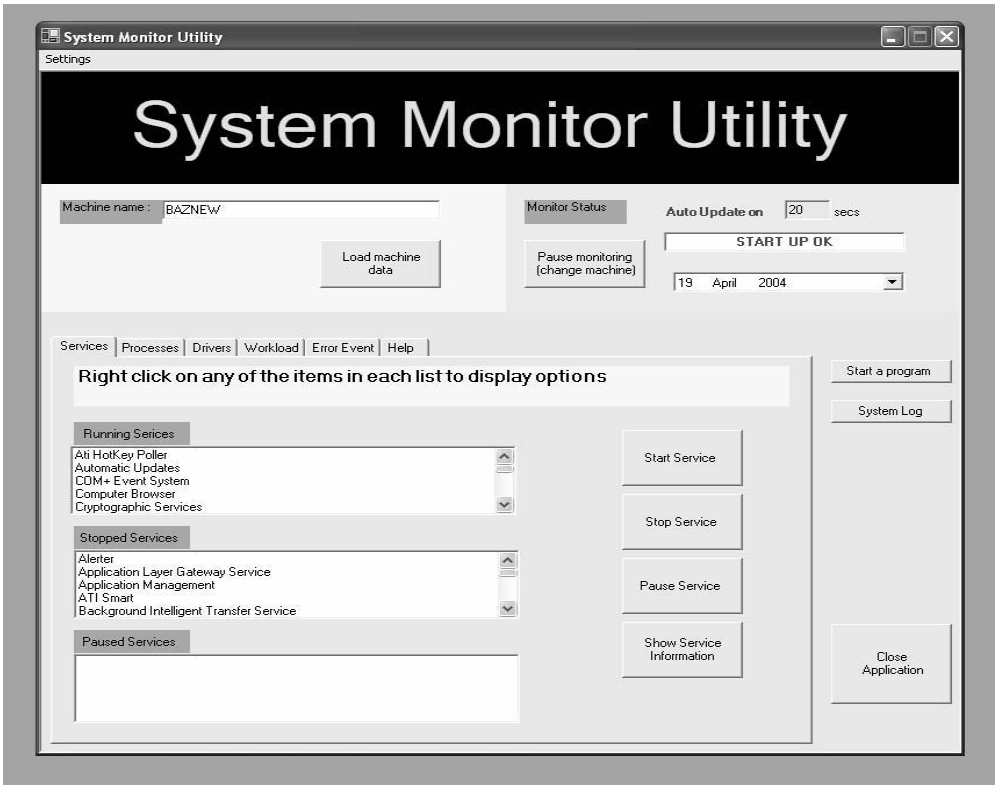

Figure 2 Windows System Self-Monitor Interface

The Windows System Self-Monitor prototype brings all the monitoring function found in the various obscure admin tools within windows (admin tools, control panel, Microsoft Management Console (MMC) and so on) into the one tool, making the details available to the autonomic manager while also providing a means to see in behind the autonomics for the typical personal computing user.

Some key elements in Figure 2 to note are:

- Machine name – can be local/remote
- Tab menu – allows one interface to show variety of categories of information so user does not get lost in screen layouts.
- Monitor status – tells the user if the auto update element is active/inactive, the update interval, start-up mode, system date.
- Error event – tells the user what to do in the event of a system error if they wish to do so manually and the autonomic manager is not activated.

These aspects of the operating system; services, processes, drivers and workload may then all be monitored by the autonomic manager and be used to reflect its view of the health of the managed component (which is broadcast through the pulse) as well as take self-* management activities to rectify any developing situations.

4 Self-Healing: Vital Signs & Pulse Monitor prototype

A personal autonomic computing self-healing tool has been designed as an initial proof of concept [15]. The assumption behind the tool is that dying/hanging processes on a PC are signs or indicators of the health of that PC. These *vital signs* may indicate that the PC is becoming unstable and possibly in immanent danger of hanging or unreliable for current processes running on that machine. Peers are notified of this situation via a change in *pulse*.

This is particularly useful in situations where the PC is unattended for example a web server, and the user may be notified via a peer PC that the machine is in difficulty [16].

The underlying functionality of the tool is a heart-beat monitor; if a process hangs it should be restarted and the pulse monitor takes note. Upon several processes hanging or the same process repeatedly hanging within specified timeframes, a change occurs in the monitor's perception of how healthy the machine is and as such brings about a change in the pulse being broadcast from that PC.

Since the tool operates in a P2P mode it also takes responsibility to watch out for its neighbours; as such other PCs (peers) will register with it and it will monitor their pulse.

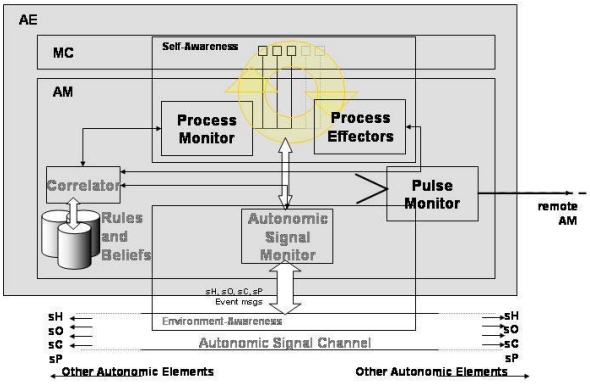

Figure 3 – AE implementation for Self-Healing-Vital Signs and Pulse Monitoring tool

Figure 3 shows the overview of the implementation of the autonomic element specifically for utilizing the processes as the vital heal signs along with the pulse monitor construct to communication the urgency levels- the pulse- (Table 1) to neighbouring autonomic elements.

Table 1 – Pulse value

Level	Description	Pulse Change Trigger (adaptable)
0	Nominal	no failed process
1	Interesting	1 failed non-essential process
2	Important	1 failed essential process *or* 2 failed non-essential processes
3	Urgent	2+ failed essential processes or 3+ failed non-essential processes
—	No Pulse	Pulse monitor, or comms has failed

The process monitor inside a host takes care of monitoring its health condition which is represented by a Pulse. The processor effectors attempts to restart a process when it hangs (as well as cleaning up the results of the hung process). Each host is able to send its Pulse to a peer via an external monitor. The 'rules and beliefs' stores the pulse level and rules (i.e. predefined knowledge) which may adapt over time; the monitoring logs; and the history of neighbour hosts. A computer system is different from a biological system; human biology reflection is involuntary while the decision making in computer system is based on a set of predefined rules or policies. For example, rules such as the Pulse sending interval and terminate the failed process after three trials of re-starting the process, are re-configurable.

The amount of processes required to cause a change in pulse is adaptable and need not necessarily remain at the values depicted in Table 1, as is the time window for qualifying failing processes.

Figure 5 depicts a scenario where several process hung and cause the changes in the pulse from nominal to interesting to urgent and back to nominal as the self-healing tool successfully cleans-up and restarts the processes.

5. Autonomic Configuration of Personal Web Servers

The internet has become the ubiquitous communication medium of today's computer-based systems. As with other personal computing scenarios running your own (personal) web servers will more than likely not enjoy the redundancy and fault-tolerant infrastructure of large scale organizations. The previous section highlighted that the pulse within the self-healing tool may be used to alert if an unattended web server is experience difficulties. The next prototype is specifically concerned with self-configuring and self-optimizing the web-server running on your PC.

Though web servers only form one component of the WWW infrastructure, their study is motivated by the fact that server delays are becoming an increasingly dominant factor in user perceived Web Performance. There have been claims that 40% of Web delays are in fact due to web servers [17].

5.1 Self-Configuring and Self-Optimizing

The problem that was examined is that of tuning the Apache webserver over time. It is problematic in that to do it manually is time-consuming, error prone and requires highly skilled personnel making it costly [18]. Therefore, it is pragmatic and cost effective to use a tool to automate this. The Apache webserver is an important element of many businesses. It is one of the most popular webservers in use today [18] so it can be assumed that a wide customer base is available for an autonomic tool that will maximize it.

The reason that the Apache webserver requires to be tuned is that it will not always be used under the same

conditions. Some users of the server may consider it important that as many users as possible can be connected to the site at any one time. Other users may put a higher priority on the speed at which a request is dealt with. It is also possible that the server could be running on two different specification machines and that one of these machines does not have the ability to deal with as many requests as the other. Business needs change (as does demand (hits) to the web site). Under these varying circumstances it is unlikely that any one set of configuration values would be optimal for all users.

Tuning the server is continuous task. If a set of optimal values are found these may not always be suitable. If the content of the website changes or if extra memory is added to the machine the tuning process will have to be repeated until the new optimal values are found. If a tool was available for this task it would therefore save the system administrator time allowing them to complete other tasks.

5.2 Related Work

The ABLE toolkit [19] has been successfully used to create Autotune Agents that can control webserver performance [18]. The approach used was to apply control theory to calculate the configuration parameters for the server. However the process of applying control theory was automated. A modeling agent was used for this purpose. The modeling agent needed to be used in non-production mode. The data collected by the modeling agent was then passed to a controller design agent which created the algorithm that would be used to control the server. The algorithm designed is implemented by a runtime control agent. This monitors the system metrics and modifies the parameters in an effort to achieve some desired level of usage.

There are a number of significant elements to the system that were not relevant to the problem here. Firstly the Apache webserver was modified to allow dynamic control of the parameters MaxClients and KeepAlive. This has a number of advantages; firstly it allows the configuration to be updated in response to a varying workload. This is beneficial because it ensures that a certain level of service can be maintained. Secondly it will ensure that as many clients as possible will be connected at any one time. It does this by ensuring that extra clients can connect when non-dynamic pages are being accessed (dynamic pages require extra CPU usage so to obtain the same level of service less clients can connect when dynamic pages are being viewed). The system was designed for a UNIX platform and if the solution to the problem is on a different platform, namely Windows, then the parameter MaxClients may not be available.

The approach taken with the Patia Autonomic Webserver [17] is to create a webserver that is autonomic rather than to try and retrospectively add autonomicity to an existing one. It involves dispersing the data over a number of Patia webservers and analyzing the client connection and then deciding on what way to deal with the request. One metric that is analyzed is the bandwidth of the connection and depending on the bandwidth the most appropriate service agent is chosen to respond to the request. If for example the bandwidth was low a different codec for streaming media may be chosen than for a high bandwidth connection.

The approach taken by the Patia Autonomic Webserver is not a solution to the problem being addressed here. It does however raise a number of important points, firstly configuring for flash crowds.

One problem that the Patia webserver had was the gathering of metrics heavily used resources so much so that it had a detrimental effect on the server. It will be important to ensure that any work trying to optimize the server does not affect the work carried out by the server.

5.3 Self-Tuning Apache Web Server Tool

The prototype was designed to self-configure the Apache web server, similar to the research in [18] that used ABLE [19] without the luxury of a version of Apache that could be reconfigured without being restarted.

The tool developed in Java and C (to access platform dependant features) concentrates on monitoring the CPU usage and memory usage.

Another important element to the research of this problem was to consider how the performance of the Apache webserver could be optimized. The Apache server is configured using a file called “httpd.conf”. This file contains the parameters that the server will use to run with. This file is read in and used when the Apache server is started and in order for any changes to be implemented it would require the server to be restarted.

Within this file there are a number of parameters that can be used to control performance. On a Windows Platform these are “ThreadsPerChild” and “KeepAliveTimeOut” and in a UNIX platform they are “MaxClients” and “KeepAliveTimeOut”.

The Apache server creates one master process which creates a child process that deals with requests. On Windows this child process creates threads to deal with requests. The Maximum number of threads that can be in the system at any one time is defined by the parameter “ThreadsPerChild”. The UNIX platform is similar except that it does not create threads instead it uses “worker processes”. The Maximum number of worker process that can be in the system at any one time is set

using the "MaxClients" parameter. This controls the maximum number of client connections that can be handled at any one time. The "KeepAliveTimeOut" is used for the same purpose on both platforms. It is used to determined how long to wait for another request from a persistent connection. These parameters can be used to configure the system to deliver the required performance. A screenshot of the tool is depicted in Figure 4, which allows the user to set policies concerning updating, range of values and logging as well as providing that view inside the autonomics.

6. Discussion

There is a great need to establish standards and mechanisms for autonomic computing to succeed. For instance as was mentioned it is possible to develop a self-healing tool with a control loop that constantly monitors the processes running on your laptop [15]. If any should hang, the autonomic tool can restart that process. For instance the scenario in Figure 5 where scrolling through a pdf file within a web browser crashed. Yet there is no means to inform the process where to restart – effectively it's a process being started from fresh with any previous state lost unless the process' application itself handles this. In the example in Figure 5 the tool successfully cleared up the multiple hung processes and restarted them – from a personal computing users perspective it was obvious Netscape and acrobat reader plug-in had hung, these disappeared and restarted – but not back to the web page where one started. There is a need for standards for autonomic signals and communications to take place not only at this level – autonomic manager to processes running on the managed component but also autonomic manager to autonomic manager. Allowing standard 'autonomic signal' routes into processes would raise security issues – yet this will need to be part of the self-protection autonomic property.

Since this implies all processes effectively need to be designed with autonomicity and self-managing capabilities in mind – not only from within but taking direction from the external environment – this not only raises issues of standards to achieve this but raises questions do the current design and development approaches meet the needs for developing autonomicity and handle human error due to complexity. The realisation of self-managing systems, which will still be complex to design, may only move the human error aspect from the administrator (who had been manually managing the running systems) to the designer.

Personal computing adds consideration for the user to the equation. In the example present in Figure 5 it was obvious the applications had hung, yet other scenarios have occurred when the user had not realised an application had hung (it was not being used at the moment) but its killing off and restarting by the self-healing tool was found to be disconcerting to the user.

This concern for the personal computing user requires that the autonomicity not exclude the user and potentially lead to a lack of trust. As such it is critical to also provide user interfaces into the autonomics.

7. Conclusion

Overall, autonomic computing is intended to improve the general usability (hide complexity) and manageability of computing systems. As such, in principle, it will benefit all computer users in due course. Since for the majority of users access to computing is through personal devices, autonomic research in this area should have a significant impact. In the longer term, the work is of direct relevance to emerging computing paradigms such as utility/grid, ubiquitous, pervasive, invisible, and so on, which require systems to self-manage to fulfil their potential.

The research in this paper investigates personal autonomic computing through a series of prototypes incorporating self-monitoring, vital signs and pulse monitoring, self-healing, self-configuring and self-optimizing.

The key lessons learnt from this personal autonomic computing research is the need for standards to communicate autonomic intentions between processes/applications and the autonomic elements; the needs for new methods to design and code autonomicity; and keeping the human in the loop – interfaces are still needed to allow visibility into what the self-managing systems are doing.

Acknowledgements

The development of this paper was supported by the University of Ulster's Computer Science Research Institute and the Centre for Software Process Technologies (CSPT) funded by Invest NI through the Centres of Excellence Programme under the EU Peace II initiative.

References

[1] Horn P, "Autonomic computing: IBM perspective on the state of information technology", IBM October 2001

[2] Bantz DF, Bisdikian C, Challener D, Karidis JP, Mastrianni S, Mohindra A, Shea DG, Vanover M, "Autonomic personal computing", IBM Systems Journal, 42(1), 2003, pp. 165-176

[3] Bantz DF, Frank D, "Challenges in Autonomic Personal Computing, with Some New Results in Automatic Configuration Management", Proc. IEEE INDIN; AUCOPA workshop, Banff, Alberta, Canada, 22-23 August 2003

[4] Bapty T, Neema S, Nordstorm S, Shetty S, Vashishtha D, Overdorf J, Sheldon P, "Modeling and Generation Tools for Large-Scale, Real-Time Embedded Systems", Proc 10th IEEE ECBS, Huntsville, Alabama, USA, 7-10th April 2003, pp11-16.
[5] Sterritt R, Bustard DW, "Towards an Autonomic Computing Environment", Proc. IEEE DEXA 2003 Workshops - 1st Int. Workshop on Autonomic Computing Systems, Prague, Czech Republic, September 1-5, 2003, Pages 694-698.
[6] IBM, "An architectural blueprint for autonomic computing", April 2003.
[7] Sterritt R, "Pulse Monitoring: Extending the Health-check for the Autonomic GRID", Proc. IEEE Workshop on Autonomic Computing Principles and Architectures (AUCOPA 2003) at INDIN 2003, Banff, Alberta, Canada, 22-23 August 2003
[8] Sterritt R, "Towards Autonomic Computing: Effective Event Management", Proc. 27th Annual IEEE/NASA Software Engineering Workshop, Greenbelt, MD, Dec. 2002,. pp 40-47.
[9] 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
[10] Russell DM, Maglio PP, Dordick R, Neti C, "Dealing with ghosts: Managing the user experience of autonomic computing", IBM Systems Journal, 42(1), 2003, pp. 177-188
[11] Conference on the Human Impact and Application of Autonomic Computing Systems (CHIACS2).Yorktown Heights .NY: IBM TJ Watson Research Center, 21 April 2004.
[12] Bluetooth, http://www.bluetooth.com/
[13] IEE 802.11, http://grouper.ieee.org/groups/802/11/main.html
[14] Technet : A Microsoft website used for fault diagnosis in Windows operating systems and applications. http://www.microsoft.com/technet/default.mspx
[15] Sterritt R, Chung S, "Personal Autonomic Computing Self-Healing Tool" Proceedings of 11th IEEE ECBS - Engineering of Autonomic Systems Workshop (EASe'04), Brno, Czech Republic, May 24-27, 2004
[16] Sterritt R, Bantz DF, (Aug 2004) "PAC-MEN: Personal Autonomic Computing Monitoring Environments", Proceedings of IEEE DEXA 2004 Workshops - 2nd International Workshop on Self-Adaptive and Autonomic Computing Systems (SAACS 04), Zaragoza, Spain, August 30th - September 3rd, IEEE, Pages 737-741
[17] McCann J, Jawaheer G, "The Patia Autonomic Webserver: Feasibility Experimentation", Proc. IEEE DEXA Workshops - ACS, Prague, Czech Rep., Sept. 1-5, 2003, pp 661-667
[18] Y.Diao, J. L. Hellerstein, S. Parekh, J. P. Bigus, Managing Web server performance with AutoTune agents, IBM Systems Journal, Vol 42 2003
[19] Bigus JP et.al., "ABLE: a toolkit for building multiagent autonomic systems," *IBM Systems J.,* 41(3), pp.350-371, 2002

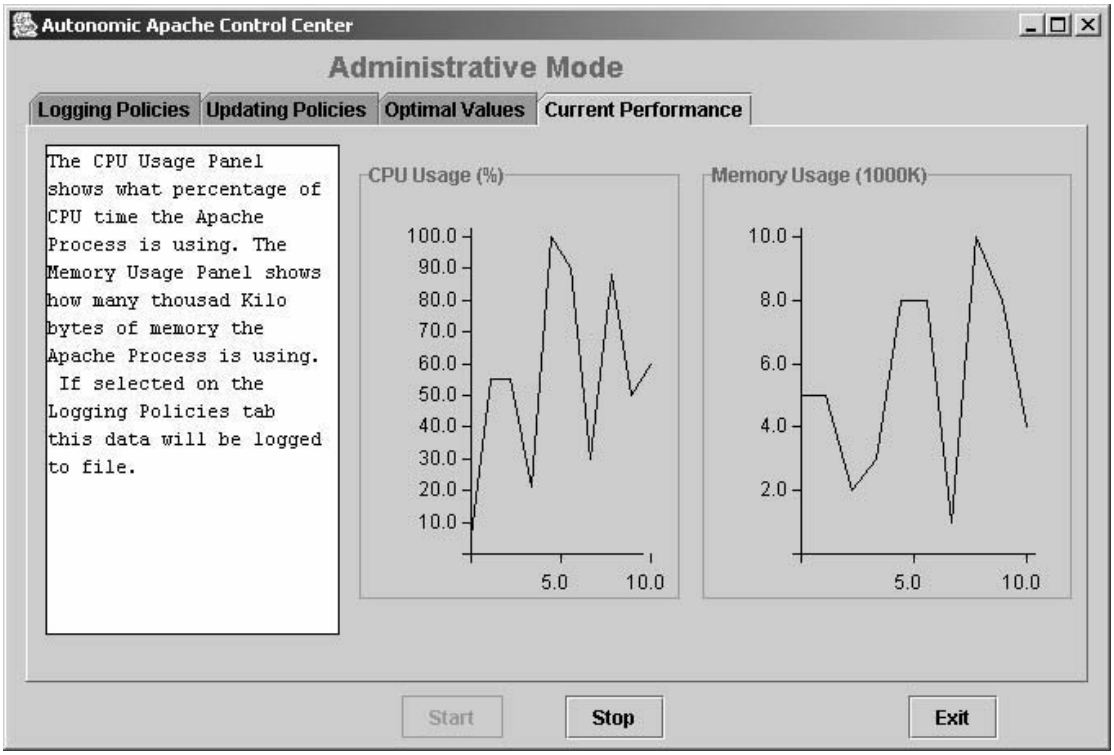

Figure 4 Self-Tuning (self-configuring and self-optimizing) Apache

Figure 5 A developing scenario within the Self-Healing Tool