Autonomic Systems Modeling and Development: A Survey

IRINA DANIELA CROITORU

A MAJOR REPORT

IN

THE DEPARTMENT

OF COMPUTER SCIENCE

AND

SOFTWARE ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE

CONCORDIA UNIVERSITY

MONTREAL, QUEBEC, CANADA

APRIL 2006

@IRINA DANIELA CROITORU, 2006

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-494-14344-4
Our file Notre référence
ISBN: 0-494-14344-4

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

Autonomic Systems Modeling and Development: A Survey

Irina Daniela Croitoru

This major report is a survey of autonomic systems modeling and development. Its aim is to explore current research and developments, such that subsequently, autonomic computing to be applied to the Concordia University developed TROM formalism and TROMLAB framework. Firstly, the report formulates the vision of Autonomic System Timed Reactive Model (AS-TRM) and briefly introduces real-time reactive systems, TROM formalism and TROMLAB framework. Secondly, it surveys autonomic computing characteristics, abstracting out algorithms with potential for development. Thirdly, it illustrates patterns for modeling and development of autonomic complex systems. In addition, the report conducts a thorough survey on existing intelligent multi-agents technologies and open standards with a high potential of enabling autonomic computing. Insights into industry and academic efforts that leverage autonomic computing are provided as well. At the end, the report provides exploratory research directions that have a high potential for realizing the Autonomic System Timed Reactive Model (AS-TRM).

Acknowledgements and Dedication

I consider myself fortunate to have had Dr. Olga Ormandjieva as a professor and as a supervisor as she is such an intelligent and exquisite person. I thank her deeply for her guidance and for encouraging me to explore autonomic computing.

It was a pleasure to work on this major report, and I dedicate my efforts to my wonderful family, for the joy they bring each day in my life.

Table of Contents

List of Tables

List of Figures

Chapter 1 Introduction

1.1 The Vision of Autonomic System TRM (AS-TRM)

Real-time reactive systems are some of the most complex systems being built today. Examples of such systems are alarm systems, command and control systems, flight control systems, avionics systems, communication systems, robotic systems, process control systems and telecommunication systems. Consequently, the modeling and development of real-time reactive systems is a very challenging and difficult task. The complexity involved comes from their real-time and reactive core characteristics: involve concurrency, have strict timing requirements, must be reliable and involve software and hardware components. In addition, recent real-time reactive systems have become increasingly heterogeneous and increasingly intelligent [5].

Researchers today, have come to the conclusion that one of the ways of removing the complexity barrier is to model and develop complex computer systems that are **autonomic**.

Autonomic computing is a new research area led by IBM Corporation concentrating on making complex computing systems smarter and most importantly easier to manage. Many of its concepts are based on self-controlling model of human autonomic system. As a result, autonomic complex computer systems are envisioned to combine the following seven characteristics: **self-configuring, self-healing, self-optimizing self-protecting,**

self-aware, contextually aware and **open** [3]. The first four characteristics listed above are considered to be the core characteristics of an autonomic computer system.

Autonomic computing paradigm is driven by the increasing complexity of computer systems and the growing induced cost to manage them. By putting this new paradigm in perspective, researchers at Concordia University envision to apply it to the Time Reactive Object Model (TROM) - a rigorous formalism for modeling and development of complex real-time reactive systems [1].

The vision of Autonomic System Timed Reactive Model (AS-TRM) is best described as follows:

"To be able to create autonomic distributed real-time reactive systems on a framework that leverages their modeling, development, integration and maintenance."

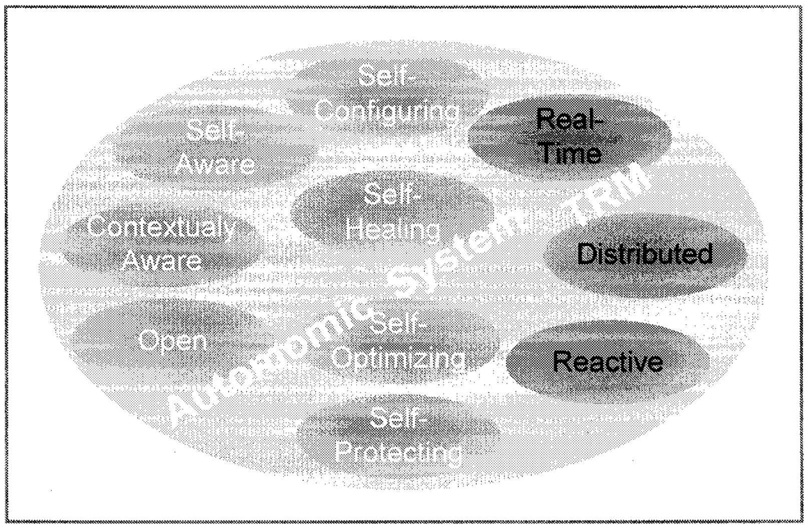


Figure 1: The characteristics of Autonomic System TRM

It is this vision that prompted the current survey whose objectives are to provide a broad overview of the IBM Corporation led research into modeling and development of autonomic systems, and to identify future research directions that have a high potential for realizing the Autonomic System Timed Reactive Model (AS-TRM).

1.2 Autonomic Computing Modeling and Development Challenge

The task of autonomic computing modeling and development was acknowledged to be very difficult and in particular to require significant exploration of **new technologies** and **innovations,** in 2001 by Paul Horn, Senior Vice President of IBM Corporation. In his manifesto at that time, he introduced the grand challenge for the entire information technology industry and academy:

"The information technology industry loves to prove the impossible possible. We obliterate barriers and set records with astonishing regularity. But now we face a problem spinning from the very core of our success - and too few of us are focussed on solving it. More than any other Information Technology problem, this one - if it remains unsolved - will actually prevent us from moving to the next era of computing. The obstacle is ***complexity****.... Dealing with it is the single most important challenge facing the Information Technology industry."* [3]

Furthermore, Paul Horn introduced IBM Corporation Vision of Autonomic Computing as follows:

"Systems manage themselves according to an administrator's goals. New components integrate as effortlessly as a new cell establishes itself in the human body. These ideas are not science fiction, but elements of the grand challenge to ***create self–managing computing systems***" [2, 3, 4].

1.3 An Overview of this Survey

Concordia University AS-TRM group is currently conducting research to extend the formal TROM framework for developing reactive systems, currently based on Timed Reactive Object Model (TROM) formalism, to Autonomic System Timed Reactive Model (AS-TRM) formalism. The purpose of this survey is to provide an insight into TROM framework and more importantly to examine autonomic systems modeling and development in general such that, in a subsequent study, a feasibility assessment for developing an AS-TRM can be made.

Chapter 2 contains details about the on-going Concordia University research on AS-TRM development environment and its distributed, real-time and reactive characteristics.

Chapter 3 discusses in detail the four core characteristics of autonomic systems, namely, self-configuring, self-healing, self-optimizing and self-protecting.

Autonomic systems modeling and development based on IBM Corporation lead research is surveyed in chapter 4.

Existing and emerging tools and open standards that leverage the development of autonomic systems are presented in chapter 5. The accent is put on the multi-agent technology, a definite player in driving autonomic characteristics of complex computing systems.

In chapter 6 case studies on use of autonomic systems in industry and academia are presented. In particular IBM, Sun, HP, Microsoft approaches and their autonomic products will be discussed in detail. Current universities' projects are introduced as well.

Chapter 7 outlines the future research directions for modeling and development of autonomic computing systems.

References

[1] R. Achuthan, *"A Formal Model for Object-Oriented Development of Real-Time Reactive Systems*", Ph.D. Thesis, Department of Computer Science, Concordia University, Montreal, Canada, 1995.

[2] Jeffrey O. Kephart, David M. Chess, "*The Vision of Autonomic Computing,*" Published by IEEE Computer Society, Volume 36 (1), pp. 41-50, 2003, available at: http://www.research.ibm.com/autonomic/research/papers/AC_Vision_Computer_Jan_2003.pdf

[3] Paul Horn, "*Autonomic Computing: IBM's Perspective on the State of Information Technology*", IBM Corporation, October 15, 2001, available at: http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf

[4] A. G. Ganeck, T.A. Corbi, *"The Dawning of the autonomic computing era"*, IBM Systems Journal, Volume 42, No. 1, 2003, available at: http://www.research.ibm.com/journal/sj/421/ganek.pdf

[5] Frantz J. Ramming, *"Autonomic Distributed Real-Time Systems: Challenges and Solutions"*, Proceedings of the Seventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'04), 2004.

Chapter 2 Timed Reactive Object Model (TROM) Formalism and TROMLAB Framework

Abstract

This chapter gives a brief introduction into real-time reactive systems, Timed Reactive Object Model (TROM), and TROMLAB framework developed at Concordia University.

2.1 Introduction

Firstly, we introduce the basic definitions of real-time systems, reactive systems, formal methods, TROM formalism and TROMLAB framework in order to set clearly the context of this chapter. The following subsections, describe in detail the TROM formalism and TROMLAB architecture.

Reactive Systems

Reactive systems react continuously to the environment, at a speed driven by the environment. The three main characteristics of reactive systems are [1]:

- *have infinite behavior*,
- *satisfy stimulus synchronization*: the system reacts always to stimulus from the environment, *and*
- *satisfy response synchronization*: the time elapsed between a stimulus and its response is satisfactory to the environment; that is, the environment is still receptive to the response when the latter is received from the system.

Real-Time Systems (RTS)

Real-Time Systems (RTS) are complex computer systems that have reactive behavior. Basically, an RTS consists of a controller part and a controlled part. The controlled part can be one or more physical devices with fundamental timing requirements. The correctness of an RTS depends both on the time in which computations are performed as well as the logical correctness of the results. The most two important requirements for real-time systems are to provide safe and reliable execution. Severe consequences may result if those two requirements are not met. In general, real-time systems are special purpose systems, require fault tolerance, and usually are embedded in larger complex systems.

Formal Methods

Formal methods are traditional techniques used to verify complex systems. A formal specification can be used as input to a model checker, to prove that the properties of the system are correct and to check for particular types of errors.

Timed Reactive Object Model (TROM)

The TROM formalism provides a formal basis for specification, analysis and refinements of real-time reactive systems. It is mainly based on object-oriented and real-time technologies [1].

TROMLAB

TROMLAB is a software framework developed at Concordia University for rigorous development of real-time reactive systems. Basically, it is an environment "where TROM formalism, language and method, can be practiced in accordance with a process model that integrates formal methods with several phases of the development life cycle. The process model incorporates iterative development, incremental design, validation, and formal verification of design models" [1]. In other words, TROMLAB integrates the formalism that provides solid formal foundation for specification and rigorous analysis with a practical object-oriented methodology to actually develop real-time reactive systems.

2.2 Timed Reactive Object Model (TROM)

The TROM formalism has three levels. The first two levels match the typical object oriented approach to system development that is: define classes, then compose subsystems by instantiating defined classes. The third level is distinctive to TROM and contains the definition of data models.

"The three tiers independently specify systems configuration, reactive objects, and abstract data types, by importing low layer specifications into upper layers. Large and complex systems can be developed incrementally by composing, verifying, and integrating subsystems" [1, 2, 10].

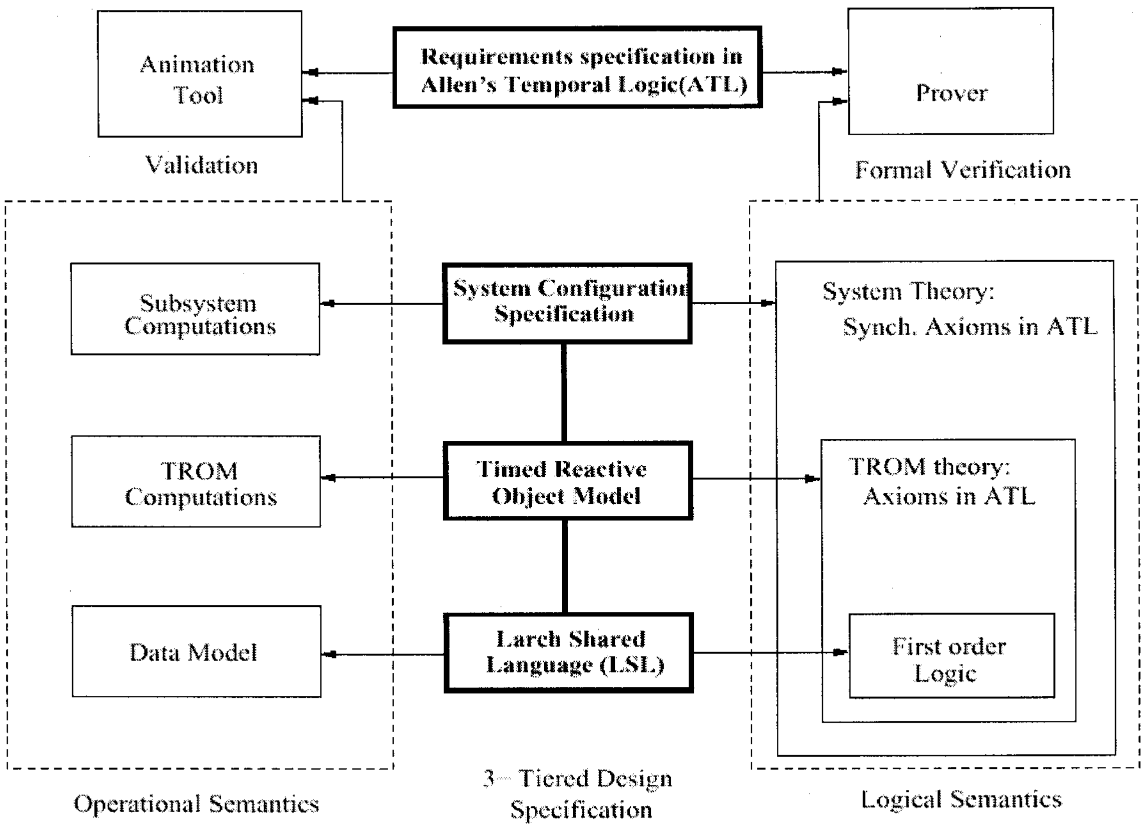

Animation Tool
Requirements specification in Allen's Temporal Logic(ATL)
Prover
Validation
Formal Verification
Subsystem Computations
System Configuration Specification
System Theory: Synch. Axioms in ATL
TROM Computations
Timed Reactive Object Model
TROM theory: Axioms in ATL
Data Model
Larch Shared Language (LSL)
First order Logic
Operational Semantics
3 – Tiered Design Specification
Logical Semantics

Figure 2: TROM model: Three Tier Formalism [1]

2.2.1 First Tier – Larch Formalism

The Larch formalism is a family of languages. It leverages two-tiered style of specification. Each specification has components written in the following two languages:

(1) Larch Interface Language (LIL), and

(2) Larch Shared Language (LSL).

2.2.2 Second Tier – TROM

The second tier is the actual TROM, which at this level is viewed as a "hierarchical finite state machine augmented with ports, attributes, logical assertions on the attributes and time constraints" [1, 2]. A TROM has the following two characteristics: (1) has a single thread of control and (2) communicates with its environment by message passing happening at a port.

2.2.3 Third Layer – System Configuration Specification

The third layer called Configuration Specification (SCS) specifies a real-time reactive system or a subsystem by composing collaborative reactive objects from the second tier or by composing smaller subsystems.

2.3 TROMLAB Architecture

The following figure gives an overview of TROMLAB distributed architecture.

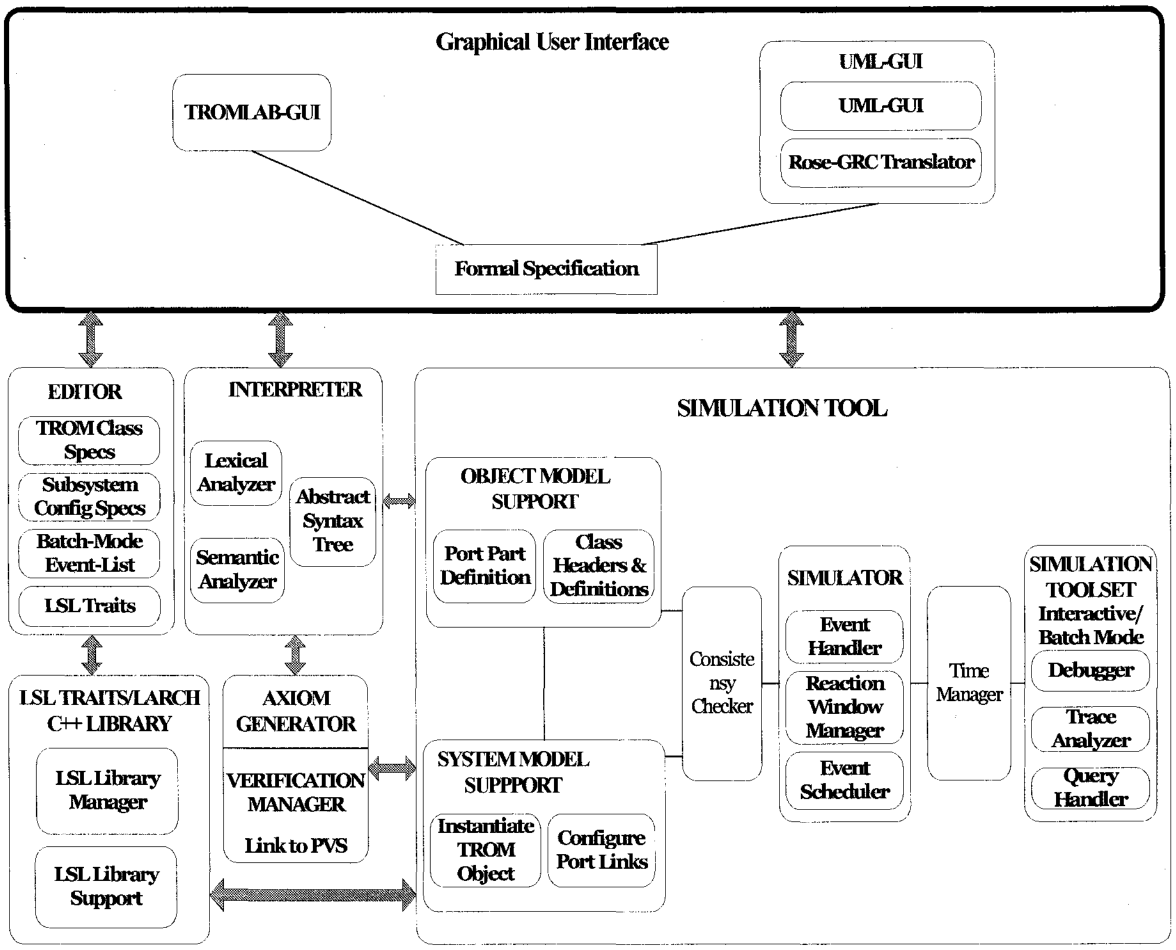


Figure 3: TROMLAB distributed architecture [2]

The following paragraphs contain brief descriptions of current TROMLAB development environment components.

Interpreter

The interpreter is a tool that parses, syntactically checks a specification and subsequently constructs an internal representation [4].

Simulator

The simulator is a tool that simulates a subsystem behavior at the design phase before the implementation, and enables a systematic validation of the specified system [3, 14]; the results are animated with the visualization animation tool [13].

Rose-GRC Translator

Rose-GRC Translator is an automatic tool that allows reactive classes to be visually composed, edited, refined, and automatically mapped to the TROM notation [12].

Browser for Reuse

The browser for reuse is a user interface to the library of system components [7]. It helps users query and access system components for reuse during system development.

Graphical User Interface

The graphical user interface is a visual modeling and interaction tool [5].

Reasoning System

The reasoning system is a tool for system testing during simulation by allowing interactive queries of hypothetical nature for system behavior [6].

Verification Assistant

The verification assistant is an automated tool that enables mechanized axiom extraction from real-time reactive systems [15].

Verification Tool

The verification tool is an automated tool that enables mechanized validation for the safety and liveness properties, and is based on PVS [8].

Test Cases Generator

The Test Generator is an automated tool for generating test cases from TROM specifications and for optimizing the test suite based on the test adequacy measurement [16].

TROM - Software Reliability Measurement System (TROM-SRMS)

TROM-SRMS is a reliability measurement module that predicts the level or reliability from the TROM specifications of the system [17].

TROM - Software Complexity Measurement System (TROM-SCMS)

TROM-SCMS calculates from the TROM specifications the architectural complexity and displays the maintenance profile of the system [9].

2.4 Managing Complexity in Real-Time Reactive Systems

In the year 2000, V. S. Alagar, O. Ormandjieva and M. Zheng were stating in their article called "*Managing complexity in Real-Time Reactive Systems*" [11] the following:

"*There is not much work done in the area of complexity measurement and management for real-time reactive systems.*" [11]

At the time, researchers at Concordia University were approaching methods [18] to assess architectural, design, representation, implementation, testing and validation complexities on TROMLAB framework. Subsequently, there was no surprise that, when IBM announced its lead in autonomic computing, AS-TRM group formulated in turn their vision for Autonomic System Timed Reactive Model. This present survey is the starting point of the journey towards achieving an ambitious goal since autonomic computing has many dimensions to conquer and practically spawns every aspect of computer science.

References

[1] R. Achuthan, *"A Formal Model for Object-Oriented Development of Real-Time Reactive Systems*", Ph.D. Thesis, Department of Computer Science, Concordia University, Montreal, Canada, 1995.

[2] V.S. Alagar, R. Achuthan, D. Muthiayen, "*TROMLAB: A Software Development Environment for Real-Time Reactive Systems*", Technical Report, Concordia University, Montreal, Canada, first version 1996, revised 1999.

[3] D. Muthiayen, *"Animation and Formal Verification of Real-Time Reactive Systems in an Object-Oriented Environment"*, Master of Computer Science Thesis, Computer Science Department, Concordia University, Montreal, Canada, 1996.

[4] A. Tao, "*Static Analyzer: A Design Tool for TROM*", Master of Computer Science Thesis, Department of Computer Science, Concordia University, Montreal, Canada, 1996.

[5] V. Srinivasan, "*An Intelligent Graphical User Interface System for TROMLAB*", Master of Computer Science Thesis, Computer Science Department, Concordia University, Montreal, Canada, 1999.

[6] G. Haidar, "*Simulated Reasoning and Debugging of TROMLAB Environment*", Master of Computer Science Thesis, Department of Computer science, Concordia University, Montreal, Canada, 1999.

[7] R. Nagarajan, "*VISTA – A Visual Interface for Software Reuse in TROMLAB Environment*", Master of Computer Science Thesis, Computer Science Department, Concordia University, Montreal, Canada, 1999.

[8] D. Muthiayen, "*Real-Time Reactive Systems Development – A Formal Approach based on UML and PVS*", Ph.D. Thesis, Computer Science Department, Concordia University, Montreal, Canada, 2000.

[9] M. Zhuo, *"Real-Time Reactive Systems Measurement Tool TROM-QM: Design and Implementation"*, Master of Computer Science Major Report, Concordia University, Computer Science Department, Montreal, Canada, 2003.

[10] Vasu Alagar and Ralf Lammel, *"Three-Tiered Specification of Micro-Architectures*", Proceedings of the 4th International Conference on Formal Engineering Methods: Formal Methods and Software Engineering, pp. 92-97, 2002.

[11] V.S. Alagar, O. Ormandjieva, M. Zheng, *"Managing Complexity in Real-Time Reactive Systems"*, In Proceedings of the Sixth IEEE International Conference on

Engineering of Complex Computer Systems (ICECCS2000), pp. 12-24, Tokyo, Japan, September 2000.

[12] Oanna Popistas, *"Rose-GRC Translator: Mapping UML Visual Models onto Formal Specifications"*, Master of Computer Science Thesis, Computer Science Department, Concordia University, Montreal, Canada, 1999.

[13] Mubarak Sami Mohammad, *"Visualization Animation for Real-Time Reactive Systems Simulation"*, Master of Computer Science Thesis, Computer Science Department, Concordia University, Montreal, Canada, 2004.

[14] Shi Hui Liu, *"Simulated Validation of Real-Time Reactive Systems with Parameterized Events"*, Master of Computer Science Thesis, Computer Science Department, Concordia University, Montreal, Canada, 2003.

[15] F. Pompeo, "*A Formal Verification Assistant for TROMLAB Environment*", Master of Computer Science Thesis, Computer Science Department, Concordia University, Montreal, Canada, 1999.

[16] M. Chen, "*The Implementation of Specification-Based Testing System for Real-time Reactive System in TROMLAB Framework*", Master of Computer Science Major Report, Computer Science Department, Concordia University, Montreal, Canada, 2002.

[17] Fong-Ann Lee, "*Reliability Measurement Based on the Markov Model for Real-time Reactive Systems: Design and Implementation*", Master of Computer Science Major Report, Concordia University, Computer Science Department, Montreal, Canada, 2003.

[18] Olga Ormandjieva, "*Deriving new measurements for real-time reactive systems*", Ph.D. Thesis, Computer Science Department, Concordia University, Montreal, Canada, 2000.

Chapter 3 Autonomic Systems Characteristics

Abstract

This chapter presents a survey of autonomic systems characteristics. Firstly, it reviews some basic terminology of autonomic systems. A quick overview of the human autonomic nervous system is introduced in the following section in order to provide an insight to its autonomic functionality. In the third part, autonomic computing systems characteristics, the counterpart to human nervous autonomic characteristics, are presented in detail. Based on articles we surveyed to date, an introductory hint of current approaches, algorithms and limitations is conducted for each autonomic characteristic in particular. At the last part, the conclusion for this chapter is given.

3.1 Introduction

Here we introduce the definitions for autonomic term in general, autonomic nervous system, pervasive computing paradigm and autonomic computing paradigm.

Definition of Autonomic

The American Heritage Dictionary of the English Language: Fourth Edition, 2000 defines formally autonomic as:

Physiology

a. *Of, relating to, or connected by the autonomic nervous system.*

b. *Occurring involuntarily; automatic: and autonomic reflex.*

Resulting from internal stimuli; spontaneous. [1]

Definition of Human Autonomic Nervous System (ANS)

The term autonomic comes from *auto*, meaning self and *nomos*, meaning law. True to its name, the human autonomic nervous systems runs by itself, involuntary, governed by its own rules as opposed to the somatic nervous system nervous systems in which activity is under voluntary and conscious control. It is responsible for the management of functions of internal organs "adapting them to the needs of the moment and maintaining equilibrium of internal environment of the human body" [2].

Definition of Pervasive Computing Systems

According to Mark Weiser, pervasive computing systems are *"so embedded, so fitting, so natural, that we use them without thinking about them"* [38].

Definition of Autonomic Computing Systems

Autonomic Computing concept was first expressed by Paul Horn at the AGENDA 2001 Conference, as "*an approach to self-managed computing systems with a minimum of human interference*" [9]. In addition, he noted that autonomic computing is very much inspired from the human autonomic nervous system, which monitors and regulates temperature, respiration, heart rate, pupil dilatation, and digestion with little or no human intervention. Furthermore in his definition, Paul Horn linked the idea of pervasiveness to the human autonomic nervous system and consequently to the definition of computing autonomic systems.

Consequently, a complete definition of autonomic computing systems is *"computer systems that achieve the same level of self-regulation and pervasiveness as human autonomic system"* [9].

3.2 The Human Autonomic Nervous System

Historically, the discovery of the human autonomic nervous system is attributed to Galen (AD 130-200). A great anatomist and physiologist of ancient times, he was the first to describe some of the cranial nerves, the sympathetic chains, the cervical ganglia, and the rami communicants [2].

Anatomically, the human nervous system is composed of nerve cells or neurons that are arranged in circuits and networks, the simple being the reflex arc. In a simple reflex arc such as is a knee jerk, a stimulus is detected by a receptor cell, which interacts with a sensory neuron. This latter in turn sends the impulse to the spinal cord, where it interacts with an inter-neuron. The motor neuron, which interacts with the inter-neuron, carries the nerve impulse to the effector, such as a muscle, and this in turn responds by contracting.

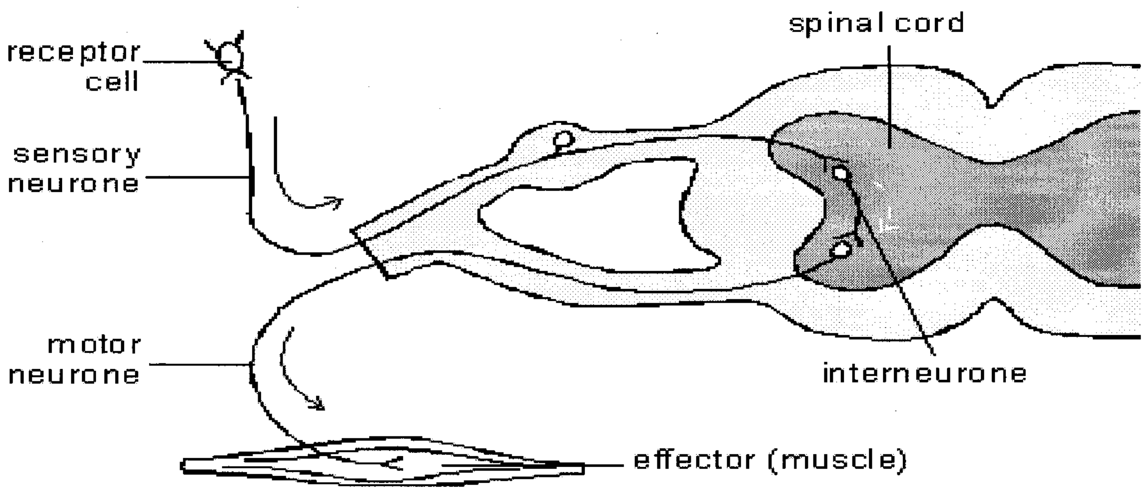


Figure 4: Illustration of simple arc reflex [2]

A simple flow diagram representation of the simple arc reflex depicted above is as follows:

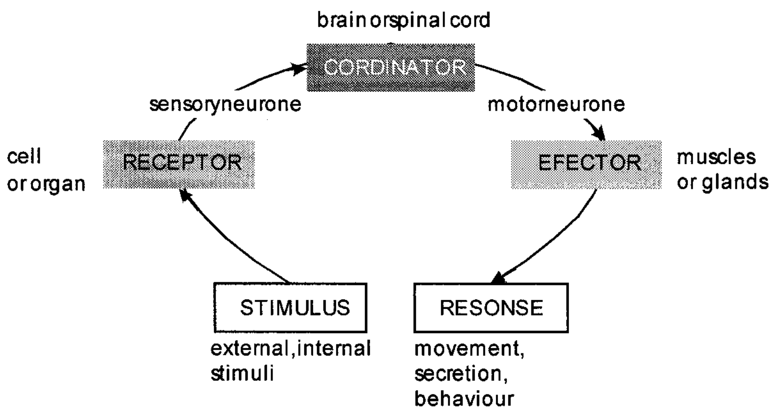


Figure 5: Flow diagram of a simple arc reflex

The human autonomic nervous system is subdivided into two major parts:

a. Sympathetic and

b. Parasympathetic.

The two systems are functionally integrated with each other to regulate activity of the different organs, although the function regulated by the two components are usually *opposite* in nature [4].

Let us take as an example the normal the cardiac autonomic function. The two components of the autonomic nervous system work in careful equilibrium to regulate the second-to-second activity of the heart and the blood vessels. This modulation of autonomic tone is influenced by:

(1) physiologic parameters that function as *afferent* components of a reflex arc. Common examples are **(a)** baroreceptors (sensitive to pressure or stretch) and **(b)** chemoreceptors (sensitive to changes in specific chemicals) and

(2) factors that function as *efferent* responses to these reflexes as are for example: **(c)** the source of the stimulus, **(d)** input from central nervous system, **(e)** state of responsiveness of the receptors, **(f)** influence from other chemical substances and **(g)** the interaction from more than one reflex arc.

This highly complex interaction at multiple levels determines the overall efferent activity of the autonomic nervous system and the cardiovascular response [3].

3.3 Characteristics of Autonomic Computing Systems

Autonomic computer systems are computer systems that can govern themselves much in the same way as our autonomic nervous system regulates and protects our bodies. Autonomic computing system components and the system, as a whole should be capable to anticipate computer systems needs and resolve problems when they appear without or very little human intervention. In 2001, Paul Horn mentioned eight general characteristics [9], which define autonomic computing as follows:

1. **To be autonomic a computing systems needs to "know itself"** – and comprise components that also possess a system identity.
2. **An autonomic computing system must configure and reconfigure itself** under varying and unpredictable conditions.
3. **An autonomic computing system never settles for the status quo –** it always looks for ways to optimize its workings.
4. **An autonomic computing system must perform something akin to healing –** it must be able to recover from routine and extraordinary events that might cause some of its parts to malfunction.
5. A virtual world is no less dangerous than the physical one, **so an autonomic system must be an expert in self-protection.**
6. **An autonomic computing system knows its environment and the context surrounding its activity,** and acts accordingly.
7. **An autonomic system cannot exist in a hermetic environment.**

8. Perhaps the most critical for the user, **an autonomic computing system will anticipate the optimized resources needed while keeping its complexity hidden.**

Paul Horn's definitions of autonomic computing systems characteristics have evolved since 2001. The table below, based on a survey performed by Paul Lin, Alexander MacArthur and John Leaney [24] summarizes the most commonly autonomic computing characteristics being used today.

First Author	Characteristics of Autonomic Computing
Horn	see description of characteristics from 1 to 8 above
Kephart	self-management, self-configuration, self-optimization, self-healing, self-protection
Sterrit	dependability, self-aware, self-managing, self healing, self-protecting, self-optimizing
Ganek	self-management, self-configuration, self-optimization, self-healing, self-protection
Kaiser	self-configuring, self-healing, self-optimizing, self-managing
Agarwal	open, self-defining, context-aware, anticipatory, self-adapting, self-composing, self-optimizing, self-configuring
Trumler	context-aware, anticipatory, self-healing, self-configuration
Warlop	self-configuration, self-optimization, self-healing, self-protection, self-management

Table 1: Different characteristics for Autonomic Computing [24]

Furthermore, in the article called "*Defining Autonomic Computing: A Software Engineering Perspective*" the authors mentioned above have tried to establish a standardized definition of Autonomic Computing and its characteristics through the application of the Quality Metrics Framework (QMF) defined in IEEE Std 1061-1998. While following the steps of Quality Metrics Methodology they came up with the following table that contains quality factors, which are in fact the characteristics of autonomic computing.

Quality Factor	Definition
Anticipatory	The Autonomic Computing system must have a projection of the user needs and actions in the future.
Context-awareness	The Autonomic Computing system must find and generate rules for how best to interact with neighboring systems.
Openness	The Autonomic Computing system must function in a heterogeneous world and implement open standards.
Self-awareness	The Autonomic Computing system must be aware of its internal state
Self-configuring	The Autonomic Computing system must adapt automatically to the dynamically changing environments.
Self-healing	The Autonomic Computing system must detect, diagnose, and recover from any damage that occurs.
Self-management	The Autonomic Computing system must free system administrators from the details of system operation and maintenance.
Self-optimizing	The Autonomic Computing system must monitor and tune resources automatically.
Self-protection	The Autonomic Computing system must detect and guard itself against damage from accidents, equipment failure, or outside attacks by hackers and viruses.

Table 2: Autonomic Computing characteristics identified as quality factors [24]

Then, by applying the quality factors from above table to the generic Quality Metrics Framework found in IEEE Std 1061-1998, they obtained the Quality Metrics Framework for Autonomic Computing as shown in the following figure:

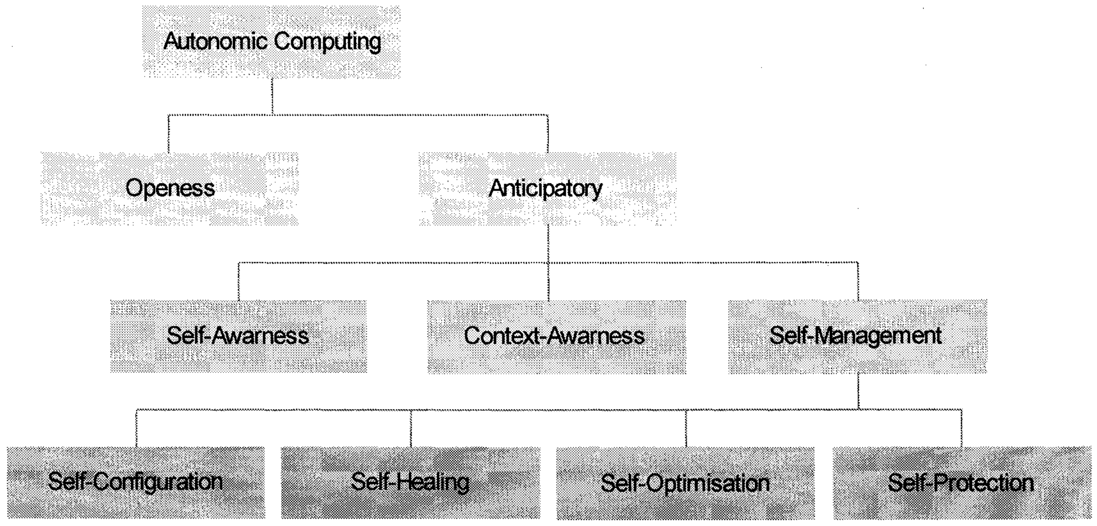


Figure 6: AC characteristics mapped to quality metrics framework [24]

According to the above figure, at the core, an autonomic system must have the following characteristics: self-configuration, self-healing, self-optimization and self-protection.

Traditionally, IBM illustrates the core self-management characteristics with the following figure:

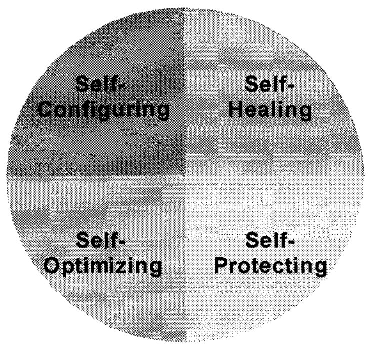


Figure 7: Autonomic systems core characteristics [5]

3.3.1 Self-configuration

Definition

Self-configuration is the capability of an autonomic computing system to automatically initially self-configure and then later to self-re-configure under unpredictable and unexpected system conditions while assessing risks involved. Self-configuration can be extended to contact external services, if needed. In an autonomic computing system self-configuration spawns over autonomic components and autonomic system as a whole and follows high-level information technology policies [5,6,7].

According to the definition, in an autonomic computing system self-configuring components and the system adapt dynamically to (a) the initial installation and configuration and (b) the subsequent maintenance. The latter addresses deployment of new components or removal of existing ones and increase or decrease in workload.

As opposed to an autonomic configuration, a standard complex systems configuration demands many hours of work for information technology personnel being subject to errors as well. Any software or application installation demands detailed planning, discussion and preparation and detailed testing to ensure that new configuration will run at the desired levels. Today, there are numerous install, configuration and maintenance procedures. The differences of system administration tools and their distribution packaging format make managing configuration of complex systems, where application functionality can be composed dynamically, very difficult.

In an autonomic system that implements self-configuration a common solution knowledge capability eliminates the complexity mentioned above by capturing install and configuration information. Solutions are combination of platform capabilities and application elements to solve a particular customer's problem. For example, IBM Trivoli Configuration Manager [18] is an integrated inventory and software distribution solution where self-configuration is possible by using software package reference models to match desired software configuration.

The following paragraphs abstract out some of the research and development efforts underway mainly within IBM for self-configuring autonomic systems.

In the article "*A System Model for Dynamically Reconfigurable Software*" [11], the difficulties of dynamic reconfiguration of software are examined. The authors K. Whinsnant, Z. T. Kalbarczyk and R.K. Tyer have come to the conclusion that in order to define a workable reconfiguration model, both (1) static structure and (2) run time behavior must be captured.

In the article "*Dynamic Reconfiguration: Basic Building blocks for autonomic computing on IBM pSeries servers*" [12] the authors J. Jann, L.M. Browning, R. S. Burgula describe Dynamic Logical Partitions (DLPR) and Dynamic Reconfiguration (DR) technologies that have enabled IBM pSeries 690 server to become truly autonomic computer servers. As mentioned by authors these servers are self-protecting and self-healing and have basic building blocks for self-configuration and self-optimization.

The paper *"Enabling autonomic behavior in systems software with hot swapping"* [13] presents hot swapping as a technique for leveraging autonomic computing in systems software. The authors J. Appavoo, K. Hui, C.A.N. Soules, R.W. Wisniewski, D.M. Da Silva, O. Krieger, M.A. Auslander, D.J. Edelsohn, B. Gamsa, G.R. Gagner, P. McKennedy, M. Ostrowski, B Rosenburg, M Stumm and J. Xenidis have developed a prototype called K42 which is a research operating system. K42 is Open Source Software that explicitly supports self-configuration by interposition and replacement of active operating system code.

An autonomic approach to network service deployment that scales to large, heterogeneous networks is explored in the article *"Autonomic service deployment in networks"* [14]. R. Haas, P Droz and B. Stiller introduce a two-phase intelligent network service deployment: (1) a macro-level operating in a hierarchical distributed way to query and collect capabilities of the nodes in the network, (2) a micro-level that refines installation according to custom capabilities of each network component.

Craig Boutilier, Rajarshi Das, Gerald Tesauro, Jeffrey O. Kephart and William E. Walsh in the paper "*Cooperative Negotiation in Autonomic Systems using Incremental Utility Elicitation*" [15] claim that cooperative negotiation using incremental elicitation is required to perform resource allocation in a distributed autonomic system. To support their claim the authors present algorithms for computing minimax regret and two elicitation strategies. They use an automated resource manager that allocates resources to workload managers in order to maximize total organizational utility and solve the resource allocation problem.

R. Buyya, H. Stockinger, J. Giddy, and D. Abramson provide economic models, system architecture, and policies for resource management in their article *"Economic Models for Management of Resources in Peer-to-Peer and Grid Computing"* [29].

R. Haas, P. Droz and B. Stiller examine management issues related to topology, service placement, cost and service metrics in an intelligent and heterogeneous network infrastructure. Their paper is called "*Autonomic Service Deployment in Networks*" [23].

L. Paulson in his article "*Computer System, Heal Thyself*" [30] considers topology based adaptation. In his approach self-configuration is concerned with physical design and deployment and their aspects: static i.e. physical topology and dynamic i.e. adapting to changes from initial state. The project called LAMDA (Lights-out, Automated Management of Distributed Applications) is based on an adaptation of the Hierarchical Queuing Petri Nets to model the environment.

3.3.2 Self-healing

Definition

Self-healing is the characteristic of autonomic systems to automatically self-detect self-diagnose and self-repair software and hardware problems. That is to say that the system is capable to recover from events that cause system failures or operational malfunctions, by recognizing the problems and being able to find their solutions. Self-healing includes the capability to contact external services in case of new problems and to learn those new problems and their resolutions [5,6,7].

The following algorithm shows a systemic approach to self-healing:

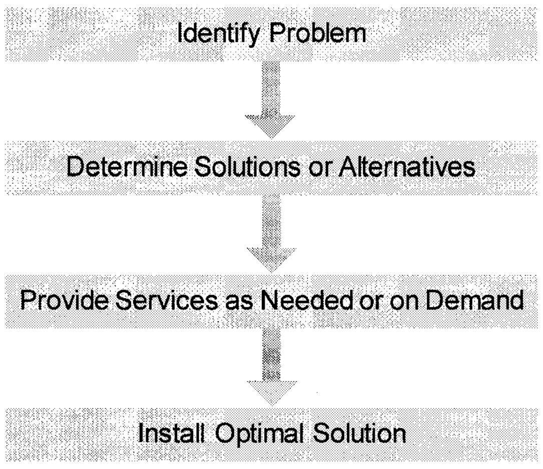

Identify Problem
Determine Solutions or Alternatives
Provide Services as Needed or on Demand
Install Optimal Solution

Figure 8: Self-healing algorithm [1]

More advanced self-healing systems can incorporate a proactive approach in which they can anticipate problems and implement actions accordingly [1].

By contrast, current standard approaches to healing computing systems has left the load to the information technology personnel with strong analytical skills who spends hours in identifying problems by looking at traces, code dumps and log files.

The paper *"Affect and machine design: Lessons for the development of autonomous machines*" [16] sets the stage for future research on how the study of *affect* [16] in biological systems might contribute in developing complex autonomic systems that must deal with unpredictable situations. The authors D.A. Norman, A.Ortony, D.M. Russell suggest that *affect* can improve overall systems behavior and point out that lack of warning is a common problem in automated systems.

In October 2003, IBM issued a paper called "*Automating problem determination: A first step toward self-healing computing systems*" [17]. Researchers at IBM propose a problem determination methodology and architecture that standardize log/trace format, content and organization. In IBM vision, autonomic computer systems that implement self-healing are based on a common problem determination architecture in order to be able to identify the problem, implement solutions in order to perform complex analysis and require rules and criteria to be able to find best solutions.

The article "*Toward a new landscape of systems management in an autonomic computing environment*" [18] presents IBM Trivoli Monitoring which implements the self-healing algorithm. G. Lafranchi, P. Della Peruta, A Perrone and D. Calvanese propose an approach to system self-healing autonomic systems based on "resource model" concept and on System Management Ontology as a way in representing Common Information Model Constructs.

D.C Verma, S. Sahu, S. Calo, A. Shaikh, I. Chang and A. Acharya propose a self-healing method that automates mirroring and replication of applications in a network of servers. In their article *"SRIRAM: A scalable resilient autonomic mesh"* [19] they describe a design based on an autonomic, self-configuring mesh of computers and a communication mechanism between nodes that operates on a rooted spanning tree.

3.3.3 Self-optimization

Definition

The self-optimizing characteristic is the property to automatically self-monitor and self-tune resources triggering actions driven by the type of tuning needed. In an autonomic system, the components and systems continually look for opportunities to improve their own performance and efficiency. Self-optimization translates itself into a high standard of service and in the end into quality of service (QoS) [5,6,7].

According to the definition, a self-optimizing system is capable to: (1) assign a solution or additional resources in order to complete a user transaction in a given time, (2) adapt to dynamically changing workload, and (3) improve overall utilization of the system.

In a computer system without this autonomic feature of self-optimizing all the tuning parameters are changed manually by human intervention. For example, databases and web servers have hundreds of tuning parameters, each new release introducing new ones. Software tools that exist today to monitor and maintain system performance are sophisticated, based on algorithms and mathematical solutions such as linear programming and modeling tools. Human intervention with knowledge of programming and extensive training is needed, in order to use those tools. Adding to this the complexity of heterogeneous distributed real time systems makes the task of system optimizing almost impossible.

In the following paragraphs we are going to outline some of the resulting solutions, algorithms and limitations from research work conducted so far with respect to self-optimization characteristic of autonomic systems.

D. M. Yellin in the paper "*Competitive Algorithms for the Dynamic Selection of Component Implementation*" [20] proposes adaptive components as framework for component based development. An adaptive component has multiple implementations; each optimized for a particular request workload. The author claims that dynamic switching between implementations at run-time will become a useful self-optimizing tool in autonomic computing.

Statistical modeling, tracking and forecasting techniques borrowed from econometrics are explored in "*Clockwork: A new movement in autonomic systems*" [21] to yield a predictive autonomic system which regulates its behavior in anticipation of need. The authors L.W. Russel, S.P. Morgan and E.G. Chron claim that systems using the Clockwork method detect and forecast cyclic variations on the future performance, and use data to reconfigure themselves in anticipation of need.

The self-optimizing feature of autonomic computing is detailed in the article "*LEO: An Autonomic Query Optimizer for DB2*" [21]. The authors V. Markl, G. M. Lohman and V

Rahman describe LEO, a learning optimizer and its two essential functions: differed feedback based learning for future queries and current query progressive optimization.

Y. Diao, J.L.Hellerstein, S Parekh and J.P Bigus describe intelligent agents, that make use of control theories techniques to autonomic adjust a web server to dynamic workloads, in their article "*Managing Web Server Performance with AutoTune Agents*" [26].

According to IBM, autonomic systems in order to implement the self-optimizing characteristic have to:

"*Institute an end-to-end transaction management infrastructure that gives knowledge of how the systems involved commit their resources to execute the workload and how changes in allocation affect performance overtime. This distributed workload management can optimize work across the distributed infrastructure in an attempt to meet all the goals associated with each service. If not all the goals can be achieved, then automatic changes can be made to ensure that the most important goals can be achieved. Eventually this enables the system to self-optimize to meet the business requirements.*" [17]

IBM Trivoli Monitoring [17] implements this autonomic characteristic using templates applied against a resource model engine. In a distributed environment, a server with one-application instance sees increasing usage of a buffer pool. Instead of just raising an event to a management console, Trivoli monitoring will notice the "increased in usage, check processor utilization, check for available system memory and automatically increase the memory pool for the application. As demand changes and usage of the pool decreases it automatically lowers the pool ceiling to return system memory to the server" [17].

3.3.4 Self-protection

Definition

Self-protection characteristic of autonomic systems is the property to automatically self-detect, self-identify and self-defend against external or internal threats, malicious attacks or failures [5,6,7]. More specifically, this characteristic of autonomic systems protects against unauthorized access and use, worms, viruses, denial of service attacks and internal threats. In a nutshell, self-protecting capabilities allow autonomic systems to consistently exert infrastructure security and privacy policies.

In systems that do not implement self-protecting feature, a major cause of poor quality of information is the diversity in the format and content provided by different technologies. A typical company might receive more than 300,000 heterogeneous security threats per day [2] from various security products that need to be analyzed by information technology personnel.

For example, IBM Risk Manager [2] developed solution offers a self-protecting core technology that is designed to protect systems infrastructure by integration of many technologies. Integration of anti-virus, firewalls, routers, virtual private networks, web servers and operating systems can "speed response to real threats by filtering in 30 meaningful threats a day out of 300000 heterogeneous security events per day" [2].

Security and privacy issues in the modeling and development of autonomic systems are discussed in the article "*Security in an Autonomic Computing Environment*" [25]. D. M. Chess, C.C. Palmer, and S.R. White provide some recommendation of autonomic principles usage in order to make systems more secure.

3.3.5 Autonomic Computing: Related Terms and Technologies

Autonomic Characteristics vs. Usability

Traditionally, usability translates itself into ease of use, easy to learn and easy to maintain. In recent years, there has been a major effort in usability analysis and analysts, designers and developers focused heavily on User Centered Design. Following, we provide some highlights from articles we surveyed that study how autonomic computing can provide a framework for usability.

In the paper "*Usability and design consideration for an autonomic relational database management system*" [27], the ease-of-use ramifications of autonomic computing are examined in the context of relational databases. R. Telford, R. Horman, S. Lightstone, N. Markov, S. O'Connell and G. Lohman propose autonomic computing as a new approach to usability focused on self-management rather than of the simplicity of the user-interface. The following advantages are mentioned in the paper (1) reducing the number of low-level system administrator tasks, (2) handling exceptions which otherwise would result in system wide alerts and (3) learning by the system of action taken by the administrator. There is also an interesting point made by the authors which is that human intervention must still be factored in, and care must be taken in the design of autonomic systems not to make the system administration more difficult [27].

On the same topic, in the article "*Dealing with Ghosts: Managing the user experience of autonomic computing*" [31] the authors state that although the goal of autonomic computing is to make systems to work continuously, robustly and simply, people interaction with computer systems will not be eliminated completely. Moreover, D. M. Russel, P.P. Maglio, R. Dordick and C. Neti argue that autonomic systems require an even greater attention to the design of the user interaction.

Usability from the complex systems administrator's perspective, is analyzed by Rob Barrett, Paul P. Maglio, Eser Kandogan and John Bailey in their paper called "*Usable Autonomic Computing Systems: the Administrator's Perspective*" [32]. As a result of their analysis they propose guidelines for constructing autonomic computing systems supporting the following administrator's activities: (1) rehearsal and planning, (2) maintaining situation awareness and (3) managing multitasking, interruptions and diversions.

Autonomic Characteristics vs. Reliability

Reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment [33]. The reliability of complex systems is not only essential, but also critical because of the high costs and catastrophic consequences associated with failure and fault recovery. "It is estimated that companies spend 33-50% of their total cost of ownership recovering from or preparing against failures" [33]. T. Marshall and Y.S. Dai state that improving systems reliability through autonomic computing will have a tremendous economic, security and safety impact. In their article "*Reliability Improvement and Models in Autonomic Computing*" [33], the authors study technologies and theories proposed for autonomic computing with a focus on those that enhance reliability of a complex system. Moreover, they propose a preliminary hierarchical layered design for reliable autonomic complex systems and suggest that monitoring plays an important role in the implementation of such systems.

Autonomic Characteristics vs. Dependability

"Dependability is a long – standing desirable property of all computer-based systems" [34]. Roy Sterritt and Dave Bustard discuss the relation between dependability and autonomic computing in their article *"Autonomic Computing – a Means of Achieving Dependability?"* [34]. The authors state that they believe autonomic computing through self-healing, self-configuring, self-optimizing and self-protecting characteristics will in fact increase dependability.

Autonomic vs. Personal Computing

Personal computing is distinct from servers, centralized storage elements or network infrastructures because of (1) more variable and less secure applications and connectivity and (2) less skilled personnel [35]. With this respect, D.F. Bantz, C. Bisdikian, D. Challener, J. P. Karidis, S. Mastriani, A. Mohindra, D. G. Shea and M. Vanover note that autonomic personal computing represents a distinct part of autonomic computing concept in their article "*Autonomic personal computing*". Moreover, they define autonomic personal computing as personal computing on autonomic computing systems [35] and they map the autonomic characteristics to personal computing as follows:

(a) Self-configuration is the property of autonomic personal computing to (1) self-automate installation and configuration of software according to the needs of platform and (2) to accommodate automatically user-initiated configuration changes.

(b) Self-healing is the characteristic of autonomic personal computing to (1) self-monitor, (2) self-detect errors and (3) automatically self-resolve the errors detected.

(c) Self-optimizing characteristic of autonomic computing systems translates itself into optimizing its own resources.

(d) Self-protecting is the characteristic of autonomic personal computing that makes it achieve security, privacy, function and data protection. This characteristic addresses the distinct attribute of personal computing mentioned earlier.

Autonomic vs. Smart Adaptive Computing

Autonomic computing systems and Smart Adaptive Systems [39] share the required self-adaptive behavior. The latter has been classified into the following levels by the Smart Adaptive Systems field:

(1) adaptation to a changing environment,

(2) adaptation to a similar setting without explicitly being ported to it, and

(3) adaptation to a new/unknown application.

Autonomic vs. Autonomous Computing

Tom De Wolf and Tom Holvoet in their article "*Towards Autonomic Computing: Agent-Based Modeling, Dynamical Systems Analysis, and Decentralized Control*" [36] express the idea that autonomic computing aims to deal with the complexity of today systems by letting the system handle the complexity autonomously [36]. In other words, they define autonomic system as the ones able to take the initiative to decide and do things themselves. To be autonomic, in their vision, the systems must "become autonomous such that real complexity can be hidden from the user of the system" [36]. Furthermore, to support their idea, the authors postulate that a notable contribution in development of complex autonomic systems can be made by the integration of:

(1) multi-agent systems which enable natural modeling of the system, autonomous behavior and distributed interaction,

(2) dynamic systems theory which enable analysis of the dynamics of the models, and

(3) decentralized control theory, to control the dynamics of autonomic systems.

Autonomic vs. Proactive Computing

Intel Research is exploring another computer paradigm called proactive computing. In their vision proactive computing "*overlaps autonomic computing but at the same time enables the transition from interactive computing systems to systems that anticipate our needs and act on our behalf*" [28].

The following figure illustrates the relationship between the two computing paradigms:

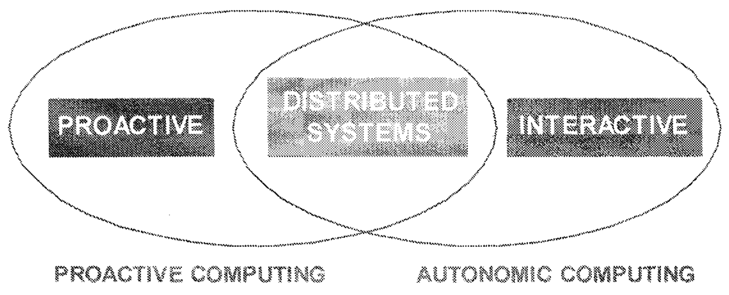


Figure 9: The relationship between proactive and autonomic computing [28]

The goals of proactive computing are the following (1) connecting to the physical world (2) real-time and closed loop operation (3) techniques that allow computers to anticipate user needs, (4) addressing security and privacy concerns, (5) dealing with uncertainty, (6) planetary scale systems and (7) deep networking.

In the article "*Comparing autonomic and proactive computing*" [28], the authors R. Want, T. Pering and D. Tennenhouse provide a detailed overview of first three proactive computing characteristics mentioned above.

An interesting note here is that a true autonomic computing system may incorporate proactive computing for example through evolutionary learning.

Autonomic vs. Introspective Computing

In the article *"Introspective Failure Analysis: Avoiding Correlated Failures in Peer-to-Peer Systems*" [37], H. Watherspoon, T. Moscovitz and J. Kubiatowicz state that autonomic computing implies a system reacting to events whereas introspective computing involves both reactive and proactive behavior.

3.4 Conclusion

In this chapter, we surveyed autonomic characteristics: self-configuring, self-healing, self-optimization and self-protection in detail. It is important to note that those attributes are not separated of one another; they overlap or differ with respect to aptitudes or motivation and jointly form the core that enables the true autonomic capability much sought for complex systems.

References

[1] American Heritage Dictionary of the English Language: Fourth Edition, 2000.

[2] Ackerknecht EH, "*The history of the discovery of the vegetative (autonomic) nervous system*", Medicine History, Volume 18, pp. 1-49, 1974.

[3] Robert W. Teasell, MD, FRCPC, "*The Autonomic Nervous System Physical Medicine and Rehabilitation*", Volume 10, No. 1, pp. 1-27, February, 1996.

[4] Richard Murch, "*Autonomic Computing*", Prentice Hall Professional Technical Reference, IBM Press, pp. 119-132, 2004.

[5] IBM Corporation, "*An architectural blueprint for autonomic computing*", IBM and autonomic computing, April 2003, available at:

http://www-03.ibm.com/autonomic/pdfs/ACwpFinal.pdf

[6] IBM Corporation, "*An architectural blueprint for autonomic computing*", Autonomic Computing White Paper, October, 2004, available at:

http://www-03.ibm.com/autonomic/pdfs/ACBP2_2004-10-04.pdf

[7] IBM Corporation, "*An architectural blueprint for autonomic computing*", Autonomic Computing White Paper, June, 2005, available at:

http://www-

03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf

[8] A. G. Ganeck, T.A. Corbi, "*The Dawning of the autonomic computing era*", IBM Systems Journal, Volume 42, No. 1, 2003.

[9] Paul Horn, "*Autonomic Computing: IBM's Perspective on the State of Information Technology*", IBM Corporation, October 15, 2001, available at:

http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf

[10] Jeffrey O. Kephart, David M. Chess, "*The Vision of Autonomic Computing*", Published by IEEE Computer Society, Volume 36 (1), pp. 41-50, 2003, available at:

http://www.research.ibm.com/autonomic/research/papers/AC_Vision_Computer_Jan_20

03.pdf

[11] K. Whinsnant, Z. T. Kalbarczyk, R.K. Tyer, "*A System Model for Dynamically Reconfigurable Software*", IBM Systems Journal, Volume 42, No. 1, 2003.

[12] J. Jann, L.M. Browning, R. S. Burgula, "*Dynamic Reconfiguration: Basic building blocks for autonomic computing on IBM pSeries servers*", IBM Systems Journal, Volume 42, No.1, 2003.

[13] J. Appavoo, K. Hui, C.A.N. Soules, R.W. Wisniewski, D.M. Da Silva, O. Krieger, M.A. Auslander, D.J. Edelsohn, B. Gamsa, G.R. Gagner, P. McKennedy, M. Ostrowski, B Rosenburg, M Stumm, J. Xenidis, "*Enabling autonomic behavior in systems software with hot swapping*", IBM Systems Journal, Volume 42, No.1, 2003.

[14] R. Haas, P Droz, B. Stiller, "*Autonomic service deployment in networks*", IBM Systems Journal, Volume 42, No.1, 2003.

[15] Craig Boutilier, Rajarshi Das, Gerald Tesauro, Jeffrey O. Kephart, William E. "*Cooperative Negotiation in Autonomic Systems using Incremental Utility Elicitation*", IBM Autonomic Computing White Paper, 2003, available at: http://www.research.ibm.com/autonomic/research/papers/utility_elicitation_UAI03.pdf

[16] D.A. Norman, A.Ortony, D.M. Russell, "*Affect and machine design: Lessons for the development of autonomous machines*", IBM Systems Journal, Volume 42, No.1, 2003.

[17] IBM Corporation, "*Automating problem determination. A first step towards self-healing computing systems*", IBM Autonomic Computing White Paper, October 2003, available at: http://www-03.ibm.com/autonomic/pdfs/Problem_Determination_WP_Final_100703.pdf

[18] G. Lafranchi, P. Della Peruta, A. Perrone, D. Calvanese, "*Toward a new landscape of systems management in an autonomic computing environment*", IBM Systems Journal, Volume 42, No.1, 2003.

[19] D.C Verma, S. Sahu, S. Calo, A. Shaikh, I. Chang, A. Acharya, "*SRIRAM: A scalable resilient autonomic mesh*", IBM Systems Journal, Volume 42, No.1, 2003.

[20] D.M. Yellin, "*Competitive Algorithms for the Dynamic Selection of Component Implementation*", IBM Systems Journal, Volume 42, No.1, 2003.

[21] L.W. Russel, S.P. Morgan, E.G.. Chron, "*Clockwork: A new movement in autonomic systems*", IBM Systems Journal, Volume 42, No.1, 2003.

[22] V. Markl, G. M. Lohman, V. Rahman, "*LEO: An Autonomic Query Optimizer for DB2*", IBM Systems Journal, Volume 42, No.1, 2003.

[23] R. Haas, P. Droz, B. Stiller, "*Autonomic Service Deployment in Networks*", IBM Systems Journal, Volume 42, No.1, 2003.

[24] Paul Lin, Alexander MacArthur, John Leaney, "*Defining Autonomic Computing: A Software Engineering Perspective*", Proceedings of the 2005 Australian Software Engineering Conference (ASWEC'05), 2005 IEEE.

[25] D.M.. Chess, C.C. Palmer, S.R. White, "*Security in an Autonomic Computing Environment*", IBM Systems Journal, Volume 42, No.1, 2003.

[26] Y. Diao, J.L.Hellerstein, S Parekh, J.P Bigus, "*Managing Web Server Performance with AutoTune Agents*", IBM Systems Journal, Volume 42, No.1, 2003.

[27] R. Telford, R, Horman, S. Lightstone, N. Markov, S. O'Connell, G. Lohman, "*Usability and design consideration for an autonomic relational database management system*", IBM Systems Journal, Volume 42, No.1, 2003.

[28] R. Want, T. Pering, D. Tennenhouse, "*Comparing autonomic and proactive computing*", IBM Systems Journal, Volume 42, No.1, 2003, additional information can be found at:

http://www.intel.com/research/exploratory/

[29] R. Buyya, H. Stockinger, J. Giddy, D. Abramson, "*Economic Models for Management of Resources in Peer-to-Peer and Grid Computing*", SPIE International Symposium at the Convergence of Information Technologies and Communications, pp. 1-13, August, 2001.

[30] L. Paulson, "*Computer System Heal Thyself*", IEEE Computer, Volume 35, No. 8, pp. 20-22, August, 2002.

[31] D. M. Russell, P. P. Maglio, R. Dordick, C. Neti, "*Dealing with ghosts: Managing the user experience of autonomic computing*", IBM Systems Journal, Volume 42, No.1, 2003.

[32] Rob Barrett, Paul P. Maglio, Eser Kandogan, John Bailey, *"Usable autonomic Computing Systems: the Administrator's Perspective"*, Proceedings of the International Conference on Autonomic Computing (ICAC'04), pp. 18-26, 2004.

[33] T. Marshall, Y. S. Dai, *"Reliability Improvement and Models in Autonomic Computing",* Proceedings of the 2005, 11th International Conference on Parallel and Distributed Systems (ICPADS'05), pp. 468-472, 2005.

[34] Roy Sterrit, Dave Bustard, "*Autonomic Computing – A Means of Achieving Dependability*", Proceedings of the 10th IEEE International Conference and Workshop of the Engineering of Computer-Based Systems (ECBS'03), pp. 247-251, 2003.

[35] D.F. Bantz, C.Bisdikian, D. Challener, J. P. Karidis, S. Mastriani, A. Mohindra, D. G. Shea, M Vanover, "*Autonomic personal computing*", IBM Systems Journal, Volume 42, No.1, 2003.

[36] Tom De Wolf, Tom Holvoet "*Towards Autonomic Computing: Agent-Based Modeling, Dynamical Systems Analysis, and Decentralized Control*", Proceedings of the First International Workshop on Autonomic Computing Principles and Architectures, pp. 10, 2003, available at:

http://www.cs.kuleuven.ac.be/~tomdw/publications/

[37] H. Watherspoon, T. Moscovitz, J. Kubiatowicz, *"Introspective Failure Analysis: Avoiding Correlated Failures in Peer-to-Peer Systems*", Proceedings of the 10th IEEE

International Conference and Workshop of the Engineering of Computer-Based Systems, pp. 113-717, October, 2003.

[38] Mark Weiser, "*The Computer for the 21st Century*", Scientific American, pp. 94-104, September, 1991, reprinted in *IEEE Pervasive Computing*, pp. 19-25, January-March, 2002.

[39] H. Lieberman, T. Selker, *"Out of context: computer systems that adapt to, and learn from, context*", IBM Systems Journal, Volume 39, No.3&4, 2000.

Chapter 4 Autonomic Systems Modeling and Development

Abstract

This chapter presents the autonomic computing reference architecture and its details, the autonomic computing reference model with its core development capabilities, and the evolutionary development process of autonomic systems. The architectural concepts presented in this section are primarily based on IBM Corporation's blueprints and on the on-going research for autonomic computing conducted at the IBM's laboratories. The three blueprints published as white papers under the name "*An architectural blueprint for autonomic computing*" in 2003, 2004 and 2005 are overviews of the basic concepts, constructs and behaviors for building autonomic capability into complex computer systems. We will use the concepts presented in this section, later in chapter 6, as a mechanism for discussing, comparing and contrasting the approaches that different vendors use to deliver self-management capabilities in complex computing systems.

4.1 Modeling

At the basis of the autonomic computing architecture is the autonomic computing control loop.

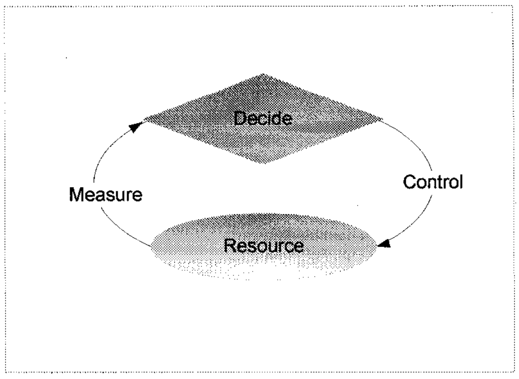


Figure 10: Autonomic computing control loop [1]

A more complex control loop is called an intelligent control loop. Ric Telford, Director of Architecture and Technology Autonomic Computing IBM refers to an intelligent control loop as "*Autonomic Computing Reference Model* " [10].

For simplicity, an intelligent control loop can be represented graphically as:

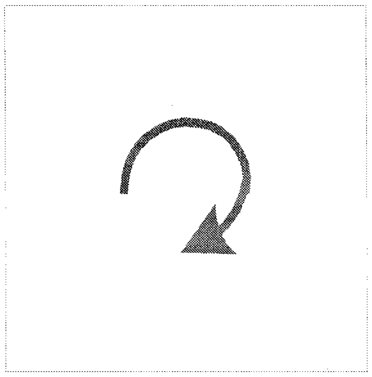


Figure 11: Simplified intelligent control loop representation

4.1.1 Architecture

According to IBM blueprints [1, 2, 3], the autonomic computing architecture is organized into two dimensions: horizontally, into layers corresponding to decision-making contexts and vertically, into parts with distributed infrastructure. The decision-making contexts are used to illustrate the purpose and role of an autonomic element within the autonomic computing architecture. The distributed infrastructure can be visualized as a service bus [2] that connects: (1) managed resources, (2) touchpoints, (3) autonomic managers and (4) an integrated solution console.

The autonomic computing reference architecture is depicted in Figure 12 below.

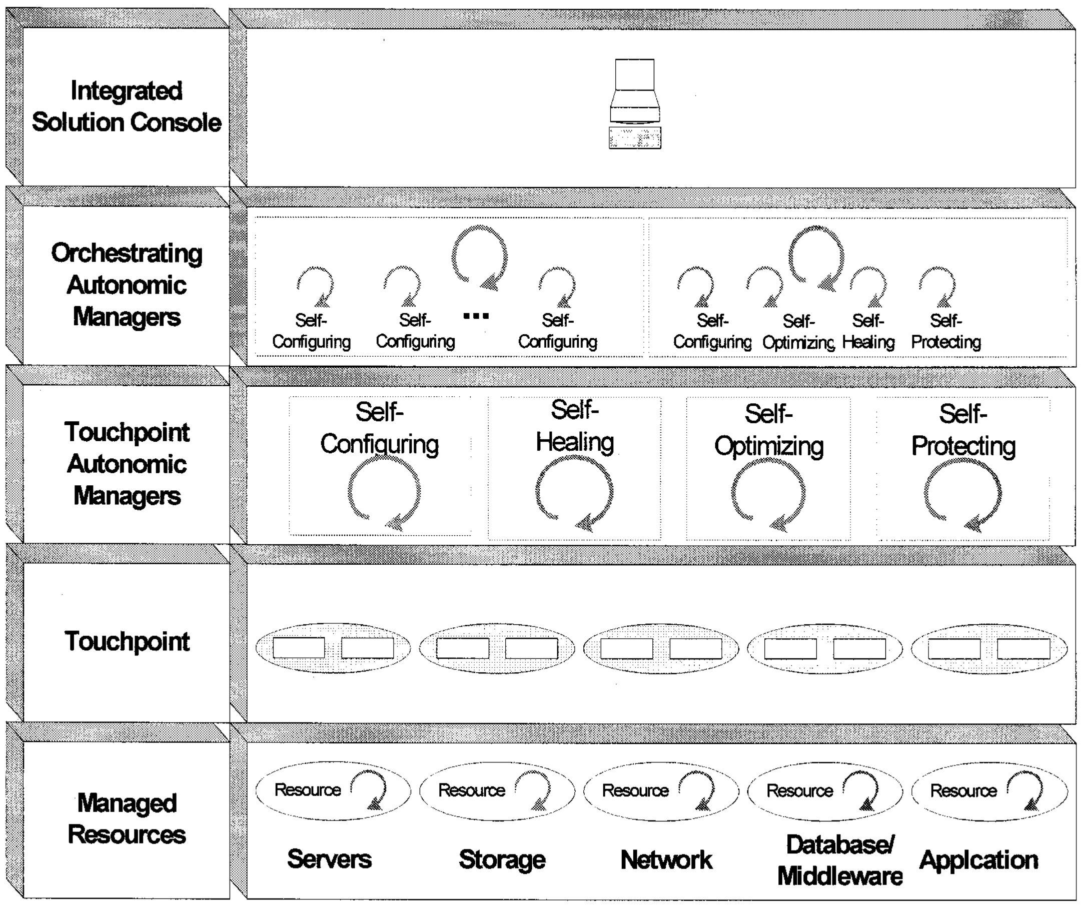


Figure 12: Autonomic computing reference architecture [2]

First Layer - Managed Resources

The first layer contains the managed resources that exist in the real-time environment or in the system environment and that can be managed. An interesting note here is that resources themselves can have embedded autonomic properties.

Second Layer - Touchpoint

The next layer contains standard interfaces called touchpoints. The touchpoints are used for the management of resources from first layer.

Third Layer - Touchpoint Autonomic Managers

Layer three contains autonomic managers each implementing in particular a self-configuring, self-healing, self-optimizing or self-protecting control loop.

Fourth Layer - Orchestrating Autonomic Managers

Layer four contains orchestrating managers. From the architecture above we note that the orchestrating managers can manage autonomic elements of the same type or autonomic elements of different types. This layer provides the system wide autonomic capability because the orchestrating managers have the broadest view of the overall system infrastructure.

Fifth Layer – Integrated Solution Console

The top layer provides an interface for common system management as an integrated solutions console.

4.1.2 Design

As stated above, the basis of the autonomic architecture is the **control loop**. Enhanced with decision-making components that (1) monitor, (2) analyze, (3) plan, and (4) execute using shared knowledge create an **intelligent control loop**. An intelligent control loop is also referred to as **autonomic element** or **autonomic component**. The autonomic component architecture is based on the technique of feedback control optimization rooted on forecasting models [4], which enables the important self-management characteristics of an autonomic system. An autonomic system contains many autonomic components that interact by communicating and negotiating with each other and eventually other types of resources within or outside system boundaries. This is called **autonomic manager collaboration**. The following subsections present in detail the key concepts just mentioned above.

4.1.2.1 Basic Autonomic Control Loop

A basic autonomic control loop was depicted in Figure 10 above, adapted from "*Autonomic Computing Concepts*", IBM White Paper, 2001.

A basic control loop is composed of a managed element called **resource** and a controller called **autonomic manager**. The resource is highly scalable, as it can be anything from a file, server, database, network, middleware, an application or even a business unit. According to the illustration, the autonomic manager makes decisions and controls the resource based on received measurements.

4.1.2.2 Intelligent Control Loop

A more refined control loop is defined as intelligent control loop in autonomic computing. In an intelligent control loop (Figure 13), the autonomic manager (1) monitors, (2) analyses (3) plans and consequently (4) takes actions to achieve the goals and objectives of the managed element. Based on the monitoring and analysis, the autonomic manager can be enhanced to eventually model and learn about the managed element.

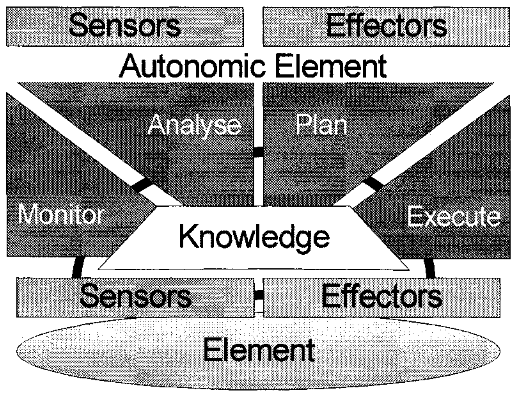


Figure 13: Intelligent control loop [1]

Steve R. White, James E. Hanson, Ian Whalley, David M. Chess, and Jeffrey O. Kephart extend the definition of intelligent control loop. In their article, *"An Architectural Approach to Autonomic Computing*", they define it as:

"*A component that is responsible for managing its own behavior in accordance with policies, and for interacting with other autonomic elements to provide or consume computational services*". [11]

4.1.2.3 Managed Element

The managed element is a controllable autonomic system component. It can be a single resource (a server, a database server, a router, etc) or a collection of resources (a pool of servers, cluster or business application). It is controlled, through autonomic component interface composed by its sensors and its effectors.

4.1.2.4 Sensors

The sensors collect information about the state and state transitions of an autonomic element. The sensors gather information about the current state or events, when the state of the element changes in an important way.

4.1.2.5 Effectors

The effectors change the state (for example the configuration) of an autonomic element in some notable way.

4.1.2.6 Manageability Interface

The sensors and effectors positioned above the autonomic element in Figure 13 constitute the manageability interface of an autonomic element. The architecture proposed by IBM "*encourages the idea that sensors and effectors are connected together. For example a configuration change that occurs through effectors should be reflected as a change notification through the sensor interface*" [1]. This idea is illustrated in Figure 14.

4.1.2.7 Autonomic Manager

The autonomic manager component of an autonomic element implements the control loop. The architecture illustrates the autonomic manager as composed of into four distinct parts that share knowledge namely: (1) monitor, (2) analyze, (3) plan and (4) execute.

Monitor

The monitor collects, aggregates, filters, manages and reports details (metrics and topologies) gathered from the managed element.

Plan

The plan part correlates and models complex situations (time-series forecasting and queuing models, for example). The plan part enables the autonomic manager to learn about the complex system environment and eventually help predict future situations. As well, the plan part provides mechanisms to organize the action needed to achieve goals and objectives. The plan part of an autonomic manager is responsible for interpreting and translating policy details, finally using those policies to lead his work.

Execute

Execute part controls the execution of a plan with respect to real-time updates.

Analysis

The analysis part is responsible to determine if the autonomic manager can abide by the policy, in current and in future situations.

An interesting note is that the four parts work together to provide the control loop functionality. The four parts are shown as a structural arrangement and not as control flow. For example the plan part might ask the monitor part to collect more or less information. Another example is that the monitor part may ask the plan part to create a new plan. "The four parts collaborate using asynchronous communication techniques, like a messaging bus" [1].

4.1.2.8 Shared Knowledge

Metrics, commands, topology information, events, logs, performance data and policies are collected through manageability interface by sensors and effectors and represent the shared knowledge part of an autonomic component. Data stored as shared knowledge is analyzed subsequently by the autonomic manager in the analysis phase. For example, knowledge used by a particular autonomic manager could be created by the monitor part, based on the information gathered by sensors, or passed into the autonomic manager through its effectors. "An example of the former occurs when the monitor part creates knowledge based on recent activities by logging the notification it receives from a managed element into a system log. An example of the latter is policy" [1]. A policy is defined as a set of behavioral constraints that drives the decisions taken by the autonomic manager. Policies are interpreted by the plan part.

4.1.2.9 Autonomic Manager Collaboration

Autonomic managers are required to collaborate through communication and negotiation in order to yield the self-management features of an autonomic element. Furthermore, the numerous autonomic managers in a complex computing system must cooperate to achieve overall system common goals. For example, "a database system needs to work with the server, storage subsystem, storage management software, and other elements of the system in order for the system infrastructure as a whole to become an autonomic complex computing system" [1].

According to the architecture presented earlier, the collaborative interaction between autonomic managers (1) spawns two directions: peer-to-peer and hierarchical, and (2) is enabled by their sensors and effectors (i.e. their manageability interfaces).

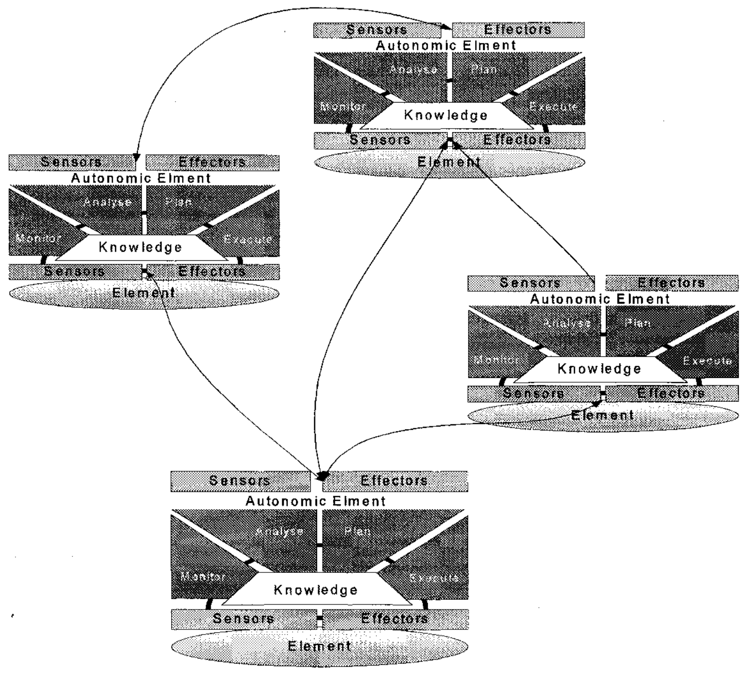


Figure 14: Autonomic manager collaboration [1]

The figure above illustrates how the collaboration of autonomic managers is accomplished through their sensors and effectors, using a "matrix management protocol". This protocol "makes it possible to identify situations in which there are multiple managers situations and enables autonomic managers to negotiate resolutions for domain conflicts, based on a system wide business and resource optimization policy" [2].

4.2 Development

There are presently seven core capabilities available for autonomic manager development [1], namely: (1) policy determination, (2) solution knowledge, (3) common system administration, (4) problem determination, (5) autonomic monitoring, (6) complex analysis and (7) transaction measurement.

4.2.1 Policy Determination

Steve R. White, James E. Hanson, Ian Whalley, David M. Chess, and Jeffrey O. Kephart in their article "*An Architectural Approach to Autonomic Computing*" [11] define policy as follows:

"A representation in a standard form, of desired behaviors or constraints on behaviors."

It is also noted that "*policy-based management of computer system has been an active topic of research for over a decade and that for autonomic computing the focus is on policy-based self-management*" [11].

Policies are basically the key part of the knowledge used by autonomic managers to make decisions because they contain the criteria used to attain goals or determine directions of action. Policies are essentially controlling the planning component of the autonomic manager. The following figure depicts policy management within autonomic components.

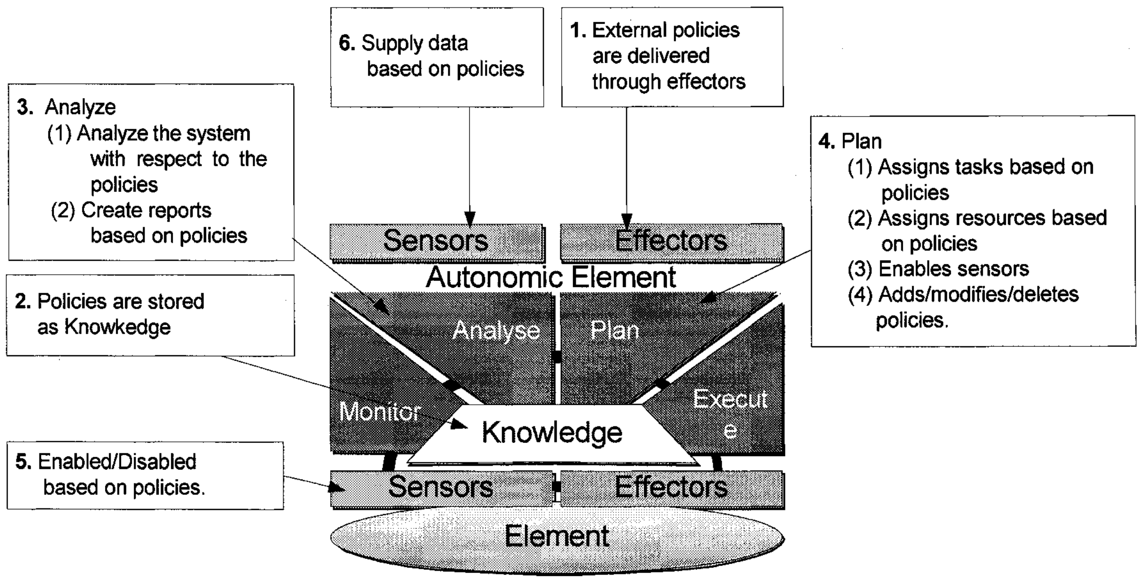

6. Supply data based on policies
1. External policies are delivered through effectors
3. Analyze (1) Analyze the system with respect to the policies (2) Create reports based on policies
4. Plan (1) Assigns tasks based on policies (2) Assigns resources based on policies (3) Enables sensors (4) Adds/modifies/deletes policies.
2. Policies are stored as Knowkedge
5. Enabled/Disabled based on policies.
Sensors
Effectors
Autonomic Element
Analyse
Plan
Monitor
Knowledge
Execut e
Sensors
Effectors
Element

Figure 15: Policy characteristics [2]

As mentioned in the definition, an autonomic complex computing system requires a uniform method for defining the policies that control the decision-making for autonomic managers. By defining policies in a standard way, they can be shared between autonomic managers such that multiple subsystems are managed in a similar manner. In other words, policies must be specified consistently for a complex autonomic system to behave cohesively.

The autonomic computing blueprint is currently defining the specifications and capabilities for policy-based autonomic managers. To date, this definition includes [2]:

- Specification of canonical configuration parameters for management elements.
- Format and schema used to specify user requirements or criteria.
- Mechanisms used, including wire formats, for sharing and distributing policies.

- Schema used to specify and share policies between autonomic managers.

As a conclusion it can be noted that one of the key responsibilities of the autonomic systems is to share policies among autonomic managers, such that in turn this capability will support and extend the policy standards.

4.2.2 Solution Knowledge

Solution knowledge contains many types of data coming from multiple points such as operating systems, application languages, system utilities, performance data, etc. This data can be used in all areas of autonomic computing.

From an autonomic systems perspective, the lack of solution knowledge is a drawback for self-configuring, self-healing and self-optimizing properties. For example, to date there exist an enormous number of maintenance mechanisms for install and configuration. The differences in system administration tools and distribution packaging formats creates unexpected problems in complex systems administration. These problems are further increased in dynamic environments where application functionality can be added/removed dynamically.

A common solution knowledge removes the complexity introduced by differences in formats and install tools. Furthermore, the knowledge acquired in a consistent way can be used by autonomic managers in contexts other than configuration, such as problem determination or optimization for example.

Taken in particular, solutions are combinations of platform capabilities (operating systems and middleware) and application elements. The idea here is to acquire this information to support the install, configuration and maintenance processes at the solution level instead of using proprietary mechanisms.

The autonomic computing blueprint defines a set of constructs for composing installable units and design patterns that make it possible to standardize solution knowledge. There are three categories of installable units [1] that build on each other:

(1) smallest installable unit which contains one atomic item,

(2) container installable unit, which aggregates a set of items for a particular type, and

(3) solution module installable unit which contains many container installable units

Moreover, the autonomic computing blueprint identifies a number of supporting technology components for solution knowledge. To date, these include:

(1) a dependency checker, which checks whether the dependencies are satisfied in the targeted hosting environment,

(2) an installer, which extracts the artifacts in the installable units and invoke necessary operations on the target hosting environment,

(3) an installable unit database, which is a library for installable units,

(4) deploy logic, which distributes an installable unit to an installer component, and

(5) an installed unit "instances" database, which is a database that stores the configuration details about installable units and the hosting environments.

4.2.3 Common System Administration

Common system administration can be achieved by a common console approach and consists of a framework for reuse and a consistent presentation of complex system autonomic properties. Autonomic complex systems require common console technology to create a consistent human-interface for the autonomic components composing the infrastructure of complex systems.

According to IBM blueprints, "*the primary goal of a common console is to provide a single platform that can host all the administrative console functions in server, software and storage products in a manner that allows users to manage solutions rather than managing individual systems or products. Administrative console functions range from setup and configuration to solution runtime monitoring and control. By enabling increased consistency of presentation and behavior across administrative functions, the common console creates a familiar user interface that promotes reusing learned interaction skills versus learning new, different product-unique interfaces*" [3].

Consistency of presentation and behavior drive the improvement of complex systems administrative efficiency, and will definitely "require cooperation among many product communities" [1]. In chapter 6, we will present current approaches taken by IBM, SUN, and Hewlett Packard for console technologies. With this respect the IBM blueprints state that "*console guidelines will take time to emerge, given the large number of human factors and design organizations involved*" [2].

4.2.4 Problem Determination

IBM autonomic blueprint defines a problem as "*a situation in which an autonomic manager needs to take action*" [1]. Autonomic managers take actions based on problems they find with the managed element. With this respect, the first basic capability of autonomic managers is to be able to extract high quality data in order to determine whether or not a problem really exists. The second basic capability of autonomic managers is that once a problem happens, the autonomic manages must be able to classify it.

In current complex computing systems a major cause of poor quality information is the diversity in the format and content of the information provided by the managed element. In addition, there are different nomenclatures to report same situations. For example, one component may report the situation that a "component has started" and another component reports that a "component has begun execution." This diversity in the description of the status makes maintaining complex systems extremely difficult. Another issue with current complex computing systems is the lack of an industry standard for classifying occurring problems.

Autonomic computing blueprints address the variability of collected data by defining common problem determination (PD) architecture that:

1. Normalizes the data collected, in terms of format, content, organization and sufficiency by determining and then defining a base set of data that must be collected or created when a problem or event occurs.

2. Categorizes the collected data into a set of situations.
3. Accommodates legacy data sources (logs and traces) by defining an adapter/agent infrastructure that provides 1) the ability to plug in adapters to transform data from a component specific format to the standard format and 2) sensors to control data collection (filtering, aggregation, etc.).

The IBM article, *"Automating problem determination is the first step towards self-healing computing systems"* [5] addresses in detail the problem determination in autonomic systems. The solutions proposed are technologies for log adapters, log analyzers, autonomic managers, and a symptom database.

4.2.5 Autonomic Monitoring

IBM blueprints of autonomic computing defines autonomic monitoring as "*the capability of the autonomic managers of gathering and filtering data from the sensors*" [1].

Autonomic monitoring capability enables autonomic managers to filter, aggregate and perform a thorough analysis based on collected data in order to detect problems in the systems when they happen. Basically, this capability includes:

(1) a tool to gather the information from the sensors,

(2) a built-in sensor data filtering mechanism,

(3) a pre-defined set resource models and mechanisms for creating new models that enable the description of the state of a logical resource,

(4) a tool to add policy knowledge, and

(5) analysis engines for basic event, cause analysis, server level correlation across multiple complex computing systems, and automate problem resolution.

Wang Fei and Li-Fan-Zhang present an interesting algorithm for decision-making in their article "*The Design of an Autonomic Computing Model and the Algorithm for Decision-Making*" [13]. They have designed the algorithm based on their formalized unified model of autonomic systems.

Autonomic monitoring capability improves the management of applications or resources by (1) allowing processing of data from industry standard interfaces, (2) linking corrective actions to the repeated occurrence of a problem condition, and (3) leveraging origin identification and response to problems.

In the article "*Towards a new landscape of systems management in an autonomic computing environment*" [7] the authors G. Lafranchi, P. Della Peruta, A. Perrone and D. Calvanese present IBM Trivoli monitoring system. This system management application is based on the resource model concept, which is the main tool for implementing an "*identify, notify and cure* " [7] autonomic systems management strategy. Based on the work from the article we introduce in the following two subsections the Resource Model concept and System Management Ontology.
The third following subsection introduces an interesting idea of knowledge discovery that makes use as well of ontology-based software.

4.2.5.1 Resource Model

A resource model is composed of a dynamic model and a reference model. A resource model is created for each problem and it contains the best practices used to identify and correct a well-defined problem.

Dynamic Model

The dynamic model uses Common Information Model (CIM) formalism standardized by the Distributed Management Task Force (DMTF). Here, we provide some basic definitions used in the formalism.

CIM classes and CIM properties are used to describe resources (for example memory) and their performance metrics (for example process working set).

CIM methods are used to describe actions that can be performed against resource for example starting a process.

CIM associations are used to represent the resource within a high level object (logical object)

Reference Model

The reference model is actually composed by "the best practices" used by the system administrators to detect problems and to identify their origin. The reference model is connected to the dynamic model and describes the resources it analyses, the properties of objects aggregated in the model, generates resources status, and triggers actions to correct problems.

4.2.5.2 Systems Management Ontology (SMO)

"*Ontologies are metadata schemas, providing a controlled vocabulary of terms, each with an explicitly defined and machine understandable semantics*" [7].
Traditionally, description logics capture ontologies on a given domain in logical terms. For this reason, their reasoning capabilities can be applied in autonomic monitoring to CIM and resource models, "*allowing for automated use of the knowledge represented in such models and making use of the state of the art reasoning tools developed for expressive description logics*" [7].

The use of ontology is taken even further, by G. Tziallas and B. Theodoulidis in their study "*Building Autonomic Computing Systems Based on Ontological Component Models and a Controller Synthesis Algorithm*" [12]. They use block-diagrams and state-transition diagrams (hierarchical state machines) adapted to the extended ontological Bunge ontology and Bunge-Wand-Weber (BWW) models. Additionally, the authors provide details into Ramadge and Woham supervisory control theory of discrete systems used for supervisory control of their proposed autonomic system.

4.2.5.3 Knowledge Discovery

John Strassner and Barry J. Menich provide a general framework to incorporate knowledge and knowledge discovery in autonomic systems in their article "*Philosophy* *and Methodology for Knowledge Discovery in Autonomic Computing Systems*" [10]. Their framework is based on information models, metadata, an ontology lattice and a hypothesis component and works as follows:

"*A system is loaded with existing business rules that the system must strive to optimize, as well as desired system behavior (e.g. in the form of sets of final state machines). Knowledge exists in two forms – expected (already modeled) and unexpected (not modeled.). The underlying model and ontology can be used to filter data received, matching expected data against its system and analyzing new data to determine if said data should be added in the model or not. Received data is used to deduce the current state of the system, which is compared to the desired state. If the states don't match then the control loop uses policy to generate commands to move the system to the desired state.*" [10]

4.2.6 Complex Analysis

Autonomic managers must collect and operate based on large amounts of data collected from sensors and managed resources. In addition, autonomic managers need to have the capacity to perform complex data analysis and reason based on the data. This data includes information about resource configuration, status, workload and throughput and consequently is static or dynamic. Therefore, an autonomic manager's ability to quickly analyze data is crucial to its successful operation.

Common complex data analysis tasks include classification, clustering of data to characterize complex states and detect similar situations, prediction of anticipated workload and throughput based on past experience, and reasoning for causal analysis, problem determination and optimization of resource configurations. According to IBM blueprints the complex analysis technology:

"*Uses a rule language that supports reasoning through procedural and declarative rule-based processing of managed resource data. The underlying rule engines can be used to analyze data using scripting as well as forward and backward inferencing using if-then rules, predicates and fuzzy logic. Application classes can be imported directly into rule sets so that data can be accessed (using sensors) and control actions can be invoked directly from rules (using effectors). The rules language features both Java programming language-like text and XML source rules end representations, enhancing productivity for rule authors familiar with Java syntax and allowing portable knowledge interchange. Rule sets can include multiple rule blocks so that a mix of procedural and inferencing methods can be used to analyze data and define autonomic manager behavior.*" [1]

Other complex analysis advanced technologies include the following:

- JavaBeans,
- machine learning beans,
- software agents,
- neural networks,
- decision trees and

- bayesian classifiers to perform statistical analysis using numerous genetic algorithms.

4.2.7 Transaction Measurement

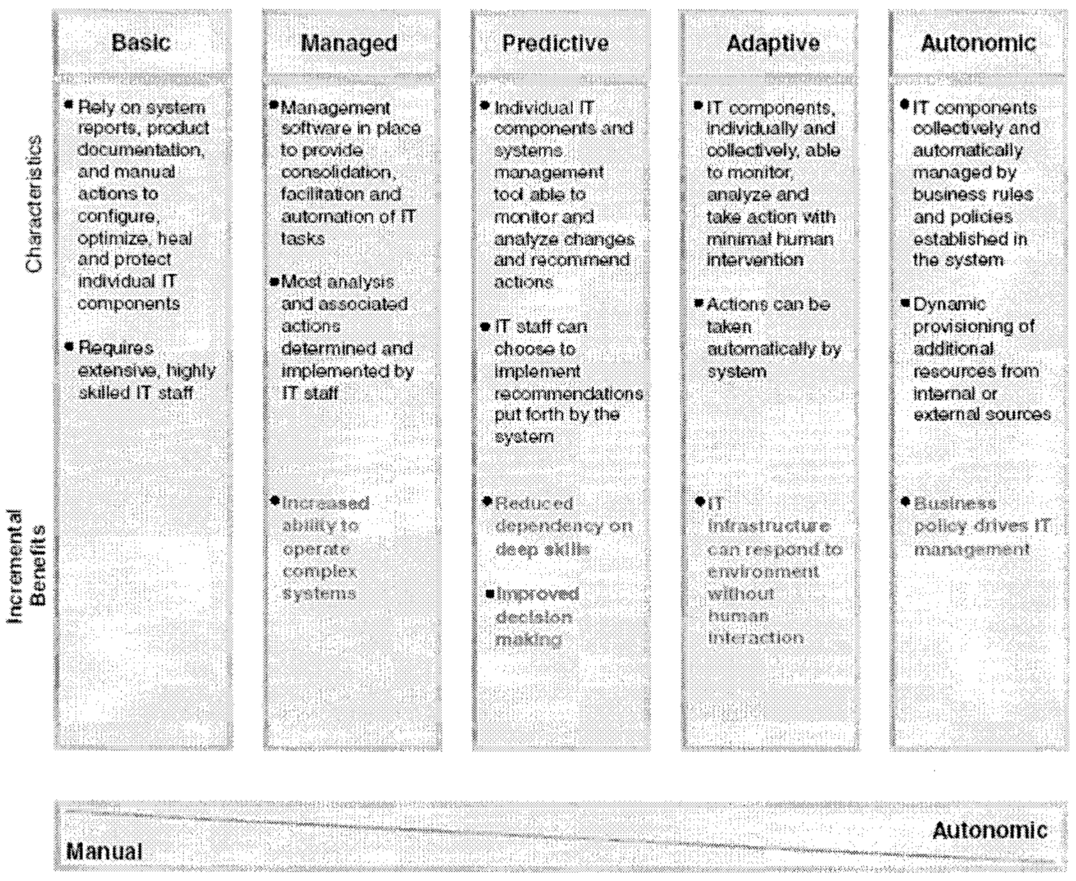
Transaction measurement represents the information based on the flow of transactions over an autonomic architecture. According to IBM, autonomic managers need:

"A *transaction measurement capability that spans system boundaries in order to understand how the resources of heterogeneous systems combine into a distributed transaction execution environment. By monitoring these measurements, the autonomic manager can analyze and plan to change resource allocations to optimize performance across these multiple systems according to policies, as well as determine potential bottlenecks in the system.*" [1]

4.3 Evolutionary Approach

To implement autonomic computing, IBM proposes an evolutionary approach in order to improve **current existing complex systems**. IBM blueprint states that "*the evolutionary process will be enabled by technology but it will ultimately implemented by each enterprise through adoption of technology and supporting processes*" [3].

This evolution toward more highly autonomic capabilities can be described with five autonomic maturity levels [3], illustrated in the following figure:



	Basic	Managed	Predictive	Adaptive	Autonomic
Characteristics	• Rely on system reports, product documentation, and manual actions to configure, optimize, heal and protect individual IT components • Requires extensive, highly skilled IT staff	• Management software in place to provide consolidation, facilitation and automation of IT tasks • Most analysis and associated actions determined and implemented by IT staff	• Individual IT components and systems management tool able to monitor and analyze changes and recommend actions • IT staff can choose to implement recommendations put forth by the system	• IT components, individually and collectively, able to monitor, analyze and take action with minimal human intervention • Actions can be taken automatically by system	• IT components collectively and automatically managed by business rules and policies established in the system • Dynamic provisioning of additional resources from internal or external sources
Incremental Benefits		• Increased ability to operate complex systems	• Reduced dependency on deep skills • Improved decision making	• IT infrastructure can respond to environment without human interaction	• Business policy drives IT management

Manual → Autonomic

Figure 16: Autonomic computing maturity index [3]

Basic level

The basic level is starting point of complex system environments, is the level at which many complex systems are today. Each complex system component is managed separately by administrators who set it up, monitor it and eventually replace it.

Managed level

The managed level is where systems management technologies can be used to collect information in a consistent manner from separate system components and display it on fewer consoles, reducing the time it for the administrator monitor the complex system.

Predictive level

In the predictive level, new technologies are introduced to provide mappings among the infrastructure components. The complex system begins to "recognize patterns, predict the optimal configuration and provide advice on what course of action the administrator should take" [3].

Adaptive level

The adaptive level is where the complex system can "automatically take the right actions based on the available information and the knowledge of what is happening in the environment" [3].

Autonomic level

In autonomic level "business policies and objectives control the complex system infrastructure operation. Users interact with autonomic technology to monitor business processes, alter objectives or both" [3].

4.4 Conclusion

As stated in the introduction, this chapter is based mainly on the architectural blueprints that IBM has set as a basis for their on-demand computing architecture. This survey has dedicated a whole chapter to the description of this architecture based on the fact that the IBM autonomic computing architecture is an open architecture for distributed interactive complex systems. Moreover, IBM architecture "*does not prescribe a particular management protocol or instrumentation technology since the architecture needs to work with the various computing technologies and standards that exist in the industry today and with new standards that will emerge in the future*" [1]. It is true that IBM endorses the WEB and web services, because on-demand systems run inherently on the web but this is just a complementary characteristics that even AS-TRM group might want to consider at a later time.

References

[1] IBM Corporation, "*An architectural blueprint for autonomic computing*", Autonomic Computing White Paper, April 2003, available at:
http://www-03.ibm.com/autonomic/pdfs/ACwpFinal.pdf

[2] IBM Corporation, "*An architectural blueprint for autonomic computing*", Autonomic Computing White Paper, October, 2004, available at:
http://www-03.ibm.com/autonomic/pdfs/ACBP2_2004-10-04.pdf

[3] IBM Corporation, "*An architectural blueprint for autonomic computing*", Autonomic Computing White Paper, June, 2005, available at:
http://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf

[4] Richard Murch, "*Autonomic Computing"*. Prentice Hall Professional Technical Reference, IBM Press, pp. 119-132, 2004.

[5] IBM Corporation, "*Automating problem determination. A first step towards self-healing computing systems*", Autonomic Computing White Paper, October 2003, available at:
http://www-03.ibm.com/autonomic/pdfs/Problem_Determination_WP_Final_100703.pdf

[6] G.Lafranchi, P. Della Peruta, A Perrone, D. Calvanese *"Toward a new landscape of systems management in an autonomic computing environment"*, IBM Systems Journal, Volume 42, No. 1, 2003.

[7] Jeffrey O. Kephart, David M. Chess, "*The Vision of Autonomic Computing,*" Published by IEEE Computer Society, Volume 36 (1), pp. 41-50, 2003, available at: http://www.research.ibm.com/autonomic/research/papers/AC_Vision_Computer_Jan_2003.pdf

[8] A. G. Ganeck, T.A. Corbi, "*The Dawning of the autonomic computing era",* IBM Systems Journal, Volume 42, No. 1, 2003.

[9] Paul Horn, "*Autonomic Computing: IBM's Perspective on the State of Information Technology",* IBM Corporation, October 15, 2001, available at: http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf

[10] John Strassner, Barry J. Menich, "*Philosophy and Methodology for Knowledge Discovery in Autonomic Computing Systems*", Proceedings of the 16th International Workshop on Database and Expert Systems Applications (DEXA'05), pp. 738-743, 2005.

[11] Steve R. White, James E. Hanson, Ian Whalley, David M. Chess, Jeffrey O. Kephart, "*An Architectural Approach to Autonomic Computing*", Proceedings of the International Conference on Autonomic Computing (ICAC04), pp. 2-9, 2004.

[12] G. Tziallas, B. Theodoulidis, "*Building Autonomic Computing Systems Based on Ontological Component Models and a Controller Synthesis Algorithm*" Proceedings of the 14th International Workshop on Databases and Expert Systems Applications (DEXA 03), pp. 674-681, 2003.

[13] Wang Fei, Li-Fan-Zhang, "*The Design of an Autonomic Computing Model and the Algorithm for Decision-Making*", Presented at 2005 IEEE GrC Program, April, 2005.

Chapter 5 Autonomic Computing Tools and Open Standards

Abstract

Today it is a known fact that technology can be used to manage technology. There are many companies that have already developed and deployed complex products based on this idea. The advances in software agents, the emergence of XML and a host of new standards promise incremental paths to the ultimate solution of autonomic computing [3]. This chapter provides a to-date global view of agents technology, and presents current existing open standards that have the capability to enable autonomic computing.

5.1 Intelligent Software Agents

The attention, while building computer systems, is always turned towards human models for inspiration. The era of artificial intelligence makes no exception. However, the understanding of human information processing is not an easy task either as:

"*attentional processes may be relatively data-driven, particularly when biologically imperatives are built into the human systems, or centrally controlled as a result of learning; the capacity to scan, store or transmit complex information is clearly limited to human organisms, however central and autonomic functions can be studied in order to explain the complexity of human reading; no integrated processing of information is possible without reference to stored information, which enables complex planning and* *prediction of the environment; this brings us to human complex memorial processes; insights can be gained as well by studying the development of human cognitive processes together with integration of data from multiple sources. There is a marvelous complexity involved in the integration of mental resources with cognitive and motor functions in order to yield such a smooth and integrated performance*". [4]

The field of artificial intelligence started over 50 years ago when Alan Turing described his prototype for intelligent machines. Since then, there has been notable evolution of speech recognition, speech generation, natural language understanding, knowledge representation, machine learning and machine reasoning through symbol processing techniques, neural networks, genetic algorithms and fuzzy logic.

Intelligent software agents' field has captured people imagination. Researchers believe now [1] that agent technology is one of the key and core technologies that could drive true autonomic computing functionality.

5.1.1 Definition

IBM researchers define intelligent agents as:

"*Software entities that carry out some set of operations on behalf of a user with some autonomy and employ either knowledge or representation of the user's goals and desires.*" [5]

5.1.2 Agent Characteristics

The table below summarizes agents and intelligent agents' characteristics.

Agent	Autonomous
	Able to communicate with other agents and user
	Monitors the state of its execution environment
Intelligent Agent	Able to use symbols and abstractions
	Able to exploit significant amounts of domain knowledge
	Capable of adaptive goal-oriented behavior
	Capable of operation in real-time
	Tolerant of error, unexpected, or wrong input
	Able to learn from the environment
	Able to communicate using natural language

Figure 17: Software agent characteristics [10]

5.1.3 Agent Classification

The following figure visualizes very clearly the software agent topology. We notice that smart (intelligent) agent is situated at the intersection of all categories.

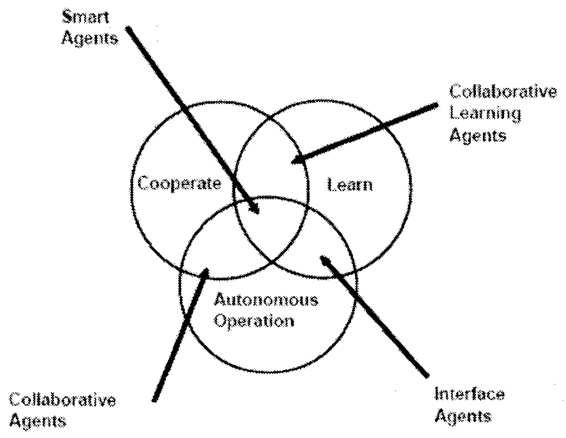


Figure 18: Agent topology [10]

The following paragraphs contain short introductions into interface agents, mobile agents, informational agent and heterogeneous agents.

Interface Agents

Interface agents are autonomous and are based on learning in order to achieve goals specified by users.

Mobile Agents

Mobile agents can be visualized as processes with computational power able to move over wide area networks (for example Internets or World Wide Web). They are capable to interact with foreign hosts and to return to the user after gathering information and performing their assigned tasks.

Information Agents

Traditionally, information agents are used for finding, analyzing and retrieving large quantity of information.

Heterogeneous Agent Systems

Heterogeneous agent systems are systems of agents with different agent architectures. From the documentation we surveyed, we found the following paragraph that highlights the key requirement of heterogeneous agent systems namely interoperability:

"*Because of the wide variety of application domains, it is unlikely that any agent architecture will be used exclusively across all domains; for each domain, the most* *appropriate agent architecture will be selected. Agents in these heterogeneous systems will communicate, cooperate, and interoperate with each other. A key requirement for this interoperation is the availability of an agent communications language that will allow different kinds of software agents to communicate with each other*". [10]

5.2 Multi-Agents

Agent based systems and architectures provide a firm foundation for design and development of autonomic systems. In the following subsections we describe some toolkits and multi-agent platforms currently available in industry or for educational use. The information presented in this subsection is a compilation of information we found in articles, user guides, reference guides and web sites we surveyed to date.

It is important to mention that there are three considerable agent standardization efforts, which are attempting to support interoperability between agents on different types of agent platform, namely, KQML, OMG's MASIF and FIPA.

Foundation for Intelligent Physical Agents (FIPA)

The Foundation for Intelligent Physical Agents (FIPA) is an international organization that was formed in order to support the industry of intelligent agents by "openly developing specifications supporting interoperability among agents and agent-based applications" [6]. It is possible by open collaboration among its member organizations namely companies and universities that conduct intensive research and development in the field of agents. FIPA makes the results of its activities public to its members and then makes available the results to the appropriate formal standards consortiums.

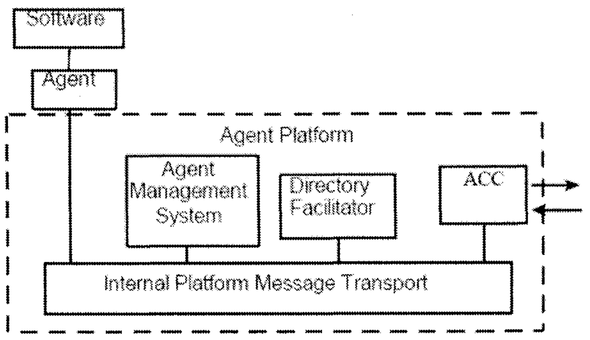


Figure 19: FIPA reference model of an agent platform [6]

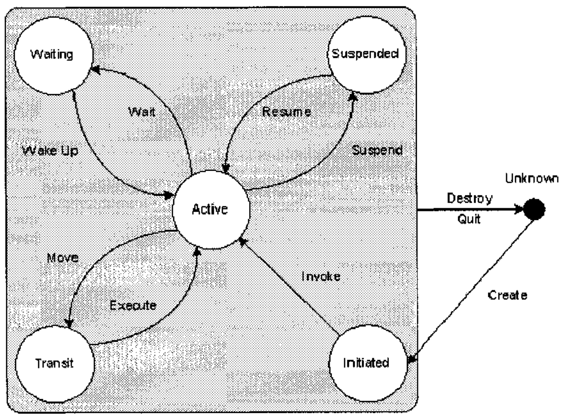


Figure 20: FIPA agent life cycle [6]

Knowledge Query Meta Language (KQML)

KQML or Knowledge Query Meta Language [7] was one of the first development to specify how to manage the "social interaction characteristic of agents using a protocol based on speech acts and is now also one of the most pervasive Agent Communication

Language (ACL)" [7]. However, to date, KQML is not a true standard because there is no consensus in the community on a single specification (or set of specifications). As a result, there are different versions of KQML. In consequence, different agent systems, speaking different dialects, may not inter-operate completely.

KQML sprung from a large effort, namely the DARPA Knowledge Sharing Effort that consisted of developing techniques and methodology for building large-scale, reusable and sharable, knowledge bases.

Mobile Agent System Interoperability Facility (MASIF)

The Object Management Group Mobile Agent System Interoperability Facility [8], MASIF, differs from both KQML and FIPA by how it defines agent's mobility from one location to another. "In contrast to MASIF, both KQML and FIPA emphasize agency and social interaction between multi-agents as the defining properties for software agents. MASIF does not support or standardize interoperability between non-mobile agents on different agent platforms. Further, MASIF restricts the interoperability of agents to agents developed on CORBA platforms whereas the focus of FIPA is to directly support the interoperability of agents deployed on agent frameworks which can support heterogeneous transports. OMG is exploring how to support other types of software agent than mobile agents. It has issued a Request for Information on agents. FIPA has supplied its specifications as input to this request. This is still work in progress at this time" [8].

5.2.1 CIAgent

The CIAgent framework developed by J. P. Bigus is an agent building framework developed for educational use [23].

5.2.2 Java Agent Template Lite (JatLite)

The Java Agent Template Lite, developed at Stanford Center for Design Research (CDR), Stanford University is a toolkit written in the Java language that enables users to quickly create software agents that communicate and interact with each other. JAT*Lite* agent's basic infrastructure model is depicted in the following figure.

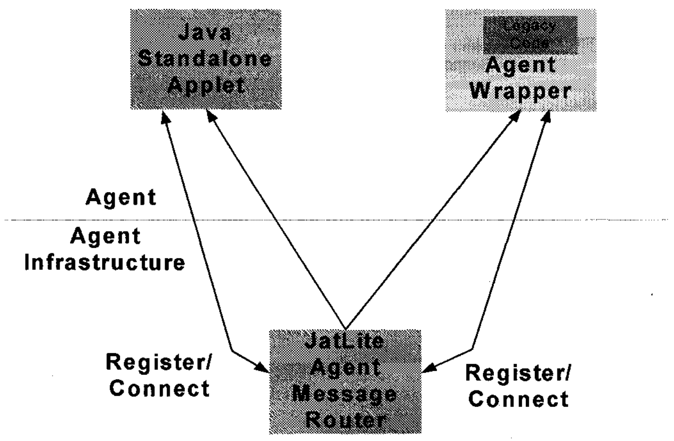


Figure 21: Java agent infrastructure with message routing [9]

JATLite agents can register with an **Agent Message Router** using a name and password, connect/disconnect from the Internet, send and receive messages, transfer files with FTP. Basically JATLite agents can exchange information with other agents on the various computers where they are executing. Agents in JATLite are based on Knowledge Query and Manipulation Language (KQML) protocol for exchanging information and knowledge. “Although JATLite does not, by itself, construct agents that seek information or automate human tasks, it does provide a robust substrate for building such intelligent agents leaving the developer free to use whatever theories and techniques are best suited for the targeted application or research” [9].

5.2.3 AgentBuilder

AgentBuilder is software toolkit that allows software developers to quickly develop intelligent software agents and agent-based applications. AgentBuilder has two major components. They are mentioned below as described in the toolkit’s documentation [10].

(1) The **Toolkit** that provides mechanisms for *managing* the development process, *analyzing* the domain, *designing* and *developing* networks agents, *defining behaviors* of agents, and *debugging and testing* agent software.

(2) The **Run-time System** that incorporates an agent engine that provides a framework for the execution of software.

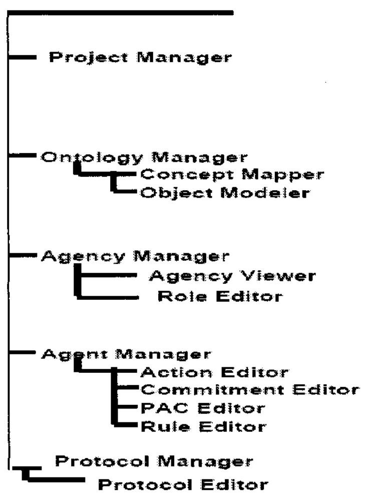


Figure 22: AgentBulider toolkit [10]

Agents constructed using AgentBuilder communicate using the Knowledge Query and Manipulation Language (KQML). In addition, AgentBuilder allows the developer to define new customized inter-agent communications commands.

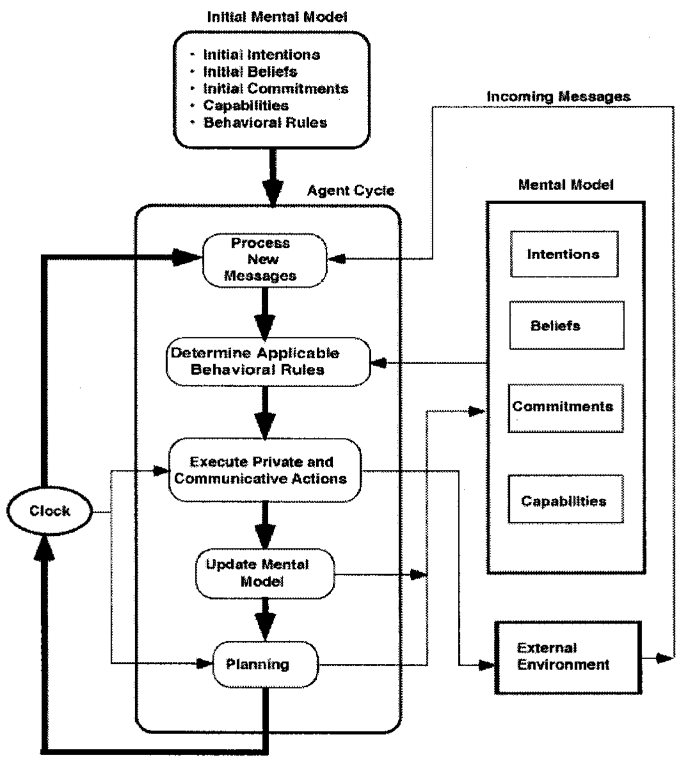


Figure 23: AgentBuilder intelligent agent architecture [10]

According to the above figure, the agent execution cycle consists of the following steps:

1. processing new messages
2. determining which rules are applicable to the current situation
3. executing the actions specified by these rules
4. updating the mental model in accordance with these rules
5. planning

5.2.4 ZEUS

Zeus is a multi-agent toolkit, developed by the British Telecommunications Laboratory. This toolkit offers a library of software components and tools that facilitate fast and friendly design, development, and deployment of multi-agents. In the article "*ZEUS: A Toolkit for Building Distributed Multi-Agent Systems*" [11] the authors Hyacinth S. Nwana, Divine T. Ndumu, Lyndon C. Lee & Jaron C. Collis illustrate in a context diagram some of the issues involved in knowledge level multi-agent collaboration.

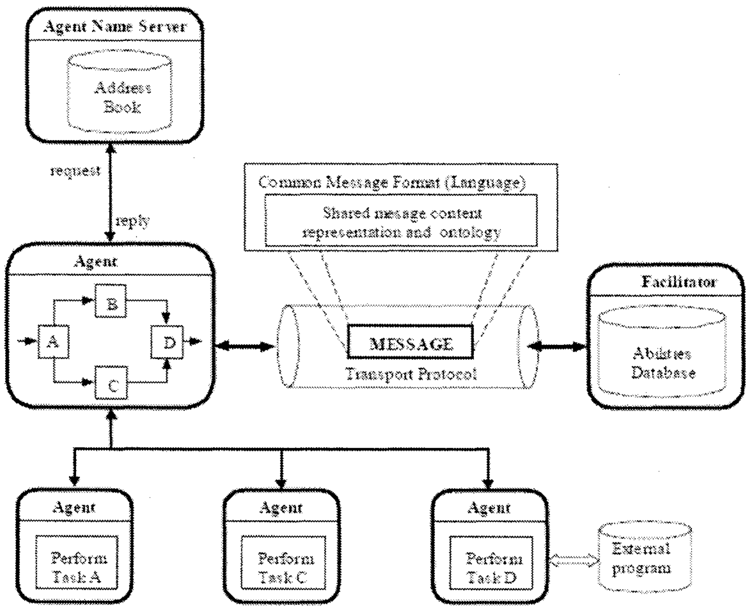


Figure 24: FIPA-based ZEUS reference model [11]

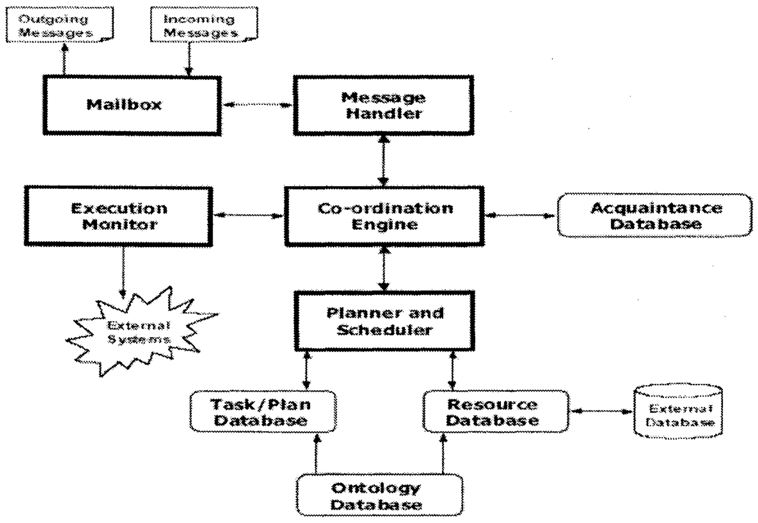


Figure 25: Generic ZEUS agent architecture [11]

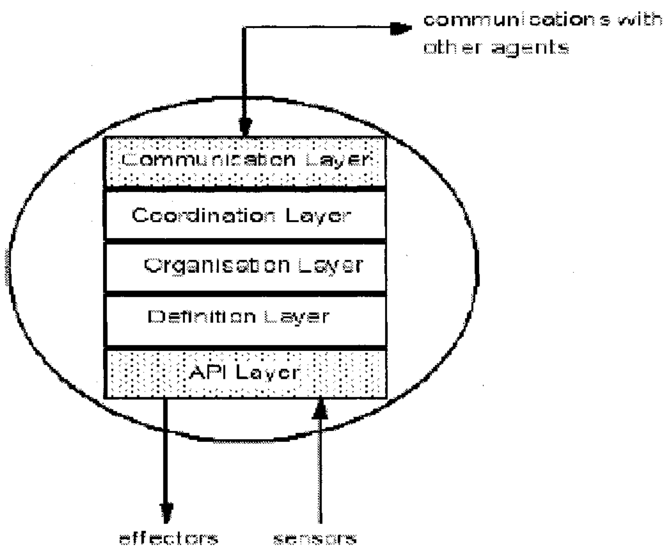


Figure 23: Generic ZEUS agent layered design [11]

Furthermore, the authors depict the ZEUS agent layered design as follows:

Application Programmer's Interface Layer

The application programmer's interface links the agent to the external programs that provide it with resources and/or implement its characteristics.

Definition Layer

At the definition layer, the agent is viewed as an autonomous reasoning entity with competencies, rationality model, resources, beliefs, preferences, etc.

Organization Layer

The organization layer enables agent's relationships with other agents, what other agents are aware of, what abilities it knows those other agents possess, what relationships its has with the other agents, etc.

Co-ordination Layer

At the co-ordination layer co-ordination and negotiation techniques are implemented such that the agent is visualized as a social entity.

Communication Layer

Communication layer is composed of protocols that implement inter-agent communication.

5.2.5 FIPA Open Source (FIPA-OS)

Foundations of Intelligent Physical Agents Open Source (FIPA OS) is an open agent framework developed by Nortel Networks. The framework is based on communication between multiple agents using an agent standard communication language. "A key focus of the platform is that it supports openness. This is naturally supported by the agent paradigm itself and by the design of the platform itself whose parts have loose coupling such that extensions and innovations to support agent communication can occur in several key areas. The openness is further emphasized in that the platform software is distributed and managed under an open-source licensing scheme" [12].

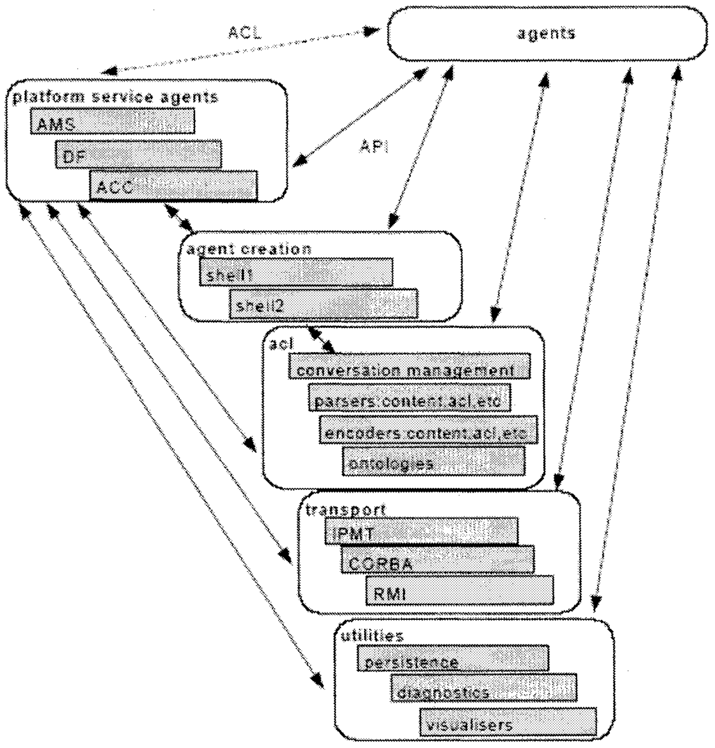


Figure 26: FIPA-OS reference architecture [12]

5.2.6 Java Agent Development Environment (JADE)

Java Agent Development Environment (JADE) was developed by TILAB in an effort to develop a distributed multi-agent system. "The intelligence, the initiative, the information, resources and control can be fully distributed on mobile terminals as well as on computers in the fixed network. The environment can evolve dynamically with agents appearing and disappearing in the system according to the needs and the requirements of the application environment. Communication between agents is completely symmetric with each agent being able to play both the initiator and the responder role" [13, 14]. Java Agent Development Environment (JADE) complies with Foundation for Intelligent Physical Agents (FIPA)

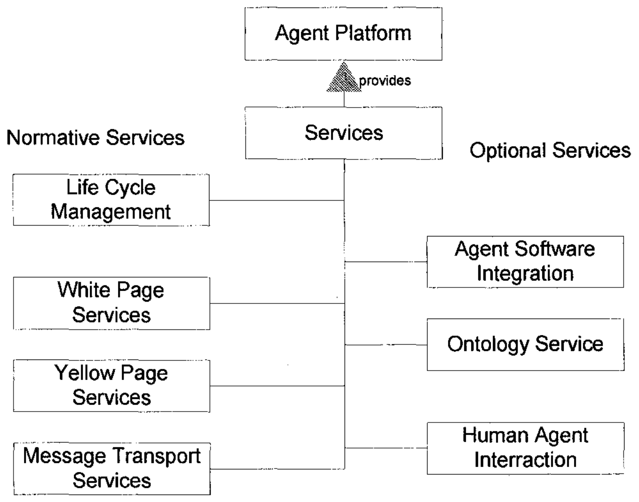


Figure 27: FIPA conceptual model of an agent platform [13, 14]

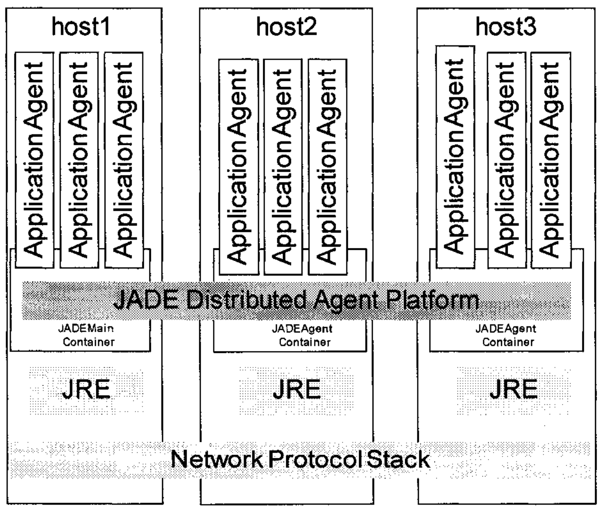


Figure 28: JADE distributed architecture [13, 14]

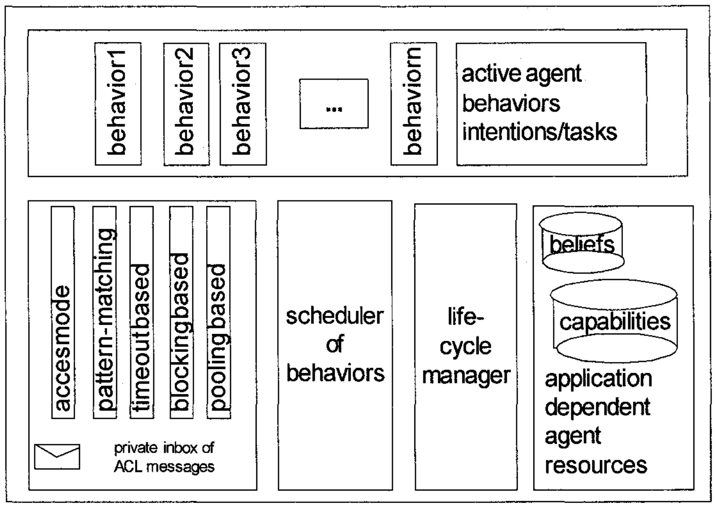


Figure 29: Generic JADE agent architecture [13, 14]

JADE internal architecture is being thoroughly detailed in the article "*JADE – A FIPA-compliant agent framework*" by Fabio Bellifemine, Agostino Poggi and Giovanni Rimassa [8].

Dario Bomino, Alessio Bosca and Fluvio Corno, in the paper "*An Agent Based Autonomic Semantic Platform*" [22], propose an autonomic agent-based platform developed using JADE. The platform is called the Distributed Open Semantic Elaboration (DOSE).

JADE was chosen as well for the development of a control system design by V. Gyurjyanm, D. Abbot, G. Heyes, E. Jastrzembski, C. Timmer, E. Wolin. In their article called "*FIPA agent based network distributed control system*" [3] they address the issues of interoperability and scalability of complex control systems by development of an open architecture control system.

5.2.7 JAS

JAS is a simulation toolkit designed to enable agent based simulation modeling. Agent based models in JAS are views of dynamic social systems made possible through object-oriented computer programs.

JAS is a clone of the Swarm library, an ABM framework initially developed by the Santa Fe Institute in order to create multi-agent simulations of complex adaptive systems.

"In the Swarm system, agents are organized in terms of the fundamental component referred to a 'swarm'. A swarm is a collection of agents with a schedule of events over those agents. The swarm represents an entire model: it contains the agents as well as the representation of time. Swarm supports hierarchical structures whereby in a nested fashion an agent can be composed of swarms of other agents. In some cases, the higher level agent's behavior is defined by the emergent phenomena of the agents inside its swarm. This multi-level model approach offered by Swarm is very powerful. Multiple swarms can be used to model agents that themselves build models of their world. In Swarm, agents can themselves own swarms, models that an agent builds for itself to understand its own world". [14]

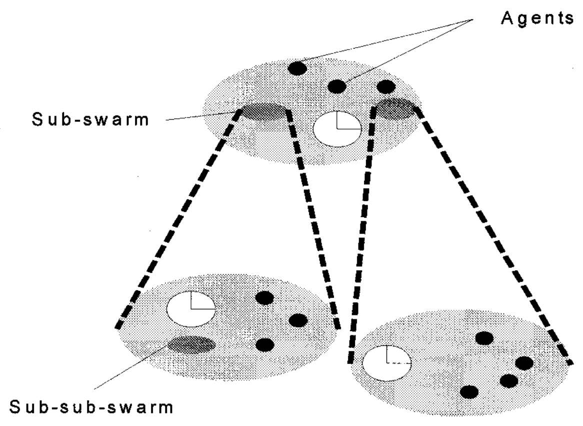


Figure 30: Swarm architecture as nested hierarchy of swarms [14]

5.2.8 Cognitive Agent Architecture (COUGAAR)

Cognitive Agent Architecture (COUGAAR) is a Java-based architecture developed to support the development of large-scale distributed agent systems. The architecture was developed as part of a research program funded by DARPA.

"Cougaar is a large-scale workflow engine built on component-based, distributed agent architecture. The agents communicate with one another by a built-in asynchronous message-passing protocol. Cougaar agents cooperate with one another to solve a particular problem, storing the shared solution in a distributed fashion across the agents. Cougaar agents are composed of related functional modules, which are expected to dynamically and continuously rework the solution as the problem parameters, constraints, or execution environment change." (BBN, Technologies, 2003)

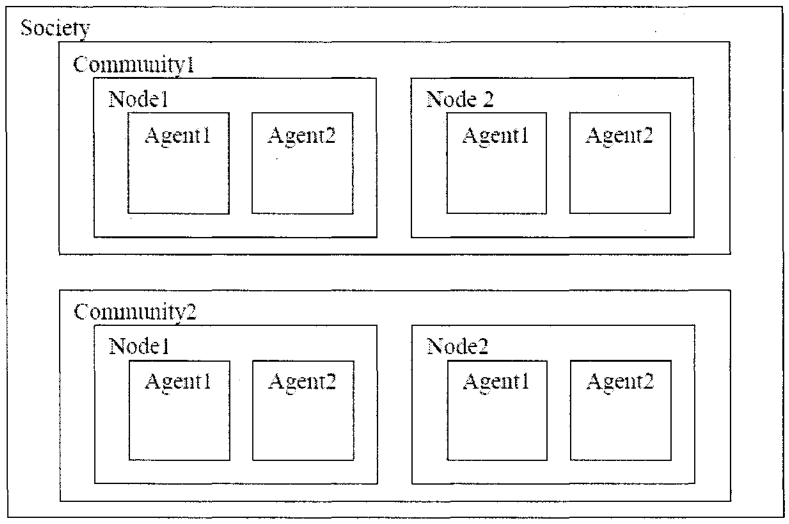


Figure 31: GOUGAR architectural view [16]

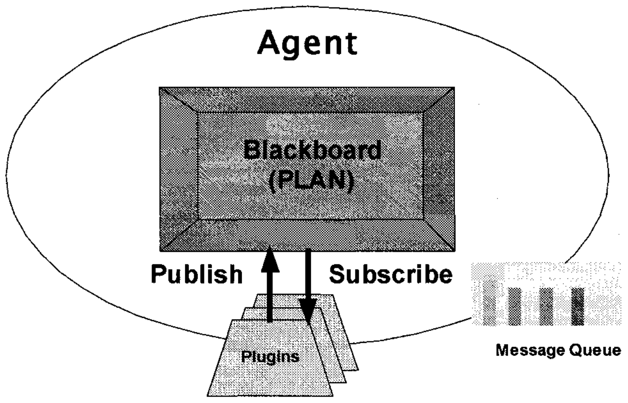


Figure 32: COUGAAR agent reference model [16]

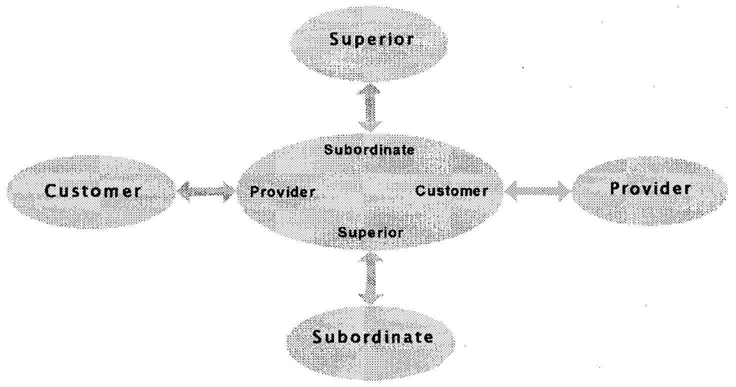


Figure 33: Agent multi-role capacity [16]

Denis Gracanin, Shawn A. Bohner, Michael Hinchey propose COUGAAR as a framework for the agent-based, model-driven architecture for building autonomic applications in their article "*Towards a Model-Driven Architecture for Autonomic Systems*" [17].

5.2.9 Agent Building and Learning Environment (ABLE)

Agent Building and Learning Environment is the autonomic IBM agent technology. It provides a Java agent framework, a library of intelligent components (JavaBeans) a set of development and test tools and an agent platform.

J. P. Bigus, D. A. Schlosnagle, J. R. Pilgrom, W. N. Mills , Y. Diao describe in detail ABLE's architecture in their paper called "*ABLE: A toolkit for building multiagent autonomic systems*" [18]. Their approach to build autonomic systems is to combine autonomous intelligent agents in a structured way. With this respect, they further mention that ABLE's architecture includes components with the following characteristics:

ABLE Agent	Has short-term, long-term and associative memory functions
	Is proactive with reflexive, reactive, goal-oriented behavior
	Has reasoning, planning and learning new behaviors
	Has emotional behavior that associates feelings with internal states
	Communicates with the environment through sensors and effectors

Figure 34: ABLE agent characteristics [18]

In the paper, the ABLE agent model is depicted with the following figure:

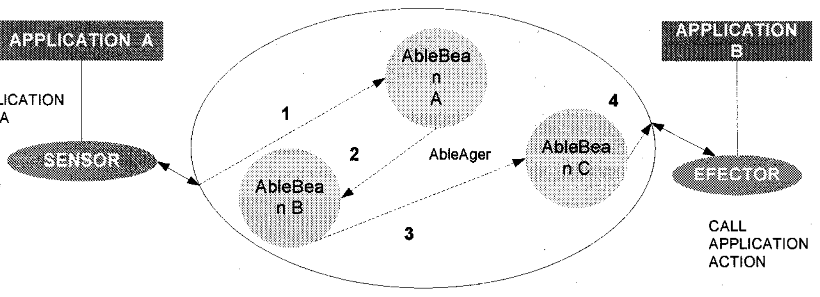


Figure 35: ABLE agent model [18]

ABLE distributed agent platform corresponds to FIPA reference architecture and is illustrated below:

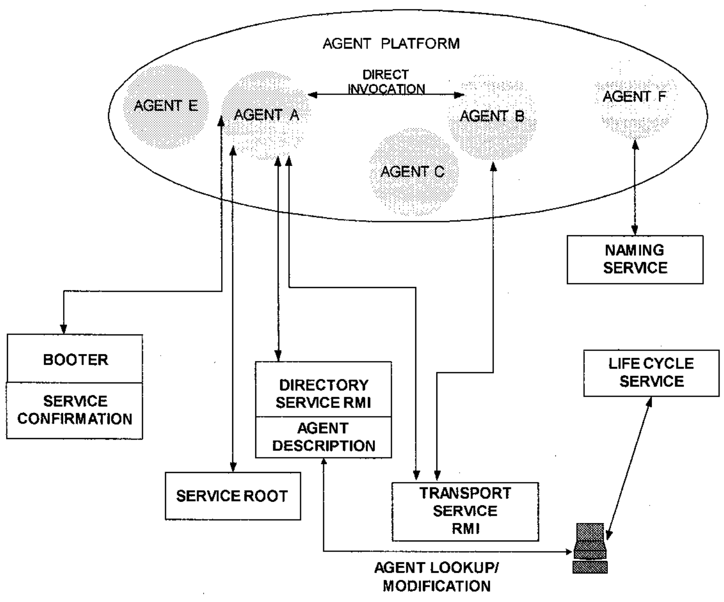


Figure 36: ABLE distributed agent platform [18]

In addition, the paper shows as well the architecture of ABLE's autonomic agent. As stated by the authors it is based on prior work in this area, in particular the work of Solomon and Minski.

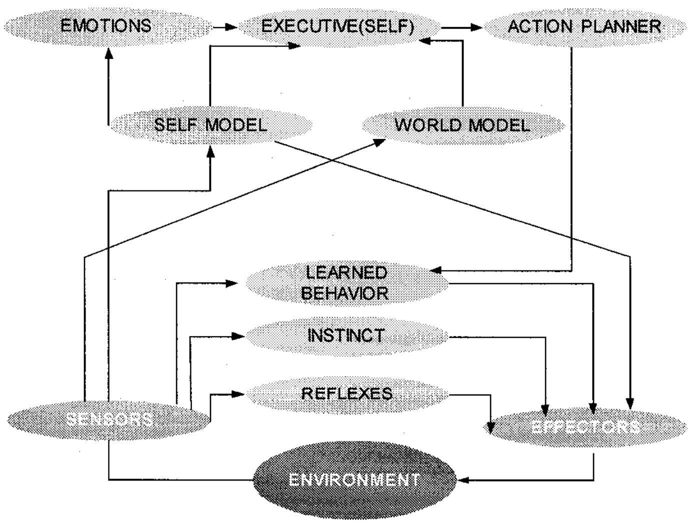


Figure 37: ABLE agent architecture [18]

5.2.10 Unity

Unity is a multi agent system approach to autonomic computing. It was developed at IBM T.J. Watson Research Center. Its aim is to develop an autonomic distributed computing system based on interactions among autonomous agents, which are called **autonomic elements**.

The Unity system components are implemented as autonomic elements. In fact, every Unity component is an autonomic element. Jeffrey O. Kephart, from IBM TJ Watson Research Center, presents Unity in his article *"A Multi-Agent System Approach to Autonomic Computing*" [19]. The following paragraphs abstract out Unity's elements described in detail in the above-mentioned article.

Computing Resources Elements

Computing resource elements can be databases, storage systems, servers, etc.

Application Manager Element

Application manager element is responsible for internal management of the environment, for obtaining the resources that the environment needs and for communication with other elements depending on the needs of the environment.

Resource Arbiter Element

The resource arbiter element calculates an optimum resource based on estimated received from each application environment.

Server Element

Server element is responsible for publishing the server's address and capabilities such that possible users of the server can use them.

OSContainer Element

OSContainer receives requests from elements to start up services or autonomic elements.

Registry Element

Registry element enables elements to locate other elements with which they need to communicate. It corresponds to registries in other multi-agent systems.

Policy Repository Element

Policy Repository element provides human-computer interfaces that allow administrators to enter high-level policies that control the operation of the system.

Sentinel Element

Sentinel element monitors the functioning of an element on behalf of another element. If the monitored element becomes unresponsive, the sentinel sends a notification to the element that requested the monitoring about the situation.

Jeffrey O. Kephart sates the following with respect to elements behavior and relationships in Unity:

"*Each autonomic element is responsible for its own internal autonomic behavior, namely, managing the resources that it controls, and managing its own internal operations, including self-configuration, self-optimization, self-protection, and self-healing. Each element is also responsible for forming and managing relationships that it enters into in order to accomplish its goals, that is, the external autonomic behavior that enables the system as a whole to be self-managing.*" [19]

An experimenting technique in Unity called "goal-driven self-assembly" is described in another article called "*Unity: Experiences with a Prototype Autonomic Computing System*" by David M Chess, Alla Segal, Ian Whalley and Steve R. White [2].

5.2.11 NASA's Lights-Out Ground Operations System (LOGOS)

Lights-Out Ground Operations System (LOGOS) [20] is multi-agent NASA autonomic system. It was developed based on agent technology, to illustrate autonomous ground and space operations with low costs and to provide new technology for future missions with long time periods between contacts with the ground. The following subsections, that describe LOGOS modeling and development and its autonomic characteristics, are based on the article "*Some Autonomic Properties of Two Legacy Multi-Agent NASA Systems – LOGOS and ACT*" [20] written by Walt Truszowski, James Rash, Christopher Rouff and Micke Hinchey.

5.2.11.1 Modeling and Development

LOGOS has multi-agent autonomous architecture depicted in the following figure:

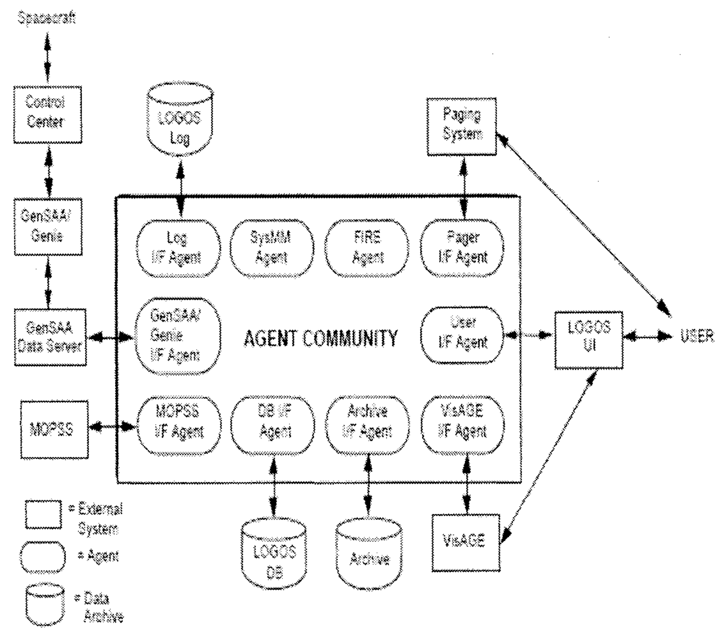


Figure 38: Logos architecture [20]

According to the architecture above, LOGOS agent community contains 10 agents as follows:

1. System Monitoring and Management Agent, which has knowledge about all agents in the community and informs agents requesting services about other agents addresses.
2. Fault Isolation and Resolution Expert (FIRE), who resolves satellite problems.
3. User Interface agent provides human-computer interface between agent community and humans that may need to interface with LOGOS.
4. VisAGE Interface agent which interfaces with VisAGE2000 data visualization module that displays spacecraft telemetry and agent log information.
5. Pager Interface agent interfaces with external analyst's pager system.

6. Database Interface agent, (7) Archive Interface agent and (8) LOG agent store short term, long term and logging data for debugging respectively.
9. GenSAA/Genie Interface agent interface ground station software in order to handle communication with the spacecraft.
10. Mission Operation Planning and Scheduling Interface agent interfaced with ground station planning and scheduling software.

5.2.11.2 Autonomic Features

Self-Configuration

LOGOS self configures when a spacecraft pass is close to happen. GIFA wakes-up the FIRE, UIFA, and VIFA agents if needed for system configuration. If for example an anomaly is detected, FIRE agent is woken up.

Self-Optimization

Self-optimization feature is envisaged through learning. Following are some examples:

- FIRE agent learns new anomalies and their fixes by adding them in its knowledge base.
- VisAGE agent learns information about session when an analyst logs into the system. Data will be available to the analyst at the next log in.
- Pager agent learns analysts' availability times and decides whom to page at a particular time.

Self-Healing

LOGOS self-healing properties are supported by the FIRE agent actions: examines anomalies, decides their resolutions, contacts external system (analyst) if new problems are encountered and learns the new fixes provided by the analyst for future use.

Self-protecting

Self-protection is performed through User Interface agent that identifies analyst at login and by FIRE agent that validates commands sent by the analyst such that they do not damage the spacecraft.

5.2.12 NASA's Agent Concept Testbed (ACT)

Similar to LOGOS, Agent Concept Testbed (ACT) [20] is a multi-agent NASA autonomic system. The autonomic properties of the system were designed and developed such that NASA missions are to be conducted without constant human intervention. The self-configuring, self-healing, self-optimizing and self-protecting properties of both LOGOS and ACT make them autonomic systems. The following subsections, that describe ACT's modeling and development and its autonomic characteristics, are also based on the article "*Some Autonomic Properties of Two Legacy Multi-Agent NASA Systems – LOGOS and ACT*" [20] written by Walt Truszowski, James Rash, Christopher Rouff and Micke Hinchey.

5.2.12.1 Modeling and Development

ACT has an evolving-system component-based architecture, in which a component can be easily replaced with a more advanced one. There are two types of agents in ACT architecture:

1. Reactive agent (simple agent) - receives inputs from environment and in turn reacts accordingly.
2. Robust agent (complex, intelligent agent) - can reason in a deliberative, reflexive or social way.

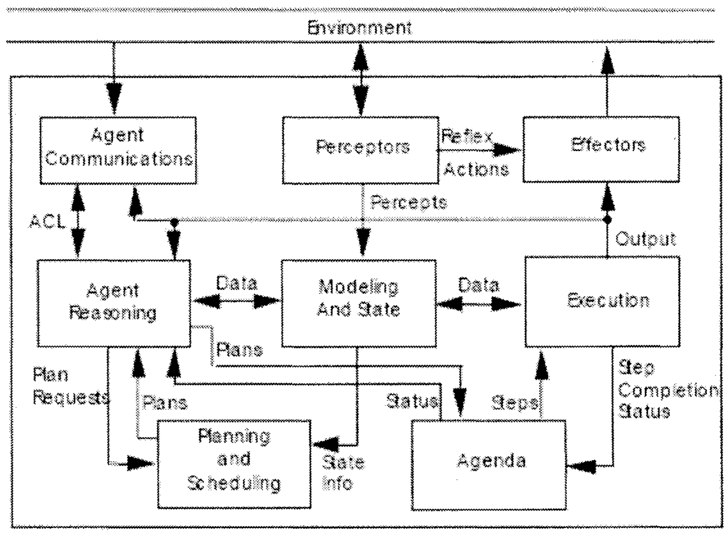


Figure 39: ACT architecture [20]

According to the above figure the ACT Architecture has the following components:

1. Modeler component responsible for the domain model of agents and environment.
2. Reasoner component makes decisions and sets goals for the agents based on data in its knowledge and based on data from modeler component
3. Planner/Scheduler component is responsible for agent planning and scheduling. Its input is a goal and its output is a plan as a directed graph of steps, composed of preconditions, actions and post-conditions.
4. Agenda and Executive components execute plans developed by Scheduler/Planner.
5. Agent communication component sends and receives messages from/to other agents.

6. Preceptor component monitors the environment on behalf of agents.
7. Effector component sends output to the environment.

The ACT message format used between agents is based on Foundations of Intelligent Physical Agents (FIPA) standard. The transmitting of messages occurs through Workplace, a proprietary agent messaging software developed by NASA.

5.2.12.2 ACT Autonomic Properties

Self-Configuring

In case of a problem, the Contact Manager changes the current satellite schedule and the problem is addressed. The spacecraft environment that is currently planned for the contact is reconfigured subsequently.

Self-Optimizing

ACT is self-optimizing when a proxy agent determines a problem and consequently a re-planing/rescheduling activity is triggered. This optimizes the entire ACT behavior.

Self-Healing

The self-healing property is ensured by the problem diagnosis/corrective action cycle, which may involve access to the human component of ACT and consequently learning.

Self-Protecting

ACT is self-protecting by continuously monitoring the spacecraft and modifying its operations when constraints are violated. It also contains a thorough validation of system commands received from administrators when self-healing thus ensuring that they will not have adverse impact on the spacecraft.

5.2.13 NASA's Autonomic and Swarm-based Systems

This section is based on the *article "Formal Methods for Autonomic and Swarm-based Systems*" written by Christopher Rouff, Amy Vanderbilt, Mike Hinchey, Walt Truszkowski and James Rash [21]. The article discusses some on-going work to develop a formal method for verifying (NASA) swarm and autonomic systems. According to the authors: *"For swarm exploration, individual autonomy is not crucial, but the mission cannot succeed unless each team has all the autonomic properties of being. Because of this, current methods must be modified or new methods must be created to properly take into account the learning, intelligence and emergent behavior of such systems*" [21]. Furthermore the authors state that to date, the following two formal approaches have been used to analyze emergent behavior:

1. Weighted Synchronous Calculus Communicating Systems (WSCCS) that was used by Tofts to model social insects, and to analyze the non-linear aspects of social insects.
2. X-Machines that have been used to model cell biology.

As noted by the authors these approaches do not predict emerging behavior but model the emergent behavior after the fact. Consequently, an effective formal method must:

- predict the emergent behavior of 1000 agents as a swarm as well as the behavior of individual agents,
- track the goals of the mission as they change, and
- modify the model of the universe based on new data that flows in.

Currently the merging of the two above formal methods and others is being performed within NASA.

5.3 Standardization

Definition

Open standards are defined as interfaces of formats that are openly documented and have been accepted in the industry through either formal or de-facto processes, and are freely available for usage by the industry.

According to IBM documentation [1], industry standards are necessary to support autonomic computing such that the following are ensured:

- Uniform approach to instrumentation and data collection. This will enable the intersystem exchange of instrumentation and control information and create the basis for collaboration and autonomic behavior between heterogeneous systems.
- Dynamic configuration.
- Operation.

5.3.1 Open Standards

Following, are some proposed standards of interest to autonomic computing together with a small description The standards descriptions are based on information found on standards web pages, and on the survey provided by Richard Munch in his book called "*Autonomic Computing*" [1].

1. **BlueFin**

   BlueFin specification is a proposed standard for data collection.

2. **Distributed Management Task Force (DMTF)**

   Distributed Management Task Force (DMTF) is the industry organization that is conducting the development, adoption, and unification of management standards for desktop, enterprise and Internet environments.

3. **Common Information Model (CIM)**

   Common Information Model (CIM) standard is an object-oriented information model that provides a conceptual view of physical and logical components of a system.

4. **Policy Core Information Model (RFC3060)**

   Policy Core Information Model (RFC3060) is the standard that presents the object-oriented information model for viewing policy information developed in collaboration in the IETF Policy Framework WG and as extensions to CIM activity in the DMTF.

5. **Open Group Application Response Measurement (ARM)**

   Open Group Application Response Measurement (ARM) is a standard for application instrumentation.

6. **Web Service Level Agreement (WSLA)**

   Web Service Level Agreement (WSLA) is a language to express SLA contracts, to support guaranteed performance, and to handle complex dynamic fluctuations in service demand.

7. **Open Grid Service Architecture (OGSA)**

   Open Grid Service Architecture (OGSA) defines standard mechanisms for creating, naming and discovering services and specifies various protocols to support accessing services. Basically is a framework for distributed computing based on Web protocols. It works even in a simply distributed system resources within an enterprise.

8. **Organization for the Advanced of Structured Information Standards (OASIS)**

   Organization for the Advanced of Structured Information Standards **(OASIS)** generates standards for security, Web Services, XML conformance, business transactions, electronic publishing and interoperability [1].

9. **Physical Agents (FIPA)**

   The Foundation for Intelligent Physical Agents is an international standards consortium working for interoperability between agents and agent platforms and has specifications for agents, agent management services and agent communication languages.

10. **Java Community Process**

    Java Community Process is an open organization for Java development.

11. **Java Management Extensions (JSR3, JMX)**

    Java Management Extensions (JSR3, JMX) specification provide a standard management architecture, API's and services for building distributed, dynamic and modular solutions to manage Java-enabled resources.

12. **Java Agent Services (JSR87)**

    Java Agent Services are a set of objects and service interfaces to support deployment and operation of autonomous communicative agents. It is being developed under SUN Java Agent Services (JSR87)

13. Simple Network Management Protocol (SNMP)

Simple Network Management Protocol (SNMP) enables network administrators to manage network performance, find and solve network problems and plan for network growth.

5.3.2 Mapping Open Standards against Core Autonomic Capabilities

The following table based on Richard Munch work in his book "*Autonomic Computing*" [1] gives insights on the mapping of open standards to autonomic core capabilities presented in chapter 4, section 4.2.

Table : Open Standards Vs Autonomic Capabilities
Solution Install
No standard existing; to be developed
Common System Administration
Portlet Specification (JSR 168)
Problem Determination
Common Information Model (CIM) Simple Network Management Protocol (SNMP) Web-Services Distributed Management (WS-DM) Java Management Extensions (JSR3,JMX) Logging API Specification (JSR47) BlueFin Open Grid Service Architectures (OGSA) Open Grid Services Infrastructure (OGSI) Web Services Common Resource Model (WS-CRM) Application Response Measurement (ARM)
Autonomic Monitoring
Common Information Model (CIM) Simple Network Management Protocol (SNMP) Java Agent Services (JSR87) BlueFin Open Grid Services Architecture (OGSA) Open Grid Services Infrastructure (OGSI) Application Response Measurement(ARM)
Policy Based Management
Policy Core Information Model

Web Services Security Java Management Extension (Jsr3, JMX) Open Grid Services Infrastructure (OGSI)
Complex Analysis
Java Agent Services Web Services Common Resource Model (WS-CRM) Open Grid Services Infrastructure Application Response Measurement (ARM)
Transaction Measurement
Common Information Model (CIM) Web Services Distributed Management (WS-DM) Java Management Extensions (JSR3, JMX) Logging AP Specification (JSR47) Java Agent Services (JSR87) Open Grid Services Architecture (OGSA) Open Grid Services Infrastructure (OGSI) Web Services Common Information Model (WS-CRM) Application Response Measurement (ARM)

Figure 40: Mapping open standards to autonomic computing capabilities [1]

5.4 Conclusion

An introduction into current agent technologies was presented in the first part of this chapter. IBM and NASA research and projects demonstrate that autonomic systems development emergent from agent technology is possible.

This chapter has also presented current standards with enabling power for autonomic computing. However, for an ultimate solution in a multi-vendor infrastructure, vendors must agree on a standards-based approach for autonomic systems. Consequently, there is a need for new standards that will define new mechanisms for inter-operating heterogeneous systems.

References

[1] Richard Murch, "*Autonomic Computing*", Prentice Hall Professional Technical Reference, IBM Press, pp. 119-132, 2004.

[2] David M Chess, Alla Segal, Ian Whalley, Steve R. White, "*Unity: Experiences with a Prototype Autonomic Computing System*", Proceedings of the International Conference on Autonomic Computing (ICAC'04), pp. 140-147, 2004.

[3] V. Gyurjyanm, D Abbot, G. Heyes, E Jastrzembski, C. Timmer, E. Wolin, "*FIPA agent based network distributed control system*", Computing in High Energy and Nuclear Physiscs, La Jolla California, 24-28 March, 2003.

[4] J. Richard Jennings, Michael G.H Coles, "*Handbook of Cognitive Psychophysiology Central and Autonomic Nervous System Approaches*", John Wiley & Sons, pp. 30, 1991.

[5] Gilbert and al, "*Intelligent Agent Strategy*", IBM White Paper, 1995, additional information at:

http://www.research.ibm.com/iagents/home.html

[6] Foundation for Intelligent Physical Agents, "*FIPA Agent Management Specification*", available at:

http://www.fipa.org/specs/fipa00023/XC00023H.html

[7] Finin et al., "*An Overview of KQML: A Knowledge Query and Manipulation Language*", 1997, available at:

http://www.cs.umbc.edu/kqml/papers

[8] Fabio Bellifemine, Agostino Poggi, Giovanni Rimassa, "*JADE – A FIPA-compliant agent framework*", Proceedings of the 4th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents, pp. 97-108, 1999.

[9] Heecheol Jeon, Charles Petrie, Mark R. Cutkosky, "*JATLite: A Java Agent Infrastructure with Message Routing*", available at:

http://www-cdr.stanford.edu/ProcessLink/papers/jat/jat.html

http://www-cdr.stanford.edu/ProcessLink/papers/JATL.html#WhatIsJATLite

[10] "*AgentBuilder: Reference Manual and Agent Builder User Manual", 16 June, 2004* available at:

http://www.agentbuilder.com

[11] Hyacinth S. Nwana, Divine T. Ndumu, Lyndon C. Lee, Jaron C. Collis,*"ZEUS: A Toolkit for Building Distributed Multi-Agent Systems",* available at:

http://www.agent.ai/doc/upload/200302/nwan99_1.pdf

[12] Stefan Poslad, Phil Buckle, Rob Hadingham, "*The FIPA-OS agent platform: Open Source for Open Standards",* available at:

http://fipa-os.sourceforge.net/docs/papers/FIPAOS.pdf

[13] Fabio Bellifemine, *"JADE Framework what it is and what it is next"*, Invited speech at ETAPS 2001 Workshop on Models and Methods of Analysis for Agent Based Systems (MMAABS), Genova, April, 2001 available at:

http://jade.cselt.it/

[14] F. Bellifemine, G. Caire, A. Poggi, G. Rimassa, "*JADE a White Paper*", Sept. 2003, available at:

http://jade.cselt.it/

[15] Hala Al-Bakour, Sheri Markose, "*Swarm tutorial*", available at:

http://www.essex.ac.uk/ccfea/swarm/SwarmTutorial/web/swarm_tutorial.htm

[16] BBN Technologies Document, "*Cougaar Architecture Document*", 23 December, 2004 available at:

http://cougaar.org/docman/view.php/17/170/CAD_11_4.pdf

[17] Denis Gracanin, Shawn A. Bohner, Michael Hinchey, "*Towards a Model-Driven Architecture for Autonomic Systems*", 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'04), p. 500, 2004.

[18] J. P. Bigus, D.A. Schlosnagle, J. A. Pilgrim, W.N. Mills, Y. Diao, *"ABLE: A toolkit for building multi-agent autonomic systems*", IBM Systems Journal, Volume 41, No. 3, 2002.

[19] Jeffrey O. Kephart, *"A Multi-Agent System Approach to Autonomic Computing*", Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04), Volume 1, pp. 464-471, 2004.

[20] Walt Truszkowski (NASA), James Rash (NASA), Christopher Rouff (SAIC), Mike Hinchey (NASA), "*Some Autonomic Properties of Two Legacy Multi-Agent NASA Systems – LOGOS and ACT*", presented at 11th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS'04), pp. 490, 2004.

[21] Christopher Rouff, Amy Vanderbilt (SAIC), Mike Hinchey, Walt Truszkowski, James Rash (NASA), "*Formal Methods for Autonomic and Swarm-based Systems*", available at:

http://isd.gsfc.nasa.gov/Papers/DOC/RashISOLA.pdf

[22] Dario Bomino, Alessio Bosca, Fluvio Corno, "*An Agent Based Autonomic Semantic Platform*", Proceedings of the International Conference on Autonomic Computing, (ICAC'04), pp. 189-196, 17-18 May, 2004.

[23] J.P. Bigus, J.Bigus, "*Constructing intelligent agents using Java*", New York, Wiley, 2004.

Chapter 6 Case Studies from Industry

Abstract

To date, there are many visions on how to deal with complexity. They sprung out of years of research and development and they are all trying to solve the paradox of creating simple to manage, yet complex systems.

6.1 An Overview of this Chapter

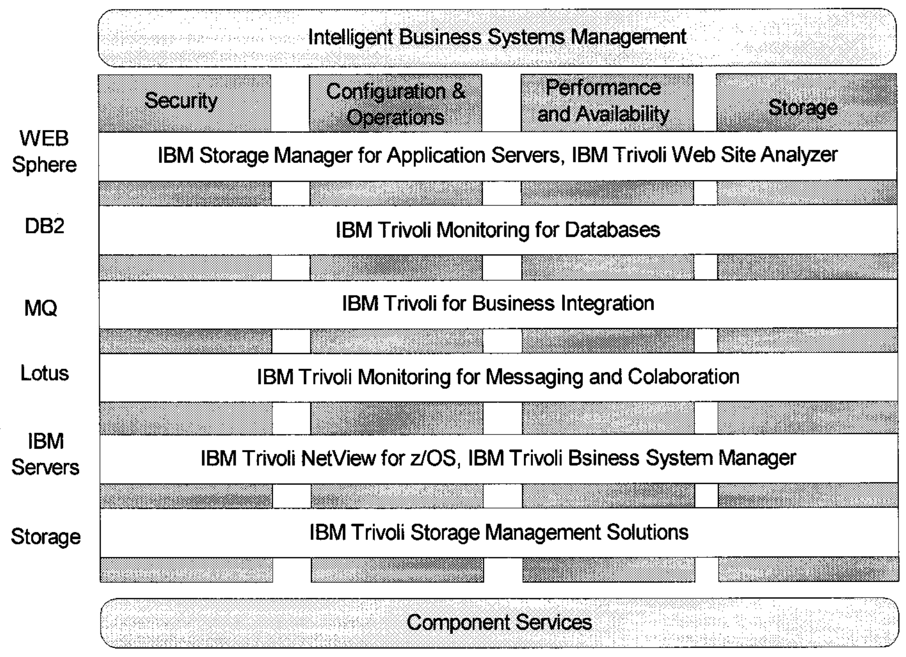
Few case studies from industry were already discussed in chapter 5, for example NASA's autonomic agent based systems. Subsections 2 and 3 will present IBM's autonomic products Trivoli and DB2 respectively including details about their architecture, modeling and development. In subsection 4, Sun's vision for the next generation data center will be discussed in greater detail as well as its modeling and development. Subsection 5 will discuss Hewlett Packard's approach to autonomic computing called The Adaptive Enterprise. Microsoft's Dynamic Systems Initiative will be depicted in subsection 6, together with its unifying software architecture centered on its "System Definition Model" (SDM) Business. Subsection 7 contains an overview of on going university researches on autonomic computing modeling and development.

6.2 IBM Trivoli Management Suite

Tivoli Management software from IBM provides a jump-start toward fulfilling the ultimate goal of a fully autonomic system. Today, it continues to participate in developing the ultimate solution. Many Tivoli management products have some level of automation that can help achieve the benefits of an autonomic computing environment.

6.2.1 Modeling and Development

The following figure depicts the coverage of IBM Trivoli Management products across the IBM portfolio of products and services:



Intelligent Business Systems Management
Security
Configuration & Operations
Performance and Availability
Storage
WEB Sphere
IBM Storage Manager for Application Servers, IBM Trivoli Web Site Analyzer
DB2
IBM Trivoli Monitoring for Databases
MQ
IBM Trivoli for Business Integration
Lotus
IBM Trivoli Monitoring for Messaging and Colaboration
IBM Servers
IBM Trivoli NetView for z/OS, IBM Trivoli Bsiness System Manager
Storage
IBM Trivoli Storage Management Solutions
Component Services

Figure 41: IBM Trivoli Management across IBM overall architecture [4]

Framework

The foundation of the Trivoli Enterprise architecture is distributed object-oriented software called *Trivoli Framework.* Most of the applications of the Trivoli Enterprise are leveraged by services included in this framework. In addition to the framework, Trivoli

Enterprise provides management products for deployment, availability, operations and security management.

6.2.2 Autonomic Computing Characteristics

Tivoli products support and incorporate the core autonomic behaviors of self-configuring, self-healing, self-optimizing and self-protecting. The following illustration contains some examples of these products.

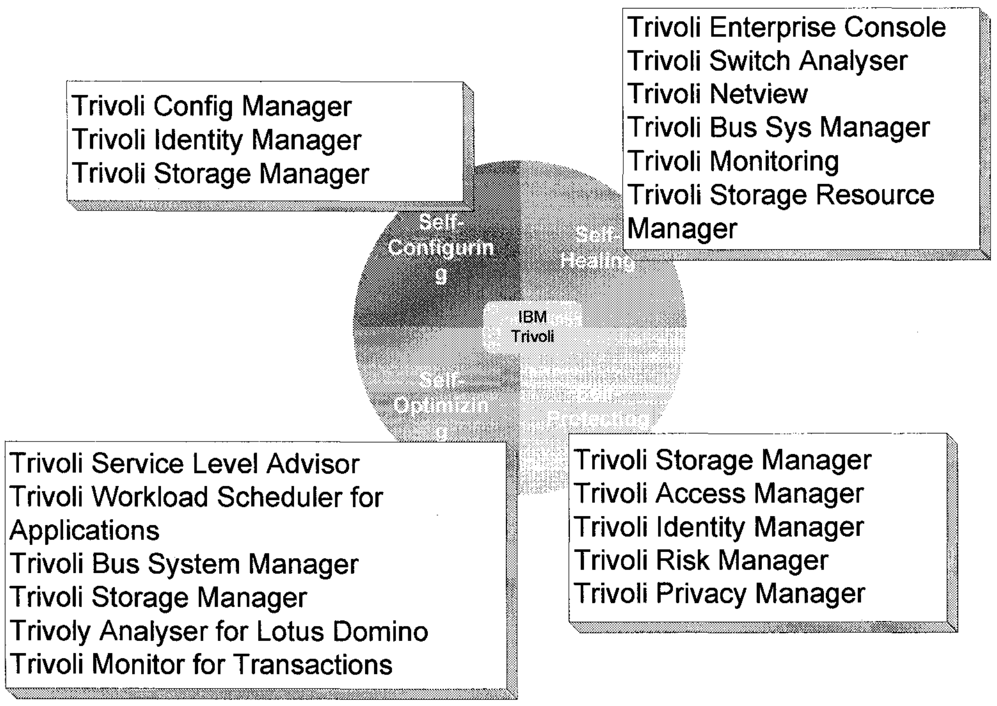


Figure 42: Trivoli autonomic software products [4]

Following are examples of Trivoli products and their autonomic characteristics. The information provided in this subsection is based on the work of R. Munch in his book "*Autonomic Computing*" [4].

Self-Configuring

The Configuration Manager can notice when software on a machine is not synchronized with the reference model and it creates a customized deployment plan for each machine in a cluster. Moreover, it executes the installation sequence.

The Identity Manager automates the user life cycle with native repositories. It communicates directly with the access-system to help defining accounts creation, passwords and account privileges.

The Storage Manager provides self-configuring tasks to automatically identify and load drivers for storage devices connected to the server.

Self-Healing

Trivoli Enterprise Console inspects error logs, derives root problem cause and initiates corrective actions automatically.

Trivoli Switch Analyzer correlates network device errors to the rot case without human intervention. It is at level 2 out of 5 autonomic levels.

Trivoli NetView discovers TCP/IP networks, displays network topologies, correlates and managing events, monitors network health and gathers data. It has router fault isolation technology that identifies the cause of the error and consequently initiates corrective actions.

Trivoli Risk Manager contains self-healing technology developed by IBM research. It assesses security threats and automates responses for server reconfiguration, patch deployment and account revocation.

Trivoli Monitoring for Applications, for Databases and Trivoli Monitoring for Middleware automatically discovers, diagnoses and initiates problem resolution.

Trivoli Storage Resource Manager notices automatically storage problems and executes policy-based actions to solve problems.

Self-Optimizing

Trivoli Service Level Advisor performs trend analysis based on historical performance data and makes predictions about critical thresholds for the future in form of events sent to the Trivoli Enterprise Console.

Trivoli Workload Scheduler for Application monitors, controls and automates the executions of workloads.

Trivoli Monitoring for Transaction Performance enables monitoring of performance and availability of transactions.

Trivoli Storage Manager supports adaptive differencing technology that helps optimize resource usage for backup.

Self-Protecting

Trivoli Access Manager self-protects by preventing unauthorized access and has the ability to control to resources for authenticated users.

Trivoli Identity Manager self-protects by centralizing identity management, integrating automated workflow with business process.

Trivoli Risk Manager provides system wide self-protection by assessing potential threats and automating responses to help administrators respond accordingly.

6.2.3 Process Development Evolution

"*The Trivoli group at IBM is introducing autonomic computing in a step-by-step, phased manner. This allows customers to leverage their existing technical and staff resources with today's autonomic offerings and enables them to progress to the next level as new offerings appear. This process can help customers achieve a steadily increasing return on their investments.*" Paul Mason, IDC

"*By focusing on staged delivery and what is actually available today and tomorrow, Trivoli software can help its customers work toward the autonomic computing vision.*" James Governor, Illuminata

6.2.4 Leveraging Open Standards

Open standards are essential for managing resources and processes across the system hierarchy layers. IBM leverages open standards as the foundation for its autonomic system management solution. According to R. Much [4] these standards currently include:

- Java Management Extensions,
- Web service-level agreements,
- Storage Networking Industry Association,
- Open-grid systems architecture,
- Web Services standards,
- Telecom management,
- Distributed Management Taskforce,

- Common Information Model,
- Internet Engineering Taskforce (Policy, Simple Network Management Protocol),
- Organization for the Advancement of Structured Information Standards (OASIS),
- Microsoft Windows management instrumentation.

6.3 IBM DB2 Database Management System

IBM DB2 is a leading example of autonomic computing in the database marketplace.

6.3.1 Autonomic Computing Characteristics

The following illustration is based on the work of Richard Munch in his book "*Autonomic Computing*" [4].

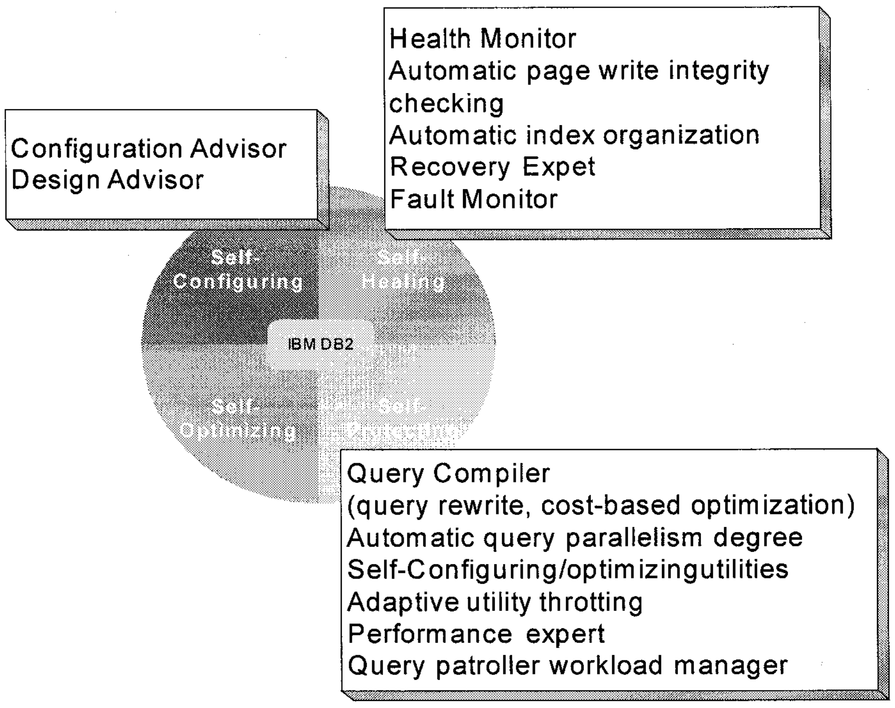

Configuration Advisor
Design Advisor
Health Monitor
Automatic page write integrity checking
Automatic index organization
Recovery Expet
Fault Monitor
Self-Configuring
Self-Healing
IBM DB2
Self-Optimizing
Self-Protecting
Query Compiler
(query rewrite, cost-based optimization)
Automatic query parallelism degree
Self-Configuring/optimizingutilities
Adaptive utility throtting
Performance expert
Query patroller workload manager

Figure 43: DB2 autonomic characteristics [4]

6.3.2 Process Development Evolution

As mentioned by IBM, future releases of DB2 will contain enhanced autonomic features available as for example autonomic maintenance policies and automatic database backup autonomic feature.

6.3.3 Open Standards

DB2 has a deep commitment and support for the following open standards [4]:

- Linux,
- Java,
- XML,

- Web Services,
- Grid computing distributed database interoperability, and
- Multi-vendor, multi-platform exploitation.

6.4 Sun Microsystems - N1

N1 stands for managing n number of computers as 1. N1 is Sun Microsystems [12, 13] product for grid computing. It provides services for managing heterogeneous environments and removes information technology complexity through technical means.

Sun was one of the first system vendors in supporting Grid Computing as a system architecture and design philosophy. When Sun acquired the technology in 2000, they decided to make the source code public in the open source Grid Engine Project, in order to promote its pervasiness and let people start use the Grid without barriers.

6.4.1 Modeling and Development

N1Ge 6 is Sun's latest proprietary product based on Grid Engine technology with added enterprise-level capabilities and it equipped with a powerful system administration tool called Sun Management Center. The latter is:

"*A single point of management for Sun Systems, the Solaris Operating Environment, applications and services for data center and highly distributed computing environments.*" [12]

According to Suns' documentation [12, 13], the Cluster Grid is the simplest form of a grid (i.e. multiple systems interconnected through a network). It has a three-tier general architecture as depicted in the following figure:

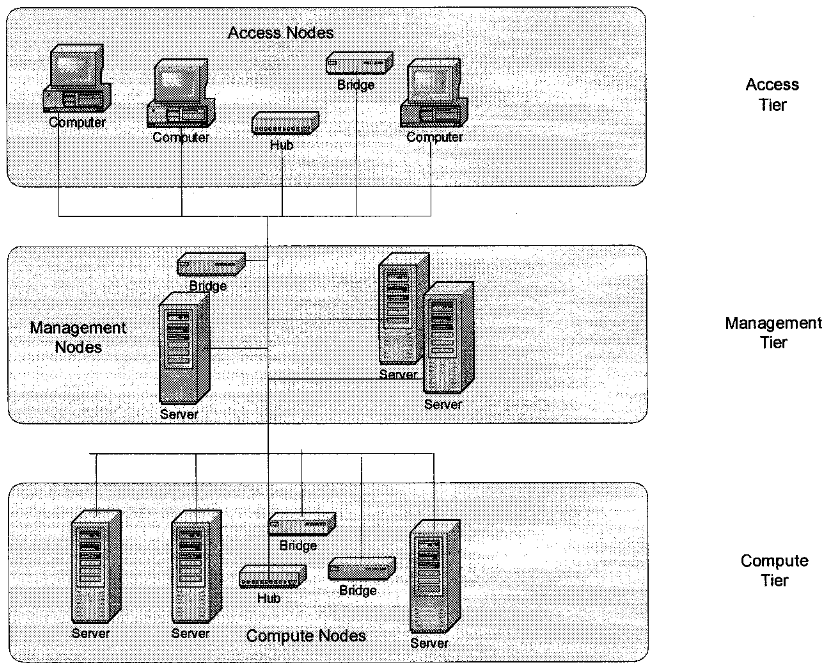


Figure 44: Grid cluster general architecture [12,13]

Access Tier

The Access Tier provides access and authentication to grid users. Access methods are based on authentication schemes such as NIS, LDAP, and Kerberos [12].

Management Tier

The Management Tier includes one or more servers, which run the *server* part of client server software such as **Distributed Resource Management (DRM),** hardware diagnosis components and system performance monitors.

Compute Tier

The Compute Tier includes servers that run the *client* part of **DRM**, the daemons associated with message passing environments, and agents for system health monitoring. The compute tier communicates with the management tier, receiving jobs to run and reporting job status and accounting details.

Cluster Grid Stack is Sun's implementation of the general architecture above is illustrated in the following figure:

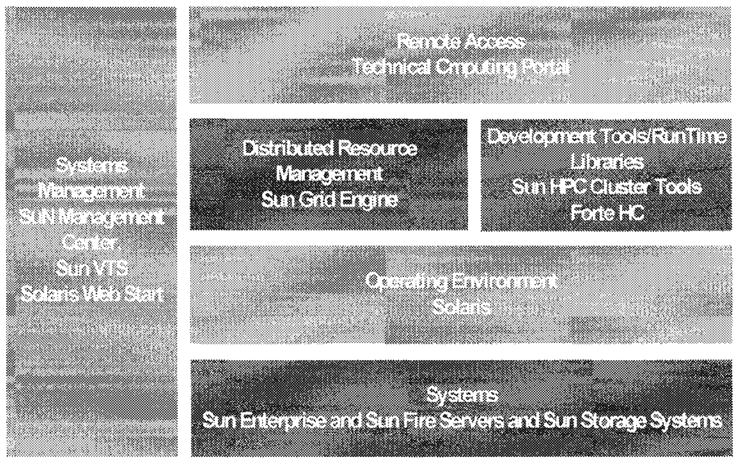


Figure 45: Sun Management Center within N1 grid cluster architecture [12, 13]

This is a software stack design, with each layer in the stack representing different functionality. As it can be noticed from the figure below the architecture is open and modular. While the software components map into the logical three layers there is not always a one-to-one correspondence. For example Sun Management Center provides distributed resource management functionality and provides services at each layer in the architecture: access, management and compute tiers [12, 13].

Sun Management Center is based on intelligent agent reference model. In this approach, a manager monitors and controls managed entities by sending requests to agents residing on the managed node. Agents collect management data on behalf of the manager.

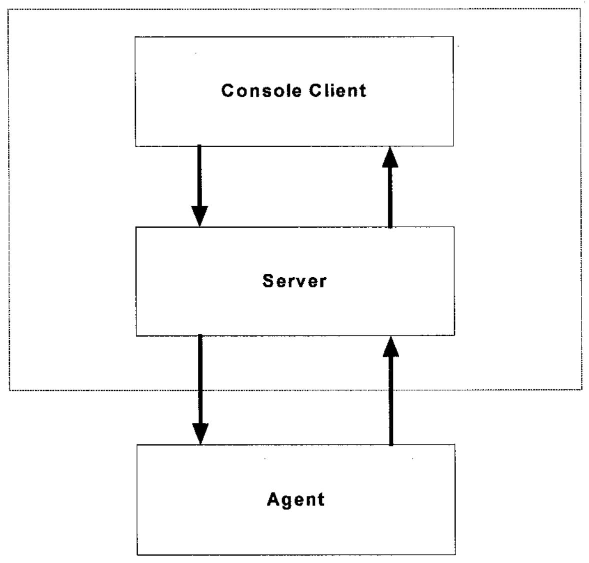


Figure 46: Sun Management Center's intelligent agent-based reference model [12]

Sun Management Center utilizes autonomous agent technology to drive its autonomic capabilities. Autonomous agent technology is a technique in which agents are not dependent of other software components, and all data collection and processing is done locally at the agent. These intelligent agents can act on data: initiate alarms, administrative notifications or specific actions.

Framework

The development tools and run-time libraries provide an integrated software environment for developing parallel distributed applications.

6.4.2 Autonomic Computing Characteristics

System Management includes powerful system administration tools, test and verification tools and automated installation and deployment technologies. The information presented in this subsection on based on Sun's documentation posted on Sun's web site [13].

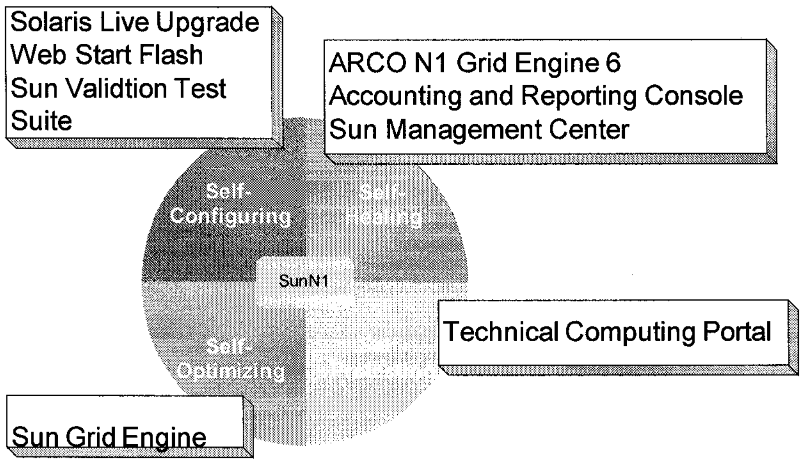


Figure 47: Sun-N1 autonomic characteristics

Self-Configuring

Solaris Live Upgrade and *Web Start Flash* provide automated installation and deployment technologies. With Solaris Live Upgrade technology, systems can be upgraded while they run, reducing downtime.

Self-Optimizing

Sun Grid Engine distributed resource management software optimizes utilization of software and hardware resources in heterogeneous networked environments.

Self-Healing

Sun Management Center, which is based on agent technologies provides self-healing autonomic capabilities.

ARCO N1 Grid Engine 6 Accounting and *Reporting Console* provides a comprehensive way to collect and analyze extensive detailed statistics of usage on the Grid.

Self-Protecting

Technical Computing Portal provides high-performance technical computing users with secure anytime, anywhere access to a single Web based point of delivery for service, content, and complex applications through a standard Internet browser and simple user interface.

6.5 Hewlett-Packard's Adaptive Enterprise

Hewlett-Packard's vision of autonomic computing is called "The Adaptive Enterprise" [14, 15] and it reflects its reference architecture, called "Darwin Reference Architecture" [15]. The Adaptive Enterprise framework enables businesses to create a tighter connection between system management and business process.

6.5.1 Modeling and Development

Darwin architecture is based on the fact that all its components comply with the following design methodologies: "simplification, standardization, modularity and integration" [15]

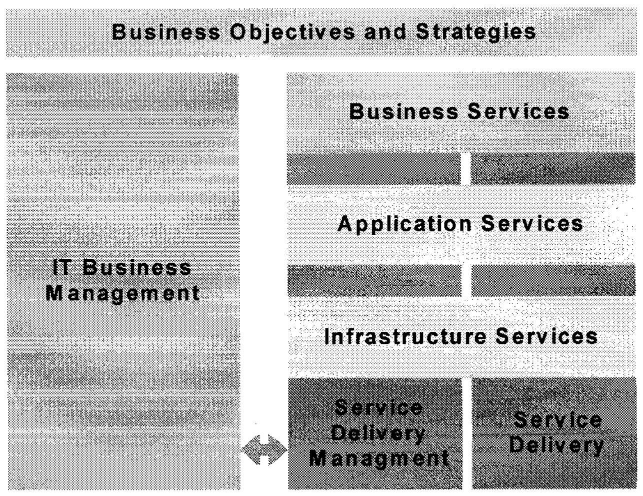


Figure 48: Darwin architecture [15]

The **Manage and Control Software** is within IT Business Management vertical dimension and:

"*It coordinates and controls the infrastructure through continuous inventory, monitoring, planning, provisioning, control and maintenance of the whole environment.*" [15]

The Manage and Control software is present at each of the three levels: the business level, the service level and the resource level. Each level has six layers namely plan, provision, inventory, control, monitor and maintain.

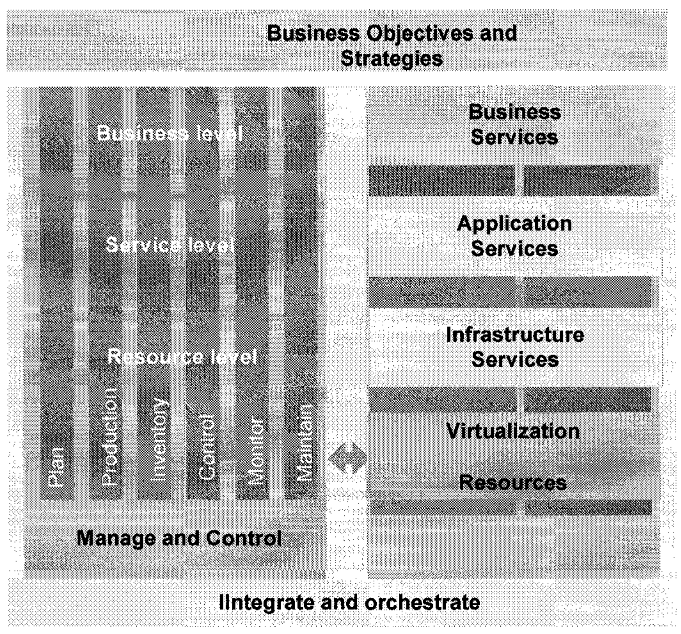


Figure 49: Manage and Control within HP's Darwin architecture [15]

"*The fundamental design elements of Adaptive Enterprise are Service Oriented-Architecture (SOAs), model driven automation and virtualization.*" [15]

6.5.2 Autonomic Computing Characteristics

Self-Configuring

The Adaptive Enterprise establishes an IT environment that can change quickly and is able to deal with dynamic changes, mergers and acquisitions and integration of new business. For example *HP Blade System Infrastructure* allows automated, policy-based deployment.

Self-Healing

HP Blade System Infrastructure provides auto-failover protection and advanced workload management across the entire system infrastructure. In addition, HP contains *the HP BladeSystem Management Suite* that leverages an integrated virtualized and automated solution.

Self-Optimizing

"Sharing IT resources eliminates underutilized or over-employed resources, spreads the load and smoothes out components" [14]. Following are few examples of self-optimization trough virtualization:

- The usage of single or multiple server environments are optimized by their configuration as pools of resources.
- *HP Storage Architecture*, removes physical storage yielding improved capacity utilization and data availability.
- *HP Open Call* allocates capacity dynamically in networks.

6.5.3 Process Development Evolution

HP development evolution includes three levels. Following paragraphs highlight the levels based on the information found on HP's white paper [14].

Element Virtualization

First level is where single resources are optimized to fulfill requirements within a single application. Resources can be: servers, storage, network, software, etc.

Integrated Virtualization

Second Level is where multiple resources are optimized within a single application to automatically fulfill service level agreements.

Compete IT Utility

Third level where all resources are pooled and shared over applications to automatically fulfill demand in real time.

6.5.4 Open Standards

HP is a major player in driving grid standardization [14] as:

- it maintains a leadership position in the Global Grid Forum, responsible for OGSA
- itself developed a World Services Management Framework (WSMF) which became a basis for OASIS/WSMD standard.

6.6 Microsoft's Dynamic Systems Initiative

Dynamic System Initiative (DSI) is a Microsoft effort to incorporate into Microsoft Windows platform a number of solutions that will ultimately implement autonomic characteristics.

"Microsoft is readying two new technologies that it says will provide the company with the same kind of self-configuration and management capabilities that IBM has been touting for the past several years." [13]

6.6.1 Modeling and Development

Microsoft autonomic computing architecture is based on System Definition Model (SDM). The SDM is a model that is used to create definitions of distributed system, i.e. definitions of resources, endpoints, relationships and sub-systems. In addition, the SDM contains deployment information, installation process, schemas for configuration, events, automation tasks, health models and operational policies.

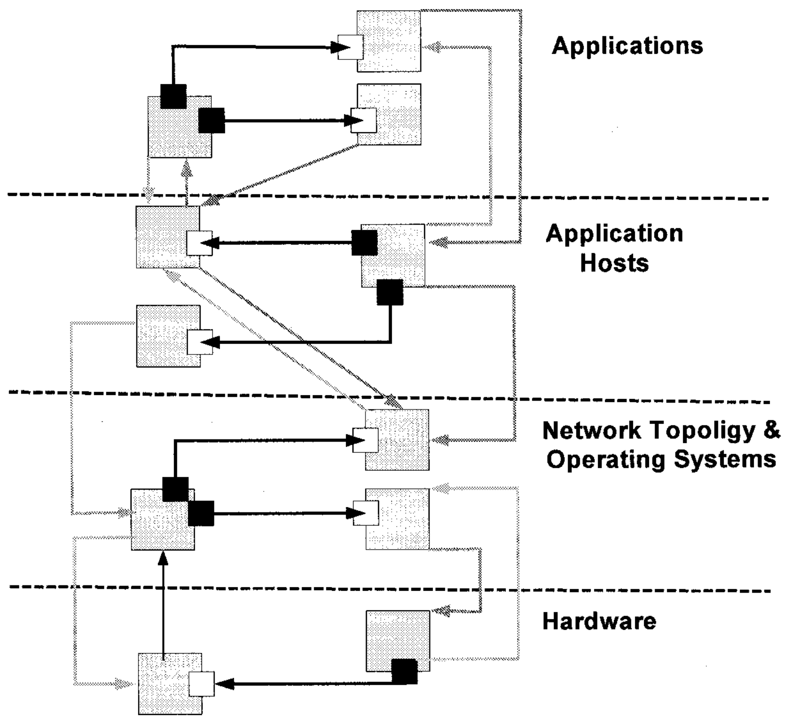


Figure 50: Microsoft's reference model [13]

Here for clarity we provide reference model definitions based on the information from Microsoft white paper [13].

Distributed System

A distributed system is composed of resources, endpoints, relationships and sub-systems.

Resources

In a distributed systems resources can be hardware components or software components.

Endpoints

Endpoints represent communications across the system.

Relationships

Relationships define associations between systems, resources and endpoints.

Sub-Systems

Sub-Systems are complete self-contained systems.

According to the white paper [13], SDM can be used through entire IT life cycle as follows:

At Design

SDM is used to define a system composed of hardware and software components i.e. all the information including necessary resources, configuration, operational features, policies, etc.

At Deployment

SDM is used to automatically deploy the system by dynamically allocating and configuring software and hardware resources.

At Operations

SDM provides a system-level view for managing the distributed system based on its definition.

Framework

Microsoft's approach is to create a framework that will enable systems to be designed with operation in mind:

"By creating this integrated feedback loop spanning the entire life cycle of a system we can facilitate the ongoing improvement of IT infrastructure with software". [13]

6.6.2 Autonomic Computing Characteristics

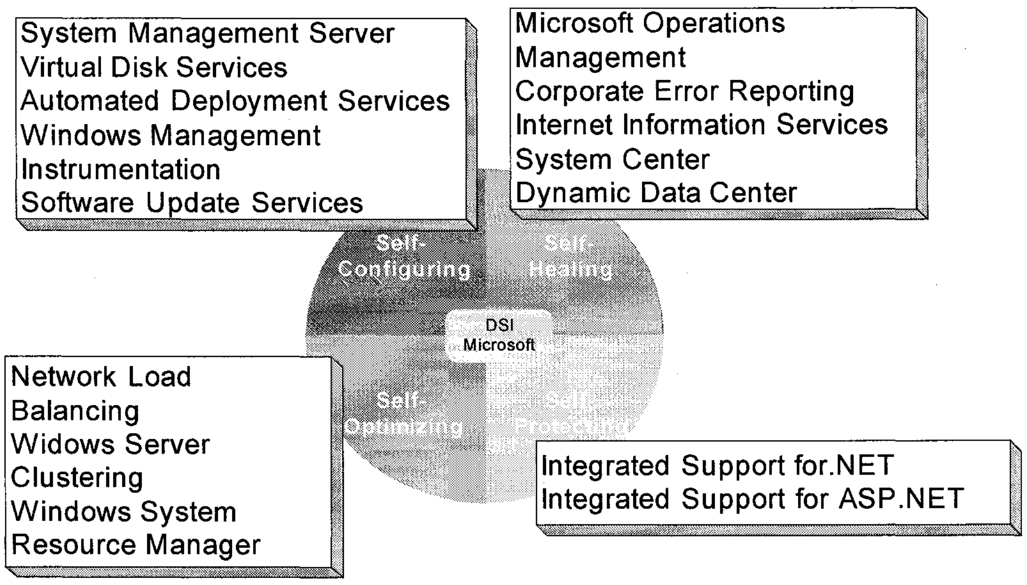

System Management Server
Virtual Disk Services
Automated Deployment Services
Windows Management Instrumentation
Software Update Services
Microsoft Operations Management
Corporate Error Reporting
Internet Information Services
System Center
Dynamic Data Center
Self-Configuring
Self-Healing
DSI Microsoft
Self-Optimizing
Self-Protecting
Network Load Balancing
Widows Server Clustering
Windows System Resource Manager
Integrated Support for.NET
Integrated Support for ASP.NET

Figure 51: Microsoft's DSI autonomic characteristics [13]

Self –Configuring

Windows Server: Virtual Disk Service provides a vendor independent interface for identifying and configuring storage devices from multiple vendors.

Windows Server: Windows Management Instrumentation Command Line and new command line tools provides administrators with direct and unified management tools locally and remotely.

Windows Server: Software Update Services automatically delivers critical patches to target computers from a single Intranet.

System Management Server Application Development provides WAN aware capabilities to reliable deploy applications to thousands of workstations.

Self-Healing

Microsoft Operation Management (MOM) incorporates event management, proactive monitoring and alerting, reporting and trend analysis, and system and application specific knowledge to improve the manageability.

Corporate Error Reporting (CER) tool which is a mechanism to provide information about problems in applications to the vendor or the in-house application developer.

Windows Server Internet Information Services is a Web server with self-healing autonomic capabilities, supported by a new fault tolerant process model.

Self-Optimizing

System Management Server Asset Management monitors application and license usage.

Self –Protecting

Windows Server provides integrated support for .NET and ASP.NET leveraging a fully managed and protected application environment for Web and XML services.

6.6.3 Development Process Evolution

The following paragraphs highlight the development process evolution undertaken by Microsoft.

Foundation of Dynamic Systems (Windows Server 2003)

First phase is to provide a solid foundation on which dynamic autonomic system can be built.

Design for Operations

Design for operation is the next phase is to make it easy to design for operations employing the Software Definition Model.

Data Center

In the data center phase Windows will evolve to:

- virtualizes the entire system, top to bottom,
- manages distributed resources across a data center,
- provides user with system-level view of their environment, and
- offers core new services to simplify the deployment and operation of distributed systems.

Dynamic data Center Driven

Dynamic data center is the phase driven by business policy. At this last phase, "*Microsoft will deliver a closed loop system level, management solution that provide new levels of automation in the data center and tie business policies to IT process*" [13].

6.6.4 Open Standards

Microsoft is investing significantly in modeling technology, in particular XML, and works with industry partners to extend those technologies to heterogeneous environments. In addition, with ADM, Dell, Intel Corporation and Sun Microsystems announced the publication of Web Services Management a specification for systems to access and exchange information. The companies mentioned above plan to present the specification to the Distributed Management Task Force (DMTF).

6.7 Current on-going Universities Projects

6.7.1 OceanStore

OceanStore is a "global persistent data storage designed to scale up to billions of users" [16]. Researchers at Berkeley University of California are studying systems that perform continuous on-line adaptation called Introspective Computing through continuous optimization to adapt to server failures, denial of service attacks and autonomic computing.

6.7.2 Recovery-Oriented Computing (ROC)

Recovery Oriented Computing is a project within Berkeley University of California that explores autonomic computing techniques for building reliable Internet services. The researchers investigate recovery from failure techniques. David Petterson et al. [5] state that ROC focuses on Mean Time To Repair (MTTR) rather than on Mean Time To Failure (MTTF) in order to provide system availability in their technical report "*Recovery Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies*". The following six techniques are mentioned in the report: (1) redundancy, (2) failure containment, (3) fault insertion testing, (4) error diagnosis, (5) non-overwriting storage systems and (6) enhanced availability.

6.7.3 Anthill

Researchers at University of Bologna, Italy are working on Anthill [18, 19] project. Anthill is a framework that leverages the design, implementation and verification of peer-to-peer application. Those applications can be visualized as Complex Adaptive Systems with inherent emergent behavior and interesting properties as resilience, adaptation and self-organization. Architecturally an Anthill system is composed of dynamic network of peer nodes, societies of adaptive agents (ants) that can travel over the network interacting with nodes and cooperating with other agents. The project is founded by Sun Microsystems.

6.7.4 Software Rejuvenation

Software rejuvenation is an on-going project at Duke University. According to researchers, software rejuvenation is a cost-effective technique to solve software faults by system restart, application restart and node/application failover [20].

6.7.5 J2EEML

To make development of autonomic application easier, researchers at Vanderbilt University Nashville, TN have developed J2EEML: Applying Model Driven Development to Autonomic Enterprise Java Bean Systems or J3 Process. According to Jules White, Douglas Schmidt, Aniruddha Gokhale in *"Simplifying the Development of Autonomic Enterprise Java Bean Applications via Model Driven Development"* [21] their project has the following four components:

(1) a domain specific language J2EEML, for describing autonomic EJB systems, their goals and their adaptation plans,

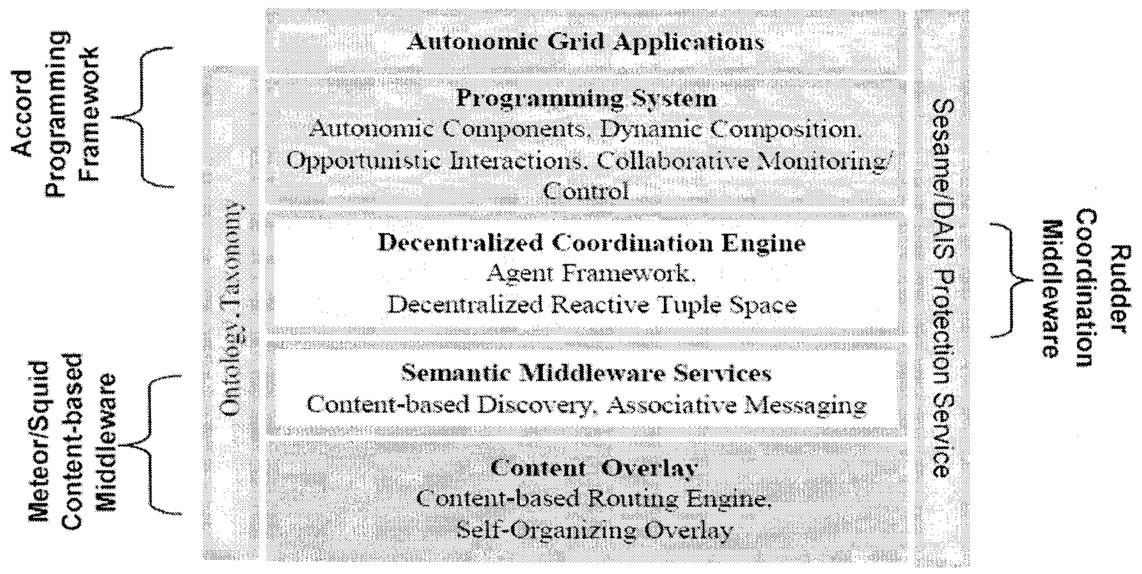
(2) a framework for Java called Jfense, and

(3) J2EEML model interpreter called Jadapt to make developing of autonomic systems more feasible.

J2EEML is actually a model-driven development (MDD) tool that can formally capture the design of EJB systems (EJB Structural Model), their quality of service (QoS) requirements (Goal Model), and the autonomic properties that will be applied to the EJBs (Goal-to-EJB Mapping). It supports quick development of autonomic EJB applications via code generation, automatic checking of model correctness, visualization of complex QoS and autonomic properties [21].

6.7.6 AutoMate

This research project is conducted at the Applied Software Systems Laboratory Rutgers, the State University of New Jersey. The researchers presented their project AutoMate: Enabling Autonomic Applications at UPP – Autonomic Computing Mt St. Michel, France, September 15-17, 2004.


Accord Programming Framework
Meteor/Squid Content-based Middleware
Ontology, Taxonomy
Autonomic Grid Applications
Programming System
Autonomic Components, Dynamic Composition, Opportunistic Interactions, Collaborative Monitoring/Control
Decentralized Coordination Engine
Agent Framework,
Decentralized Reactive Tuple Space
Semantic Middleware Services
Content-based Discovery, Associative Messaging
Content Overlay
Content-based Routing Engine,
Self-Organizing Overlay
Sesame/DAIS Protection Service
Rudder Coordination Middleware

The project has the following architecture:

Figure 52: AutoMate's autonomic computing architecture [23]

Hua Liu and Manish Parashar present the Accord Programming Framework component for autonomic applications, in the article "*A Component Based Programming Framework* *for Autonomic Applications*" [22]. It can be noticed from the architecture above that Accord builds on a middleware that provides services required for the autonomic applications. The following paragraphs detail the four concepts of Accord according to the article [22]:

1. **Defining Application Context** – agreement by the autonomic components on commons syntax and semantics to define and describe ontologies, namespaces, sensors, actuators, interfaces events.
2. **Defining Autonomic Component**. The definition of autonomic component is illustrated in the following figure:

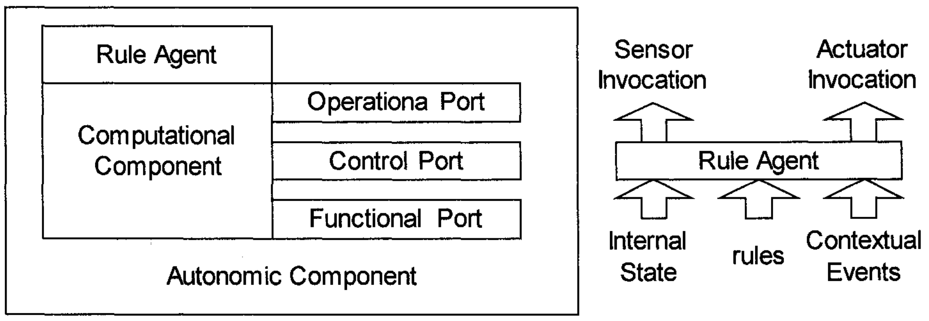


Figure 53: Accord autonomic component [22]

The structure of an autonomic component in Accord includes the following:

(1) a functional port which is a set of functionalities provided and used by the autonomic component,

(2) a control port is a set of sensors and actuators and a set of constraints that control the access to the sensors and actuators,

(3) an operational port which is a set of rules. It has the capability to define, introduce new rules and manage existing rules, and

(4) a rule agent which monitors the state of the autonomic component and its context and controls the firing of the rules. At the application level rule agents cooperate to fulfill application objectives.

3. **Dynamic Composition of Autonomic Components** - definition of rules and mechanisms to define organization and interaction between components.
4. **Agent infrastructure** - for self-management and dynamic composition behavior. The multi-agent system consists of peer rule agents embedded within autonomic components mentioned above and a composition agent.

6.7.7 Autonomia

Autonomia, an Autonomic Computing Environment [25] is an on-going research project at University of Arizona. It is an autonomic computing development environment, which includes the following main modules: Application Management Editor (AME), Autonomic Middleware Services (AMS), Application Delegated Manager (ADM) and Monitoring Services.

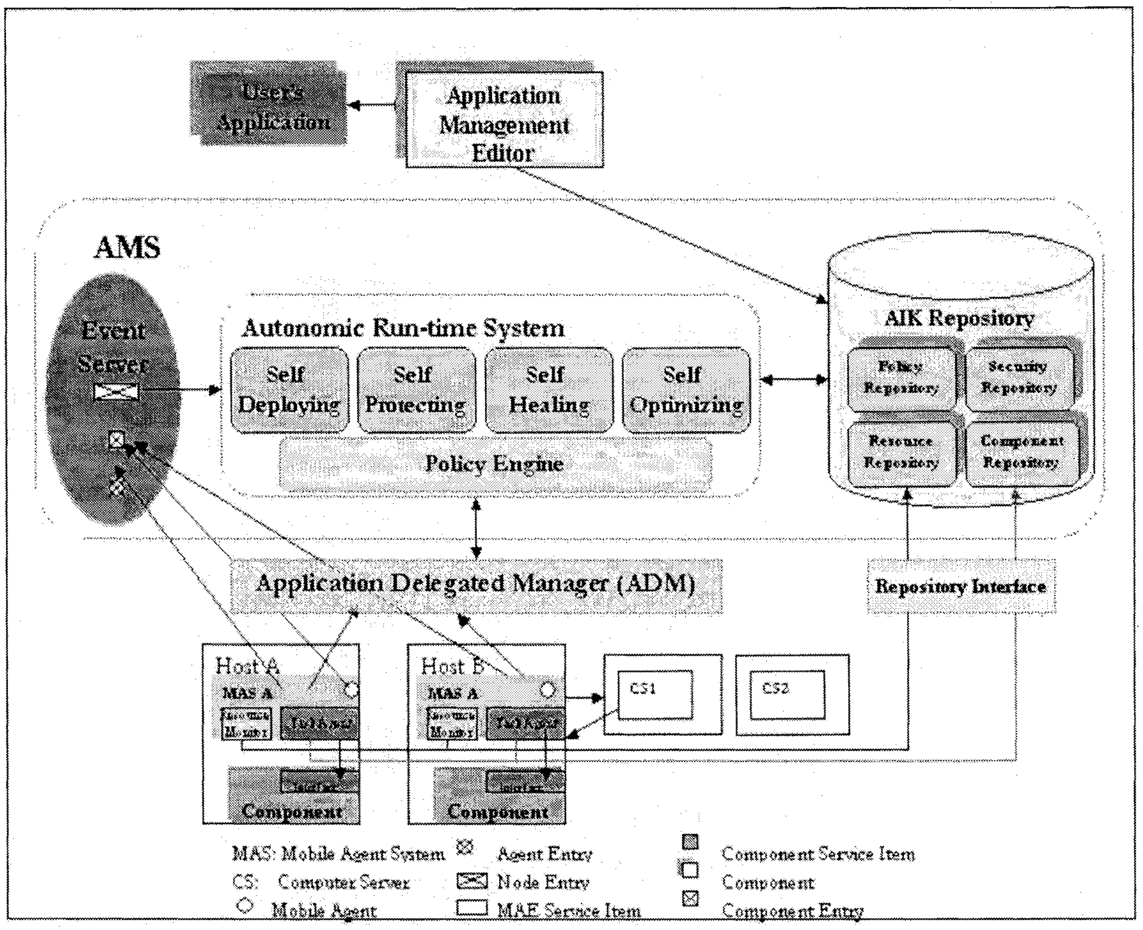


Figure 54: Autonomia mobile multi-agent autonomic computing architecture [25]

Autonomia is envisioned to leverage development and deployment of smart applications and provide a secure and open computing environment. In addition, it automates smart applications performance, fault tolerance, deployment, registration, discovery of their components, automate their configuration and system resources.

Autonomia is a mobile multi-agent system based on Java/Jini technologies. In the architecture illustrated above, the Middlware Services (AMS) provide applications with all the services and tools necessary to achieve autonomic requirements. The implementation of AUTONOMIA is discussed in detail in the article "*AUTONOMIA: An*

Autonomic Computing Environment" [24] by Xiangdong Dong, Salim Hariri, Lizhi Xue, Huoping Chen, Ming Zhang, Sathija Pavuluri, Soujanya Rao.

6.7.8 Autonomic Computing Infrastructure (MAACE)

MAACE is a project conducted at Institute of Artificial Intelligence, Zhejiang University, Hangzhou China and aims to support the development and deployment of intelligent applications. To date, the team of researchers have implemented a prototype system that enables self-configuring and self-optimizing of any networked application. MAACE's architecture is depicted in detail in the article "*Multi-Agent System based Autonomic Computing Environment*" [26] by Jun Hu, Ji Gao,and Jiu-Jun Chen. The authors note that the architecture is based on previous work: An Infrastructure for Managing and Controlling Agent Cooperation and An Infrastructure for Managing and Controlling the Social Behavior of Agents.

6.7.9 Smart Grid

Smart Grid [27] is a project conducted at Columbia University where researchers apply autonomic computing principles in order to solve Grid problems.

References

[1] Jeffrey O. Kephart, David M. Chess, "*The Vision of Autonomic Computing*", Published by IEEE Computer Society, Volume 36 (1), pp. 41-50, 2003, available at: http://www.research.ibm.com/autonomic/research/papers/AC_Vision_Computer_Jan_2003.pdf

[2] Paul Horn, "*Autonomic Computing: IBM's Perspective on the State of Information Technology*", IBM Corporation, October 15, 2001, available at: http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf

[3] A. G. Ganeck, T.A. Corbi, "*The Dawning of the autonomic computing era*", IBM Systems Journal, Volume 42, No. 1, 2003, available at: http://www.research.ibm.com/journal/sj/421/ganek.pdf

[4] Richard Murch, "*Autonomic Computing*". Prentice Hall Professional Technical Reference, IBM Press, pp. 235-245, 2004.

[5] David Patterson, Aaron Brown, Pete Broadwell, George Candea, Mike Chen, James Cutler, Patricia Enriquez, Armando Fox, Emre Kıcıman, Matthew Merzbacher, David Oppenheimer, Naveen Sastry, William Tetzlaff, Jonathan Traupman, Noah Treuhaft, "*Recovery Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies*", Technical Report CSD-02-1175, University of California-Berkeley, March 2002.

[6] K. Evans-Correia, "*Simplifying Storage Management Starts with More Efficient System Utilization* ", Interview with N, Tabellion, searchStorage, 29 August, 2001.

[7] Aberdeen Group Inc, "*IBM Data Management Tools: New Opportunities for Cost-Effective Administration*", Profile Report, April, 2002.

[8] D. Petterson, "*Availability and Maintainability >> Performance: New Focus For a New Century*", Conference on File and Storage Technologies (FAST'02), January 28-30, 2002.

[9] A. Brown, D.A. Patterson, "*To err is human*", Proceedings of the first Workshop on Evaluating and Architecting System Dependability (EASY '01), July, 2001.

[10] Yankee Group, "*How much is an Hour of Downtime Worth to You*?" from Must-Know Business Continuity Strategies, July, 2002.

[11] D. J. Clancy, "*NASA Challenges in Autonomic Computing*", Almaden Institute, IBM Almaden Research Center, San Jose CA, April, 2002.

[12] Sun Microsystems, "*Sun Cluster Grid Architecture*", White Paper describing the foundation of Sun Grid Computing, 2002, available at:
http://whitepapers.silicon.com

[13] Sun Microsystems, "*ARCO, N1 Grid Engine 6 Accounting and Reporting Console*", White Paper, May, 2005, available at:
http://www.sun.com/software/gridware/ARCO_whitepaper.pdf

[13] Microsoft Corporation, "*Microsoft Dynamic Systems Initiative*", White Paper, October 2003, available at:
http://download.microsoft.com/download

[14] Hewlett Packard, "*HP virtualization solutions: IT supply meets business demand – Enabling the Adaptive Enterprise*", White Paper, 2003, available at:
http://whitepapers.zdnet.co.uk/0,39025945,60094872p-39000636q,00.htm

[15] Hewlett Packard, "*HP's Darwin Reference Architecture Helps Create tighter Linkage Between Business and IT*", News Release, San Jose California, May 6, 2003, available at:
http://www.hp.com/hpinfo/newsroom/press/2003/030506b.html

[16] http://oceanstore.cs.berkeley.edu/

[17] http://roc.cs.berkeley.edu

[18] http://www.cs.unibo.it/projects/anthill

[19] http://www.cs.unibo.it/projects/anthill/documentation.html

[20] http://www.software-rejuvenaion.com

[21] Jules White, Douglas Schmidt, Aniruddha Gokhale, *"Simplifying the Development of Autonomic Enterprise Java Bean Applications via Model Driven Development"*, available at:
http://www.cs.wustl.edu/~schmidt/PDF/J2EEML.pdf

[22] Hua Liu, Manish Parashar, Salim Hariri, "*A Component Based Programming Framework for Autonomic Applications*", In Proceedings of the International Conference on Autonomic Computing (ICAC'04), pp. 10-17, New York, NY, 2004.

[23] http://automate.rutgets.edu

[24] Xiangdong Dong, Salim Hariri, Lizhi Xue, Huoping Chen, Ming Zhang, Sathija Pavuluri, Soujanya Rao, "*AUTONOMIA: An Autonomic Computing Environment*", Proceedings of the Computing and Communications Conference, 2003.

[25] http://www.ece.arizona.edu/~hpdc/projects/AUTONOMIA/

[26] Jun Hu, Ji Gao, Jiu-Jun Chen, "*Multi-Agent System based Autonomic Computing Environment*", Proceedings of the Third International Conference on Machine Learning and Cybernetics, Shanghai, Volume 1, pp. 105-110, 26-29, August, 2004.

[27] Columbia University Smart Grid information available at:
http://www.ldeo.columbia/ed/res/pi/4d4/testbeds/

Chapter 7 Future Research Directions

7.1 Exploratory Approaches

Practically every aspect of autonomic computing leads to a future research direction as autonomic computing spawns hardware, software and most importantly self-regulatory human characteristics.

"*The underlying technologies to enable greater automation of system management are ripe for innovation.*" [1]

In addition, in 2001 Paul Horn [1] mentioned the following approaches for exploration:

- artificial intelligence,
- control theory,
- cybernetics,
- self-evolving systems,
- load management, and
- cellular chips.

These approaches were further extended and detailed in another IBM article issued in 2003, called "*The Vision of Autonomic Computing*" by Jeffrey O. Kephart and David M. Chess. They identified emergent engineering and scientific challenges for autonomic computing. The following paragraphs depict some interesting scientific exploratory principles for autonomic computing.

Behavioral abstractions and models

The authors note that fundamental mathematical work, advanced search and optimization techniques with parameterized models of the local-to-global relationships and control theoretical approaches are just a starting point for research into behavioral abstractions and models.

Robustness theory

Another path toward autonomic computing is the research into robustness, diversity, redundancy and optimality including relationships between them.

Learning and optimization theory

Learning and optimization are well supported by theories for single agents. However in multi-agent systems they become challenging problems as so far the research has yielded only empirical results but no solid theorems.

Negotiation theory

Research is needed for negotiation theories from the system perspective as a whole, that is, a compound of negotiation algorithms of multiple autonomic elements.

Emergent phenomena theories

Insights that are gathered from scientists studying nonlinear dynamics and emergent systems constitute another path for research to support autonomic computing.

Psychology theories

Research for new goal definition and visualization paradigms is needed to "*help humans build trust in autonomic systems*" [2].

Open Standards

From the trials of modeling and development of autonomic systems has emerged a crucial need of standardization. Definitely, extensive research is needed such that new emergent standards will ultimately leverage the final solution of autonomic computing.

There are many design patterns highlighted in this survey as a reference from an article, blueprint, white paper, website, documentation; each one is an avenue for a research direction towards the paradigm of autonomic computing and then in particular towards Autonomic System Timed Reactive Model (AS-TRM).

To date, a first step has already been taken towards extending the TROM formalism to Autonomic System Timed Reactive Model (AS-TROM). H. Kuang presents the characteristics of AS-TRM, the architecture and communication mechanism of AS-TRM for implementing autonomic as well as reactive functionalities in his master thesis called

"*Architecture for Autonomic System: AS-TRM Approach*" [3]. In addition, he describes the reliability assessment model of AS-TRM for the evolution of an AS-TRM system.

References

[1] Paul Horn, "*Autonomic Computing: IBM's Perspective on the State of Information Technology*", IBM Corporation, October 15, 2001, available at: http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf

[2] Jeffrey O. Kephart, David M. Chess, "*The Vision of Autonomic Computing,*" Published by IEEE Computer Society, Volume 36 (1), pp. 41-50, 2003, available at: http://www.research.ibm.com/autonomic/research/papers/AC_Vision_Computer_Jan_2003.pdf

[3] H. Kuang, "*Architecture for Autonomic System: AS-TRM Approach*", Master of Computer Science Thesis, Computer Science Department, Concordia University, Montreal, Canada, 2006.