Self-healing and Self-protecting Computing Systems: In the Search of Autonomic Computing

Luis M. Fernández-Carrasco, Hugo Terashima-Marín, Manuel Valenzuela-Rendón

Center for Intelligent Systems
Instituto Tecnológico y de Estudios Superiores de Monterrey
Monterrey, México
{A00789695, terashima, valenzuela}@itesm.mx

Abstract. This document presents a work-in-progress agent-based architecture design to tackle two important autonomic computing features: self-healing and self-protecting. Research in the autonomic computing systems area has seen a great growth in recent years as more complex systems have been developed and more communication among those is needed. However, there is still the lack of an architecture that facilitates the emergence of autonomic computing features, and more specifically, self-protection andself-healing. Consequently, this document aim satfilling this need and proposes an agent-based approach in order to achieve these autonomic computing properties. In order to evaluate the design, a simulation is used. Based on this simulation, it is believed that the proposed approach is a good first step to wards an autonomi ccomputing architecture.

1 Introduction

Richard Murch [1] defines autonomic computing as the ability to manage one's computing enterprise systems through hardware and software so that they automatically and dynamically respond to the requirements of the changing environment. In other words, this means achieving *self-healing*, *self-configuring*, *self-optimizing*, and *self-protecting* hardware and software that behave according to defined policies [1]. These systems are thought after the autonomic nervous system which responds to the needs of the body, so autonomic computing systems should respond to the needs of the environment.

Autonomic computing was born as a consequence of the great advances in networking, computing technologies, and software tools. These sophisticated applications and services are complex, heterogeneous, and dynamic [2]. Moreover, the underlying information infrastructure (e.g., the Internet) globally aggregates large numbers of independent computing and communication resources, data stores, and sensor networks, and is itself complex [2]. This combined scale of complexity, heterogeneity, and dynamism of networks, systems, and applications have made computational and information infrastructures be brittle, unmanageable, and insecure. This situation has made researchers look for a new paradigm

© G. Sidorov (Ed.)
Advances in Artificial Intelligence: Algorithms and Applications
Research in Computing Science 40, 2008, pp. 109-118

for systems and application design. This new vision has been referred to as autonomic computing and is based on strategies used by biological systems when dealing with similar challenges.

Nevertheless, meeting the grand challenges of autonomic computing requires scientific and technological advances in many fields and from many technologies [2]. The goal of this document, therefore, is to provide a new mechanism to achieve autonomic computing systems. This paper proposes an architecture, following a multi-agent approach, capable to provide and support self-healing and self-protection properties. For evaluation purposes, a simulated environment emulating a computer was built.

The following paragraphs present a thorough description of the related work, pointing out differences and similarities. Then, the main idea behind the proposed model is explained so that, when the next section introduces the architecture model, no doubts arise. Right after the model is presented, the simulation used to test the architecture is explained. After, the preliminary results of this work-in-progress research are presented, followed by the the conclusions that can be drawn from the experiments and by some final words about the proposed approach towards autonomic computing.

2 Related Work

A multi-agent approach to autonomic computing is not something new. Jennings [3] has already mentioned the advantages of decomposing problems in terms of decentralized, autonomous agents. Agents add flexibility and high-level of interactions to the design of a system. Thus, it is no surprise to use autonomous agents when trying to build autonomic systems as todays large-scale computing systems get highly distributed with increasingly complex connectivity and interactions.

As a result, there are works that make use of autonomous agents to achieve autonomic systems. Tesauro et al [4] developed *Unity*, a decentralized architecture to enable autonomic computing based on multiple interacting agents called *autonomic elements*. However, Unity covers only some of the wanted self-$\star$ desired properties, namely, goal-driven self-assembly and real-time self-optimization. The proposed architecture in this document aims at supporting not only the ones achieved by Tesauro et al [4], but most self-$\star$ properties an autonomic system should show, making emphasis on *self-optimizing*, and *self-protecting*.

Bonino et al [5] proposed an agent-based autonomic semantic platform. They makes use of the autonomic computing vision and propose *DOSE* [5] to automatically index new resources in response to search failures and auto-detection of low covered conceptual areas when performing tasks on a semantic platform. Although, this work makes use of an agent-based autonomic platform, it is different from the work proposed in this article as it attempts to go beyond using autonomic computing features on a specific area and, on the contrary, aims at being the core that enables autonomic computing features.

Other related works in the autonomic computing field are *OceanStore* [6], which is a global, consistent, highly available persistent data storage system that supports self-healing, self-optimization, self-configuration, self-protection. Also, *Storage Tank* [6] is a multi-platform, universally accessible storage management system. It supports self-optimization, self-healing. *Oceano* [6] facilitates cost effective scalable management of computing resources for software farms. Nevertheless, all the last mentioned works do not follow an agent-based approach. This article's objective is to provide the first results of a work-in-progress software architecture that is capable of not only enabling autonomic behavior.

3 Main Idea for the Architecture

The main idea is to treat everything as an autonomous agent, even the computer. This follows the idea to use autonomic elements suggested by Kephart and Chess [7]. They point out that autonomic elements should function at many levels, from individual computing components such as disk drives to small-scale computing systems such as workstations. Moreover, many ideas developed in the multi-agent systems community, such as automatic group formation, emergent behavior, multi-agent adaptation, and agent coordination could likely be adapted for autonomic computing.

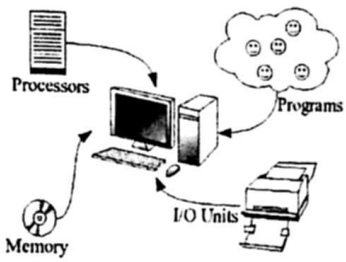

Fig. 1. Agentifying all components.

Consequently, the computer processor, the memory, and even the programs that are run by the computer are modeled as agents, in other words, all components are ***agentified***[1]. These agents communicate with the computer by sending messages as seen in Figure 1. All computing elements have to register with the computer so that this last one *knows* what it has and what it can do with them. The messages that are sent to the computer contain information such as properties, tasks that the component can do, etc.

[1] Agentify is an expression used to turn software and hardware components into agent entities.

4 Proposed Autonomic System Model

This section details the autonomic system model and its components, the relations among them and how they communicate with each other. Graphically, this model is shown in Figure 2. At first glance, one can see that the *Computer* is in charge of managing all other components or modules the autonomic system has available. Therefore, the other components, represented as agents, only know what they are capable of doing, their requirements to work and the results they generate.

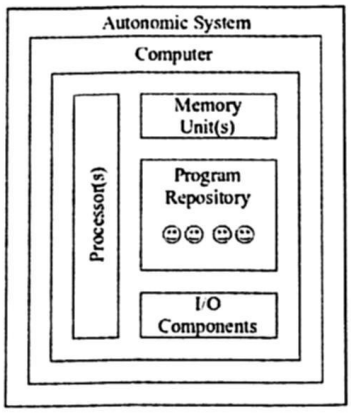

Fig. 2. Autonomic system agent architecture.

Figure 3 presents the communication flow proposed among the principal agents as one program is executed by the autonomic system. Such figure presents all the links among agents and who asks for what at program execution time. The agents that are in Figure 3 are explained in detail lines below.

4.1 Computer Agent

This is the one in charge of organizing all the tasks in the autonomic system. Every new component has to register with the computer as it is *plugged in*. These components send a message, called *Discovery Message*, stating what they can do, what *methods* they have embedded and what output they provide (i.e., a manifest). Once a new component registers with the computer agent, this last one maps into memory the address of the module, its embedded methods and the capabilities that the modules has. Only the computer agent knows where these are and these sections of memory cannot be erased unless the module is *unplugged*.

At start-up, this agent also checks if it has registered all available components to work (i.e., a memory and a processor unit). Once it *knows* it has the available

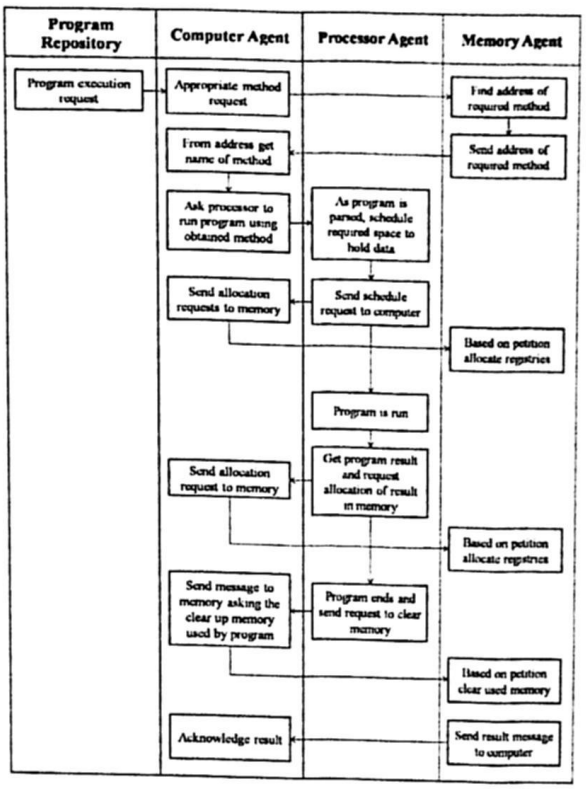

Fig. 3. Communication flow among agents.

resources to respond to petitions, it is able to perform the tasks that other modules send. For instance, if there is the need to run a program, it checks for the appropriate method to conduct that task within the registered processors the computer has. The computer does this by looking at the restricted memory section it has access to. If a suitable method is found, then, it asks the processor to run the program with the method the computer finds suitable. The processor, in turn, asks the computer for memory space. As it was said, only the computer knows what other modules there are in the system and what they are capable of doing. As soon enough memory is allocated for the program to be executed, the processor runs the program and sends back the result to the computer which, then, asks the memory agent to register such result.

4.2 Memory Unit Agents

These agents represent all kinds of storage media. Some examples are hard drives, USB memories, external disks, etc. They can read and write from and to a memory addresses. They can also allocate space for a single value or for an array.

Finally, it is able to erase its contents. However, all the mentioned methods are performed on a request basis, in other words, when the computer agent sends a message asking the memory to perform one action.

4.3 Processor Agents

Agents belonging to this category are the ones in charge to run and perform the tasks the programs demand. In other words, it interprets the programs. It also sends requests for memory access and manipulation to the computer agent. Consequently, it requests to schedule registry readings and writings, or address manipulation. Any processor that is added to the autonomic unit has to register with the computer agent by sending it its manifest. This manifest tells the computer that a new processor has been added and that new capabilities are in the system.

4.4 Program Repository Agent

This agent is the one in charge to spawn programs. A set of programs is already placed in the repository and they are instantiated using a probabilistic method. Let us explain this further, computer users have a set of programs already installed on their PCs; however, there is a number of programs that are used most frequently. Hence, those programs which are used more frequently are instantiated more often. This probability-based instantiation approach is conducted using the genetic algorithm roulette wheel selection method [8]. In other words, this agent acts as the user who is constantly requesting the execution of programs in real-life systems.

4.5 I/O Unit Agents

These agents represent all those peripherals that help display and/or print results. They receive the petition from the computer and display, through whatever available means they have, some information to the user. Similar to the other agents, their methods are also mapped into memory by the computer agent using their manifest.

5 Simulation Building

One observation that the current document may get is the fact that it is proposed as a simulation instead of implementing the proposed architecture in a real computing system. However, a lot of work has been devoted in order to ensure that the simulation is as close as possible to a real system.

There are two components to consider here: the platform that enables agent-based implementation, and a way to generate programs that are significant in the sense that they require memory allocation, etc. Moreover, this programs

should at least be Turing-complete, meaning that it can compute every Turing-computable function [9].

The next lines of this section present both components. First, a description of the platform used to implement the agents and, second, the programs that were created to simulate real computing processes.

5.1 The MadKit Platform

MadKit is a multi-agent platform for developing and running application based on an organizational oriented paradigm. This multi-agent paradigms uses agents, groups and roles as the basic standpoint for building complex applications. MadKit does not enforce any consideration about the internal structure of agents, thus allowing a developer to freely implements its own agent architectures [10].

MadKit is also a distributed platform which allows for the development of efficient distributed applications. For the programmers, all considerations about basic distributed components such as "sockets" and "ports", are totally transparent [10]. An application developed in a multi-agent way can be run in a distributed way without changing a line of code.

MadKit is built around the concept of "micro-kernel" and agentification of services. The MadKit kernel is rather small, but agents offer the important services one needs for his own application. Distribution and remote message passing, monitoring and observation of agents, edition, etc. are all performed by agents [10].

5.2 HAL Programming Language

It was decided to create a new programming language that could allow the building of testing programs and, in that way simulate a computing system. This language was called HAL and it is Turing-complete [9]. It has most of the functions any high level programming language should have.

Anyway, a program coded is not sufficient as it needs a compiler, something that can transform a simple text with commands into something that is *executable*. Moreover, as the simulation is built in Java, it should provide a Java-based code that can be interpreted by the JVM. The next lines present both, the tool used to compile the HAL programs and the HAL programs themselves.

The Java Compiler Compiler. The Java Compiler Compiler (or JavaCC) is a parser generator and a lexical analyzer generator. Compilers and interpreters incorporate lexical analysers and parsers to decipher files containing programs, however lexical analysers and parsers can be used in a wide variety of other applications [11]. Lexical analysers can break a sequence of characters into subsequences called tokens and they also classify the tokens [11].

Usually, in compilers, the parser outputs a tree representing the structure of the program. This tree then serves as an input to components of the compiler responsible for analysis and code generation [11]. The lexical analyser and parser

are responsible for generating error messages, if the input does not conform to the lexical or syntactic rules of the language [11].

A part of the specification file used to create the HAL programs is shown lines below where mainly a few tokens are specified.

```
TOKEN :
{
  < ELSE: "else" >
| < FOR: "for" >
}
HALProgram Parse():
{Function main;
HALProgram result;}
```

HAL Program Characteristics. The HAL programming language comes with the required tools to program almost any kind of applications. It also offers the necessary *functions* for the simulation of the autonomic system. It is not object oriented but a functional program. It is capable to handle *String* and *Char* data types by mapping them as integers to memory. It is capable to declare arrays by typing `matrix = <5>;` or `->matrix = (10 11 12 13 14);`. In the first case, it allocates in memory an array of size 5 that is called `matrix`; in the second one, it already provides the elements to be assigned to the `matrix` variable. As it can be seen, the second example has at the beginning two special characters that are explained lines below.

Regarding the control statements, HAL provides *for*, *while*, and *if* commands. It also provides some already built-in methods for addition *sum*, subtraction *sub*, multiplication *mul*, division *div*, relational operators, etc. These methods have to be preceded by the at @ symbol so that JavaCC regards them as special tokens.

When handling values and variables, there are special characters that provide different data management. These special sequences of characters are:

Assign value to a memory address (− >). This sequence is used when one wants to assign a value to a specific memory address. It should always be on the left of an assignation (=). For instance, the line: `->30 = 4;` means that the value 4 should be stored in the memory address 30.

Get value from memory address (∗). This character gets the value that a specific memory address holds. For instance `*30` asks for the value stored in the memory address 30. If the previous example is considered, the result of using this character is 4.

Get a parameter from function (#). This one is used to get the parameters of a function. For instance if the function is `@sub(20 15);`, which asks to subtract 15 from 20, and then the assignation `paramOne = #1;` is called, the value that `paramOne` holds is 15. This operator is zero-based meaning that the first element is considered to be in the *zero* position.

One could say that these special characters work quite similar to the pointers C or C++ has.

As it can be seen, HAL offers all the possibilities to build sound programs. This characteristic is needed for the present simulation as one wants to test how the processor and memory agents behave as programs are executed following an agent-based

approach. For the purposes of the simulation many HAL programs have already been coded and are kept by the repository agent. It is wanted that these programs represent those that a user makes use of in real systems.

6 Testing the Model

In order to test the proposed model, a number of programs were generated and these were assigned a certain probability. These programs represent those a user make use of. Among these programs, some that attack the memory agent by erasing their restricted registries were coded. However, these agents have a very little probability of being generated. These can be considered as threats to the system. Then, the computer agent is started and it checks if there are sufficient resources to work. As previously mentioned, at least a processor and a memory agent should be present. Once these agents are acknowledged, the computer agent can star processing the programs that are being spawned by the repository agent. One also has the ability to add the components manually, kind of plugging in components and, as this is done, the system maps such new modules to memory.

The main idea behind this architecture is to enable self-⋆ properties so that autonomic computing is achieved. The next lines describe how such properties are enabled by the software architecture proposed here.

6.1 Self-⋆ Properties

Self-healing is provided here whenever an error is found. For the purposes of the simulation, there is a set of HAL programs that were created to disrupt the restricted memory sections were the registered modules have their methods and addresses mapped. These HAL programs change and erase the addresses and registries which in turn leads to an error. The memory agent monitors and ensures that no changes take place in the restricted area and, if such event does happen, it informs the computer of such occurrence. The computer agent then asks for the modules to send their manifests again and repairs the sections that the program might have damaged. The computer agent also, once the notification of an *attack* has been received, stops all programs from being spawned so that no more errors take place, and it only allows the communication of requests once the error has been fixed.

Finally, *self-protection* is achieved by the proposed architecture by not allowing the execution of known programs that *attacked* the system. Lines above, it was mentioned that some HAL programs were capable of damaging restricted memory, once such programs have been identified, the computer agent blacklists them and does not allow the instantiation of any more of those programs. This helps protect the system from known attacks.

In general, being this an agent-based approach, where all elements have been agentified, prevention and anticipation of events, optimization at any level of the system, and configuration based on the capabilities of each component are possible. The software architecture here proposed allows these features.

7 Conclusions & Final Remarks

Tackling autonomic computing by using an agent-based approach is a good path to follow. Agents have the ability to provide easy-to-implement self-⋆ properties as they rep-

resent a unit with their own methods, abilities and responsibilities. This document proposes an agent-based system architecture that enables the emergence of self-protection and self-healing properties, and makes use of a simulation to test the design.

This document tested the architecture using a simulation that allowed agent implementation (i.e., a virtual machine over the JVM). Moreover, in order to make it as real as possible, HAL programming language was developed using JavaCC [11]. Programs built using HAL are Turing-complete, resembling those used by people everyday. The presented software architecture agentifies all its components and establishes that all communications should go through one agent, called *computer agent*. This allows a better control of the system and hides the methods that other modules may have reducing possible attacks to the elements.

Worth pointing out is that this document presents a work-in-progress report and that there are still some things to be worked on.

Acknowledgments. This research project is funded by Tecnológico de Monterrey, Research Chair CAT010. The first author also thanks Jesús Héctor Domínguez Sánchez for providing insightful ideas and recommendations when designing this simulation.

References

1. Murch, R.: Autonomic Computing. On Demand Series. IBM Press (2004)
2. Parashar, M., Hariri, S.: Autonomic Computing: Concepts, Infrastructure, and Applications. CRC Press (2007)
3. Jennings, N.R.: On agent-based software engineering. Artificial Intelligence **117** (April 2000) 277 – 296
4. Tesauro, G., Chess, D.M., Walsh, W.E., Das, R., Segal, A., Whalley, I., Kephart, J.O., White, S.R.: A multi-agent systems approach to autonomic computing. In: AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, IEEE Computer Society (2004) 464 – 471
5. Bonino, D., Bosca, A., Corno, F.: An agent based autonomic semantic platform. In: Proceedings of the International Conference on Autonomic Computing, IEEE Computer Society (2004) 189 – 196
6. Sterritta, R., Parasharb, M., Tianfield, H., Unland, R.: A concise introduction to autonomic computing. Advanced Engineering Informatics **19**(3) (July 2005) 181 – 187
7. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**(1) (2003)
8. Blickle, T., Thiele, L.: A comparison of selection schemes used in genetic algorithms. Technical Report 11, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH) Zurich (1995)
9. Herken, R.: The Universal Turing Machine: A Half-Century Survey. Springer (1995)
10. MadKit: The MadKit project. `http://www.madkit.org/` (2005) Web Page.
11. Norvell, T.S.: The JavaCC Tutorial. `http://www.engr.mun.ca/~theo/JavaCC-Tutorial/javacc-tutorial.pdf` (2002)