JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2005**

Vol. 4, No. 1, January-February 2005

The Platform Based-Agents to Test and Evaluate Software Architecture

Amar Ramdane-Cherif, Samir Benarif and Nicole Levy, PRiSM, Université de Versailles, Saint-Quentin en Yvelines, France

Abstract

Architecture conception is a difficult and time consuming process, requiring advanced skills from the software architect. The tasks of an architect are alleviated if means can be provided to generate architectures that can be evaluated with respect to functional and non functional requirements. This paper discusses of an easier approach to evaluate the software architecture. To achieve this goal, we orient our research to the development of intelligent and autonomous platform based-agents in order to evaluate and test the software architecture. The platform is oriented to intelligent system based-agents, which is an emerging technology that is making computer systems easier to use by allowing people to delegate work back to the computer. However, we exploit the advantage of the multi-agents systems like the flexibility, the performance, the parallelism and the high level of abstraction for the construction of our platform. We propose in our paper, a multi-agents platform which can be used, in static mode, as a tool to guide and help architect to make the right architectural choice during the design phase process. In this phase; the architects are interested in the outcome of the evaluation and have the power to make decisions that affect the future of the project. This platform can also be used, in dynamic mode, as tool to show to the architects different scenarios about the evolution and the behavior of one or several quality attributes. Each quality attribute is given an estimated value, using qualitative or quantitative assessment technique. Then, the platform offers the advantage to observe the relationships and the influence between several quality attributes required for an application. The platform offers the possibility to reconfigure dynamically the architecture in order to maintain one or several quality attributes. These qualities are represented by some scenarios grouped in some profiles that capture typical changes in quality requirements.

1 INTRODUCTION

The architecture of complex software or system is a collection of hard decisions that are very expensive to change. Successful product development and evolution depend on making the right architectural choices to achieve the quality required. An unsuitable architecture will precipitate disaster on a project. Performance goals will not be met.

Cite this article as follows: Amar Ramdane-Cherif, Samir Benarif, Nicole Levy: "The Platform Based-Agents to Test and Evaluate Software Architecture", in *Journal of Object Technology*, vol. 4, no. 1, January-February 2005, pp. 67-82. http://www.jot.fm/issues/issue_2005_01/article1

Security goals will fall by the wayside. The customers will grow impatient because the right functionality is not available, and the system is too hard to change. For this reason, we based our research on the conception of a platform which will operate at the top of the a software architecture. This platform must be able to know the mechanism of this architecture, to interact and monitor the system of this architecture, to give the important results about the quality attributes by testing and evaluating this architecture in dynamic mode and to help the architect to take the correct choices. Thus, the platform can play the role of managing the architecture and maintaining the quality required by the its dynamic reconfigurations.

To construct this platform, we direct our research to the comprehension of the architecture, the dynamic reconfiguration of an architecture and the multi-agents approaches. The "architecture" term conveys several meanings, sometimes contradictory. In our research we consider that architecture deals with the structure of the components of a system, their interrelationships and guidelines governing their design and evolution over time [1][2]. The architecture then becomes the basis of systematic development and evolution of software systems. It is clear that a new architecture that permits the dynamism reconfiguration, adaptation and evolution while ensuring the quality management of an application is needed. In addition, the complexity of emerging applications and trend of building trustworthy systems from existing, untrustworthy components are urging quality concerns be considered at the architectural level. Therefore, architecture analysis can be used to evaluate the influence of the design decisions on important quality attributes. Software monitoring is a well-know technique for observing and understanding the dynamic behavior of programs when executed, and can provide for many different purposes [3][4]. Other purposes for applying monitoring are: testing, debugging, correctness checking, performance evaluation and enhancement, security, control, program understanding and visualization, ubiquitous user interaction and dynamic documentation.

Recently, a number of new scenario-based software architecture evaluation methods have been developed by different academic groups and published in form of bock or doctoral dissertation theses. Many of these methods are refinement of Software Architecture Analysis Method (SAAM) [5][6] or Architecture Tradeoff Analysis Method (ATAM) [5][7]. They usually restrict themselves to a particular class of systems. For example, Architecture Level Modifiability Analysis (ALMA) [8][9] method focuses on the modifiability of business information system. Another newly developed approach, the Family Architecture Analysis Method (FAAM) [10], assesses the interoperability and extensibility of information system families. Actually, scarce tools supporting evaluation session methods exist. For example, in the SAAM, the voting procedure invoked for the scenario prioritization and modifiability estimates with respect to cost and effort to adapt the architecture are the only techniques used. Newly, The ATAM session count a tool to support the evaluation [10].

Our approach can be used as tools to support certain steps of all evaluation method based-scenarios, precisely questioning techniques (scenarios) and measuring techniques (metrics, simulations, prototypes and experiments) [11][12]. This paper covers following

topics: In the first, we introduce our platform and describe its structure, the next section we explain how the platform monitors the quality attributes of the software architecture and interacts with the environment. So, we present an example of the monitoring of some qualities and its application on real architecture. Finally, we finish this paper by discussion and conclusion.

2 THE PLATFORM MULTI-AGENTS

In recent years, agents and Multi-Agent Systems (MAS) have become a highly active area of Artificial Intelligence (AI) research. Agents have been developed and applied successfully in many domains. MAS can offer several advantages in solving complex problems compared to conventional computation techniques. The purpose of traditional Artificial Intelligence is to perform complex tasks, thanks to human expertise. This often assumes assimilation of many competencies to be subject of centralized programming. Moreover, in such monolithic system, the consensus between various expertise is difficult to model; indeed, the structure of communication between the experts is fixed whereas it should depend on the considered problem. Thus, a formalization close to reality where several people work together on a same problem is needed. Such formalism should describe the participants and interactions between them. This approach is the paradigm of the Distributed Artificial Intelligence (DAI). The DAI leads to the realization of systems known as "multi-agent" systems allowing modeling the behavior of all the entities according to some laws of social type. These entities or agents have certain autonomy and are immersed in an environment in which and with which they interact. Their structure is based on three main functions: perceiving, deciding and acting.

The conception of our agents is based on two descriptions. The first is a reasoning view point of agents, it can be considered as a system of reasoning, aiming at determining the possible actions, privileges, and coordination with the environment and other agents. The second description is a cooperative view point of agents. It uses vowel approach, it is based on four dimensions which are: Agent (A), Environment (E), Interaction (I), and Organization (O) (see Figure 1). Facet (A) indicates the whole of the functionalities of internal reasoning of the agent. The facet (E) gathers the functionalities related to the capacities of perception and actions of the agent on the environment. Facet (I) gathers the functionalities of interaction of the agent with the other agents (interpretation of the primitives of the communication language, management of the interaction and the conversation protocols). The facet (O) can be most difficult to obtain, it relates to the functions and the representations of the capacities of structuring and management of the relations between agents.

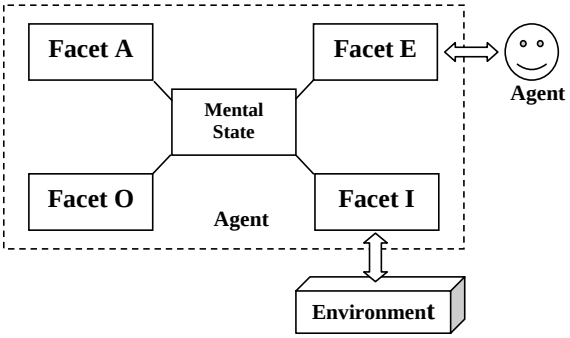


Figure 1: Cooperation view point of the agent.

Our goals is a creation of an autonomous platform able to act on the software architecture. While following a logical reasoning, we can see three layers in our platform, one layer represents the intelligent part of the platform (decisional part), and another layer represents the active part of our platform (reactive part). The third layer acts as link between the decisional and the reactive parts of the platform (see Figure 2). This offers the advantages of the division of the tasks and the specialization of the layers. Other aspect of our problem is the dynamic nature of our architecture, indeed architecture does not cease to evolve, to reconfigure and to extend. It is inconceivable to create a rigid and static platform which can follow the evolution of this architecture. We must thus already think of such a dynamic and evolutionary platform so that it can constantly reach and follow the evolution of this architecture. We will consider that our software architecture is a such board cut out in small pieces. We consider that we can extend this board as parts are added. We have also the freedom to modify the parts and to make them move on the board. While considering this example, we will establish specific rules to the platform based multi-agents which we will build. We will consider that the available software architecture is divided into localities, grouped; they form one or several zones. This strategy will enable us to better control the characteristics of the modifiability and the extensibility of the available architecture.

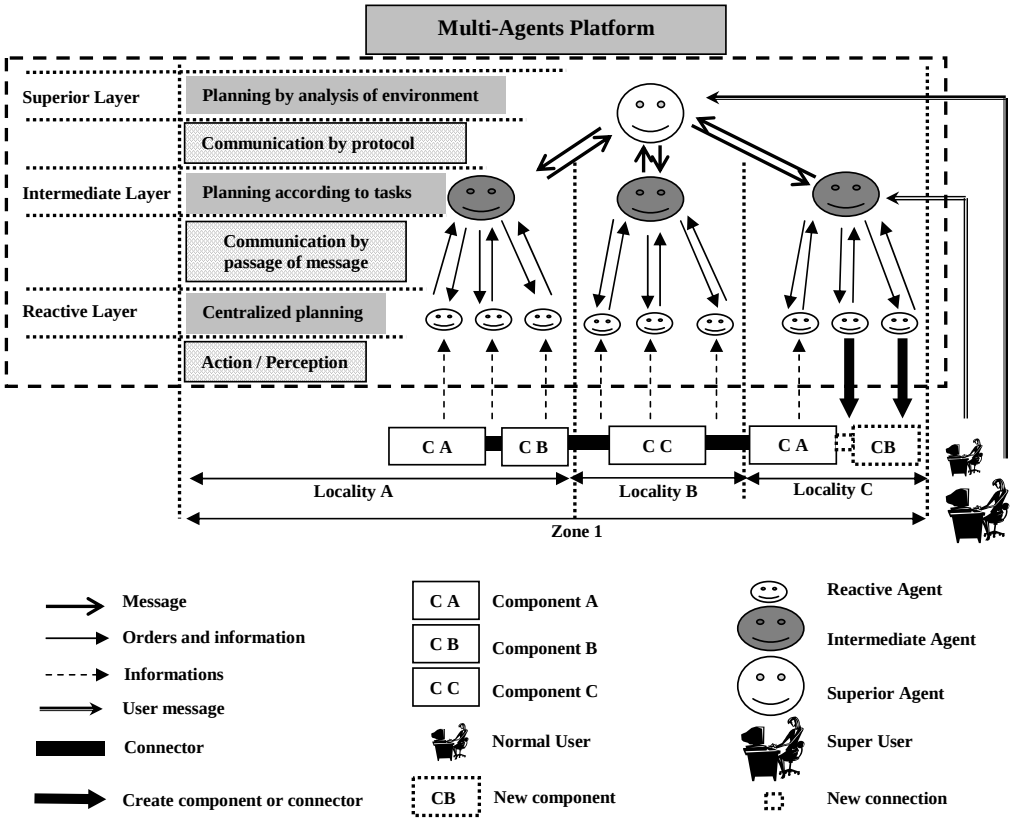


Figure 2: Structure of multi-agents platform

The architecture of our platform consists of three distinct layers.

The Higher Layer

The higher layer is the highest layer of the platform, it is thus, more evolved than the others. This layer has the capacity to analyze information coming from architecture, thanks to the facet E of its agents. Thus, it can evaluate the qualities of an architecture constantly and intervene in a targeted way, since the agents have a facet A, implying the reasoning. The facet O and I, of the agents enter in action when the agents of the intermediary layer do not manage to find a solution to a problem. The agents of the higher layer have the capacity to organize a group of agents in the intermediary layer (implies a cooperation) or to utilize another agent of the higher layer (implies a negotiation) in order to achieve the goal. The agents of this layer can constantly exchange information relating to the zone which they control so that they always have a global and complete architecture vision. Each agent of this layer controls a zone of architecture; it is responsible for a group of agents of the intermediary layer.

Intermediary Layer

As its name indicates it is a layer which is placed between the higher layer and the reactive layer. Each agent of this layer takes care of several agents of the reactive layer, it is responsible for a quite precise locality. The agent itself is connected to only one agent of the higher layer. A set of agents of the intermediary layer forms what is called a zone. The principal role of this layer is to take care of the good progress of the reconfigurations imposed by the higher layer. It is a question of controlling and coordinating the agents of the reactive layer in order to carry out and to achieve a goal. Another role of this layer is the collection of information coming from the reactive layer in order to forward them to the agent of the higher layer.

The Reactive Layer

This layer is the body of perception and of action of the platform. It is equipped with purely reactive agents which act with simple stimulus coming from the intermediary layer. The reactive agents belong to a locality depending on only one agent of the intermediary layer whose they receive the plans. These agents answer to a centralized planning and work in cooperation. The exchange between the reactive agents and the agent of intermediary layer is simple. The perception induces sending simple information toward the central agent, the action is the consequence of a stimulus or a simple command.

3 MONITORING SYSTEM

Software monitoring is a well-know technique for observing and understanding the dynamic behavior of programs when executed and can provide for many different purposes. We adopt this system and adapt it to the agent approach. We think that the utilization of agents increase the capability of the monitoring system because we add to it the advantages of the multi-agents system such as :

- Autonomous monitoring of the architecture.
- Intelligent monitoring.
- Possibility to introduce a new exception or a test by adding a new planning in the knowledge base of the agents.
- Parallel monitoring of components and their proprieties.
- Possibility to see several tests and the influence between them.
- Filtrate of critical information in the package of information for a higher identification of the fault in an architecture.
- Evolve treatment part of monitoring by introducing various behaviors of agents according to the environment (the information collected).
- Record of collected information in the database of agents for its future exploitation (for example, graphic representation).

- Possibility to initiate different parameters of monitoring, and capacity to change them at dynamical mode.

We decompose the monitoring system in two parts, the detection phase and treatment phase. The first part of system monitoring consists in collecting information from the system execution, detecting particular events or states using the collected data, analyzing and presenting relevant information to the user. As the information is collected from the execution of the program implementation, there is inherent gap between the levels of abstraction of the collected events, states of the software architecture. For monitoring, there are basically two types of monitoring systems based on the information collection: sampling (time-driven) and tracing (event-driven) (see Figure 3.a). By sampling, information about the execution state is synchronously (in a specific time rate), or asynchronously (through direct request of the monitoring system). By tracing, on the other hand, information is collected when an event of interest occurs in the system. Tracing allows a better understanding and reasoning of the system behavior than sampling.

The treatment part of monitoring occurs after detection phase, it reacts to the collected information. The treatment intervenes in two cases, when event is detected by agent or when state of component don't respect the constraints imposed by the user. The event occurs when exception is sent to monitor agent of reactive layer, this exception is due at the abnormal behavior of components (internal or external exception). The abnormal state of component is due to overtaking of limits imposed by the user (for example, memory consumption out of fixed limits). The treatment of monitoring system is an action of the platform, according its decisional part, two actions can occur. The platform can recover the warning (event or abnormal state) by adding, deleting or modifying components or connectors (for example by replacing the failure component by adding another component with the same functionality). Another solution is that the platform can recover the warning, by reconfiguring the interaction among components of the architecture (for example, isolation of failure components by reorienting their connections).

Detection Phase of the Monitoring System

a) The monitoring system in the case of state : The basic information is collected (by sampling) by reactive layer (see Figure 3.b) and sent to the intermediate agent (see Figure 3.c). The intermediate agent decrypts information by identifying the origin of the message, the type of the information and the value of test (see Figure 3.d). The higher agent receives the message from the intermediate agent and takes decision according to the nature of the message. If the value of the test respects the constraint, the state of the components will be saved in the state of the architecture knowledge base. If the value of the test does not respect the constraint, the higher agent reacts by using its knowledge base (planning).
b) The monitoring system in the case of event: The component emits an event to reactive agent (see Figure 3.b). The intermediate agent receives this information; identifies the sender agent (by its knowledge base of reactive agents) and the

name of the exception (see Figure 3.c). Then, it sends the message to higher agent. The higher agent decrypts the message of its intermediate agent and starts research procedure in order to identify the exception and find the solution in its knowledge base (planning) (see Figure 3.d).

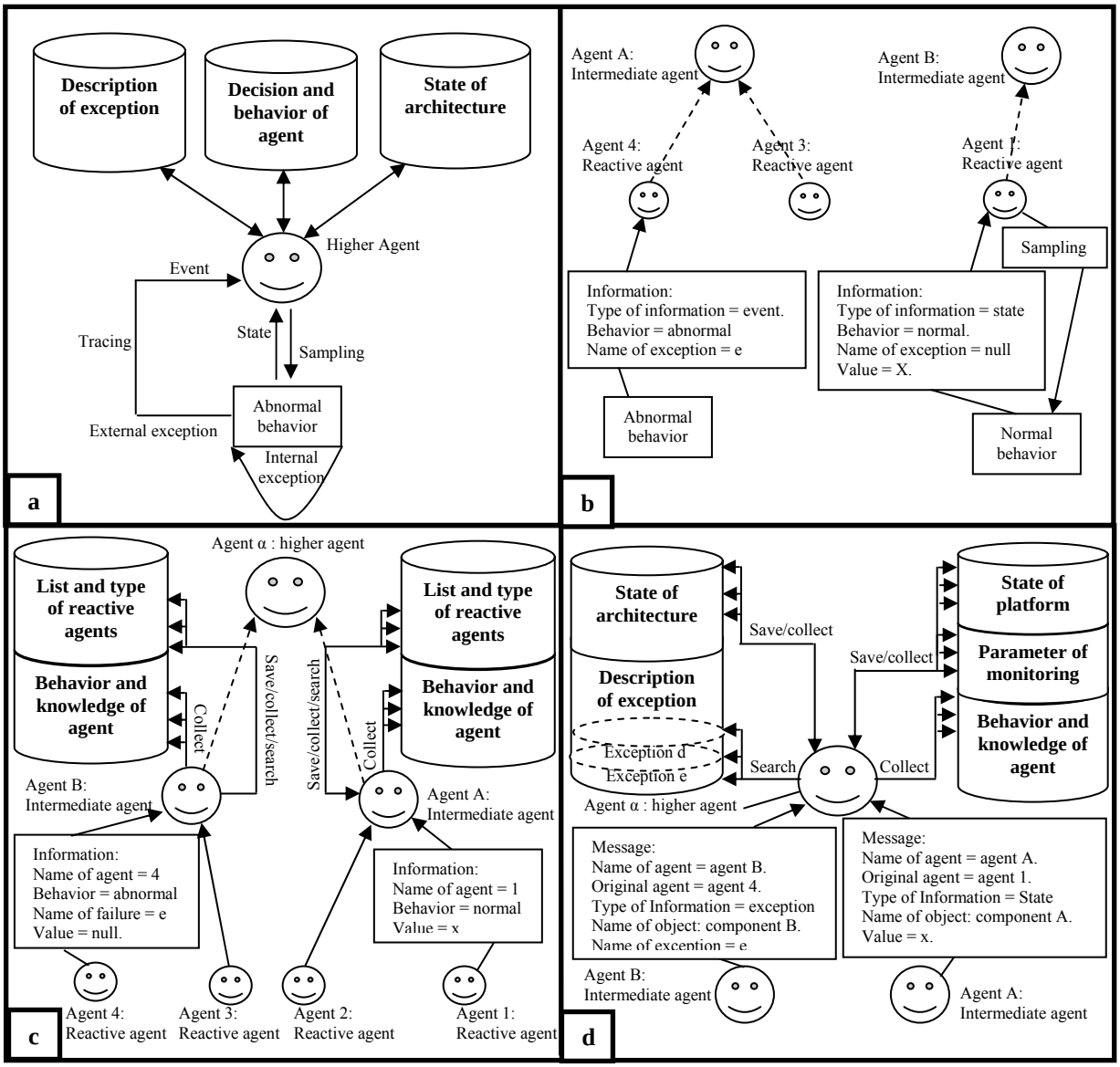


Figure 3: Detection phase of monitoring system

Treatment Phase of the Monitoring System

The monitoring system in the case of state and event: The higher agent reacts to the overtaking of constraints or events by using its planning in order to recover the fault or respect constraints (see Figure 4.a). The platform can recover the fault generated from the architecture by identifying the exception. The quality attributes are maintained by respecting the constraints (performance constraint, availability constraint, etc.) imposed by the user, so, the role of the platform is to monitor the architecture in order to verify the respect of the all constraints.

In the two cases (the case of state and event), the strategy is the same one, the higher agent takes decisions based on its knowledge base, organizes actions and sends the planning to its intermediate agents. The intermediate agent uses the cooperation of reactive agents to solve the problem and act on the architecture (see Figure 4.b). The treatment process uses tow types of plans, the first plan consists to reconfigure the architecture connections for finding temporary solution for the fault (disabled component or connector), and the second plan recovers errors by addition or changing disabled components or connectors.

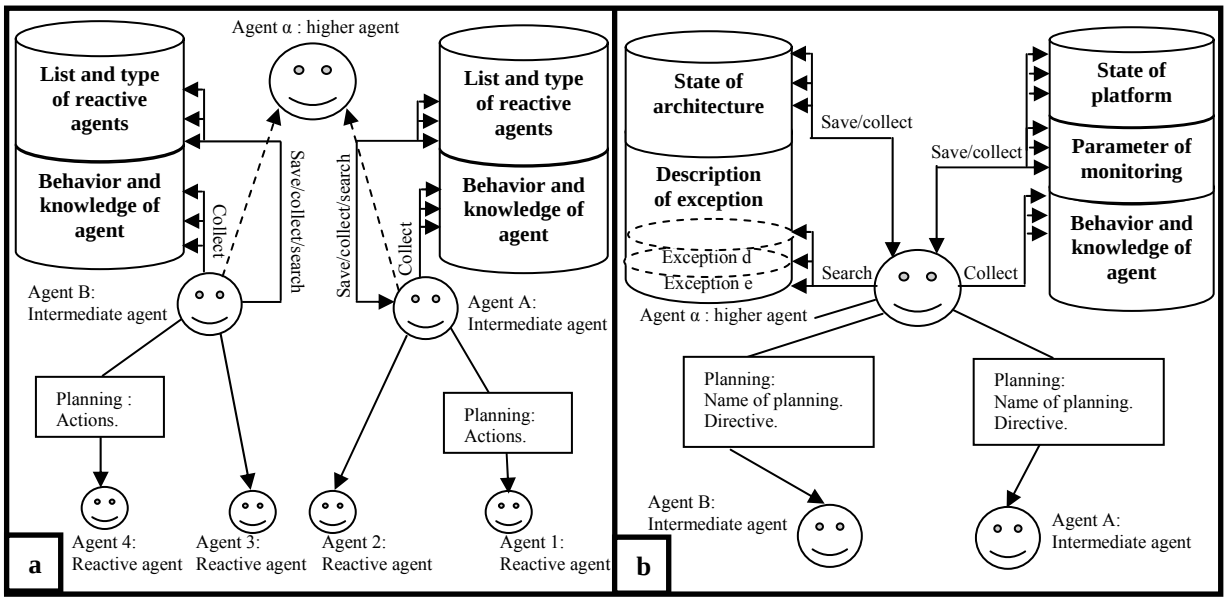


Figure 4: Treatment phase of monitoring system

4 IMPLEMENTATION OF THE PLATFORM ON CLIENT/SERVER ARCHITECTURE

The Problem

We realize our application by developing a software for a company specialized in hydrocarbon industry, it intervene in different building sites in hostile and distant places. This little company develops its activity and engages a lot of human resources, so the leaders require to manage and estimate the resources needed and the scheduling of the projects of the different sites. This information must be exchange between sites and the direction. The developed software contains the personnel files, the state on the projects (calculation of number of worked hours, days worked…etc.). The principal requirements of the customer are performance and availability. We have used our platform to test and maintain these quality attributes, materialized in different scenarios.

The Overview of the Software Architecture

We used to develop this software C++ and Access database, we dispose of two servers, principal server (server A) and second server (server B) and the dispatcher of connections (see Figure 5). The clients can connect to the data base by internal network (100 Mo/s network card) and via internet (56 Kb/s modem). The client can consult the personnel data base and perform the calculation on the server application.

In our application, the principal requirements of users in term of software quality are availability and performance. By applying the ATAM method of evaluation, we develop some scenarios. We present two of these scenarios. The first will be used to test and evaluate the performance at dynamical mode, the second will be focused on the availability of the architecture. We will show how the platform intervene to maintain these quality attributes.

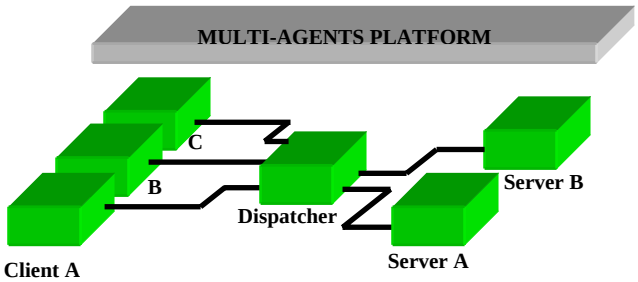


Figure 5: General functional view of architecture

The Performance Scenario

The first scenario is about the performance. The performance is the ability of a system to allocate its computational resources, to request for services in a manner that will satisfy timing requirements. The system must do so in the presence of competing requests (see Figure 6). Stimuli trigger a computation to be initiated. For the performance, the stimuli include external events such us message or user key strokes, internal events based on state changes, and click interrupts. The performance architectural decisions include various types of resources consumption. The resource types comprise for example: processors, networks, buses and memory. The resource arbitration, also know as scheduling, concerns policies for determining which of asset of pending resource requests (from entities such as processes and messages) will be served. The resource allocation concerns policies for moving resources demands in a manner that will achieve better throughput or minimize the number of necessary resources. The resources consumption is measured in terms such as execution time on processors or bandwidth for networks. The response is characterized by measurable quantities such as latency, throughput, and precedence.

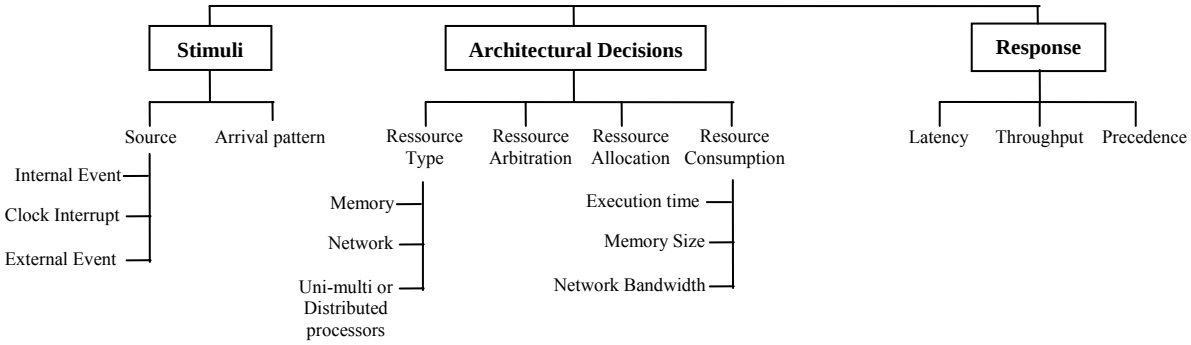


Figure 6: Performance characterization

We use external stimuli to evaluate the performance of the software, the evaluator can test the performance of the architecture via the platform by using its monitoring system. The reactive agents execute a sampling on the critical software and hardware components to collect information about the architecture. The constructed scenario monitors the resources consumption of the software like memory size, the execution time. The metrics evaluations give different values to help the architect to show the performance of its architectural decisions. The platform monitors each software and hardware component implicated in the performance of the architecture, all the information collected (by reactive agents) and transmitted (by intermediate agents) to the higher agents will be saved in its knowledge base. these information of test can presented by graphics (see Figure 7).

In addition the evaluator, can make another functionality of the monitoring which consist to parameter the limit of the performance in the decisional part of the platform, so, the platform can react to performance state of components if the values is over the constraints by using modifiability of the architecture (for example, if principal server is over its CPU charge, the platform can reorient task to the second server)

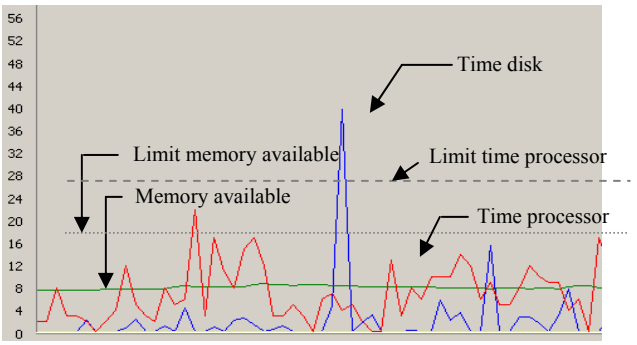


Figure 7: The monitoring of performance of the server A

The Availability Scenario

The important stimuli in an availability characterization are faults, in both hardware and software components (see Figure 8). Such faults are the events that cause systems to fail. We measure the system response by looking at measures such as reliability (the probability of not failing over period of time), mean time failure, and steady state availability.

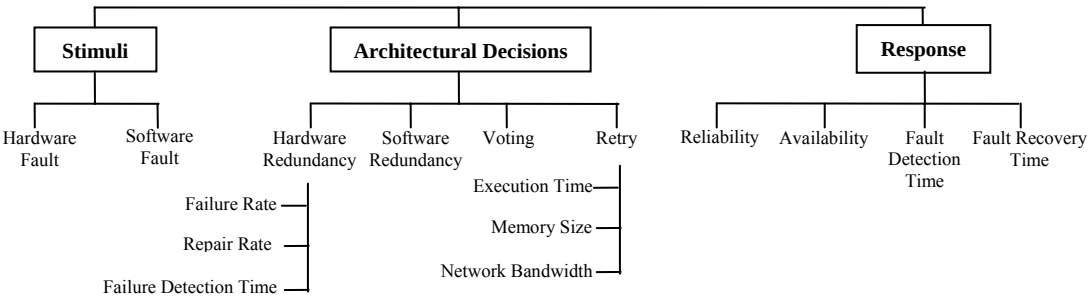
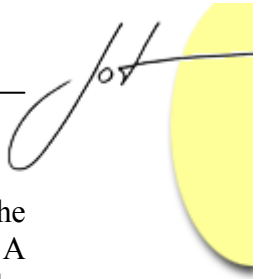


Figure 8: Availability characterization

In our scenario, we suppose that the principal server breakdown (stimuli are hardware faults), the client must continue to work on the software without interruption of its functionalities and services, the detection and the treatment processes of monitoring system follow these steps:



Step 1: The reactive agents of the platform monitor the architecture, precisely, the servers and the dispatcher. The reactive agents collect the information from the server A and send it to the intermediate agent, the information contain the abnormal of the behavior (see Figure 9.a).

Step 2: The state of the server A is analyzed by the intermediate agent, and this agent detects the exception and the name of the server failure (breakdown) and sends a message to the superior agent (see Figure 9.b). The superior agent decrypts the message and consults its knowledge base, so, it creates the planning in order to recover fault (see Figure 9.c). The platform reacts automatically when the breakdown of server A is detected.

Step 3: The planning of higher agent sends to the intermediate agent. The intermediate agent uses reactive agents in cooperative mode to achieve its goals. A reactive agent recovers the clients of breakdown server and reorients the connections to the second server (server B), another agent disables the connector between dispatcher and server A, other reactive agent parameters the dispatcher to do a unique connection to the server B. Also, the client can continue to use database without problem, the platform find solution in order to maintain the availability of software architecture.

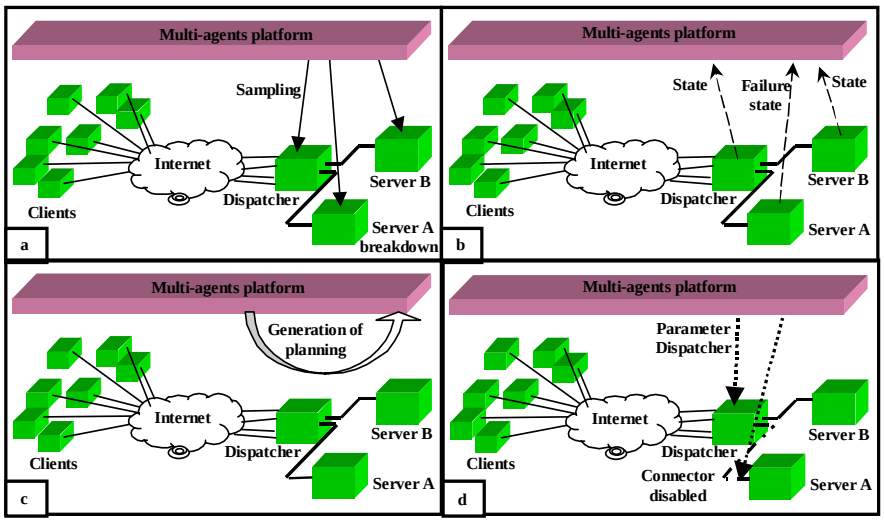


Figure 9: Monitoring system of the platform on Client/Server architecture.

5 STRENGTHS OF OUR APPROACH

- The platform takes advantage of multi-agents system like flexibility, parallelism, high level of interaction between agents.
- Autonomous monitoring of an architecture.
- Utilization of intelligent agents for reasoning part of the platform.

- Capability to test several qualities at the same time and observe the interaction and the influence of these quality attributes each other.
- The platform manages and maintains the quality attributes of the architecture.
- The platform gives metrics results of the quality tested, represented by graph or values.
- Dynamic reconfiguration of the architecture.
- Overview of the architecture and its configuration.
- The platform can produce the report of different states of components and the modifiability of the architecture during the application.
- Possibility to increase the ability of the platform, by adding new functionality in its knowledge base.
- Portability of the multi-agents system.

6 CONCLUSION

The right architecture is the first step to success. The wrong architecture will lead to calamity. We can identify causal connections between design decisions made in the architecture and the qualities/properties that result downstream in the system. This means that it is possible to evaluate an architecture, to analyze architectural decisions. The architecture then becomes the basis of systematic development and evolution of software/hardware systems. It is clear that a new architecture that permits the dynamism reconfiguration while ensuring the use of software in multiple contexts and the ability of software to support evolution and changing requirements in various contexts are needed. This paper presents a new platform based multi-agents which monitors the global architecture of a system and manages its provided quality attributes (in our case, performance and availability). It will achieve its functional and non functional requirements and evaluate and manage changes in such architecture dynamically at the execution time. In this paper we have developed our generic platform and we have applied and implemented it on the Client/Server architecture. We have showed by some scenarios the dynamic reconfigurations for the improvement of the quality attributes. Our approach can be extended to deal with other architectural “non-functional” quality attributes in the context of developing complex and reliable systems, and to support major method of evaluation based-scenarios.

REFERENCES

[67] Shaw, M., Garlan, D., Software Architecture. Perspectives on Emerging Discipline, Prentice-Hall, Inc. Upper Saddle River, New Jersey, 1996.

[2] Dias, M.S., and Richardson, D.J., The role of Event Description in Description in Architecting Dependable Systems. In proceeding of WADS: Workshop on Architecting Dependable Systems. Orlando, USA 25 May 2002.

[3] Shroeder, B., On-line monitoring, IEEE Computer, vol. 28, n. 6, June 1995. pp. 72-77.

[4] Snodgrass, R., "A Relation approach to monitoring complex systems", ACM Trans. Computer Systems, vol. 6, n. 2, May 1988, pp. 156-196.

[5] Paul Clements, Rick Kazman and Mark Klein, Evaluating Software Architectures: Method and Case Studies, Addision Wesley, 2002.

[6] Rick Kazman, Len Bass, Gregory Abowd, and Mike Webb, SAAM: A Method for Analyzing the Properties Software Architecture, Proceeding of the 16th International Conference on Software Engineering, Sorrento, Italy, May 1994, pp. 81-90.

[7] ATAM: Method for architecture evaluation: ATAM – Architecture Tradde-off Analysis Method report.

[8] Lassing, Nico, Ph.D. Thesis, architecture-level Modifiability analysis, Ph.D. thesis, Free University Amsterdam, February 2002.

[9] Bengtsson, PerOlof, Ph.D. Thesis, architecture level modifiability analysis, Department of software Engineering and Computer Science, Belkinge Institute of technology, Sweden 2002.

[10] Stephan Kurpjuweit, Ph.D. Thesis. A family of tools to integrate software architecture analysis and design. Final draft version, to be published 2002

[11] R. Kazman, G. Abowd, L. Bass, P. Clement, Scenario-based Analysis of Software Architecture, IEEE Software 13, 6 (November 1996): 47-55.

[12] Len Bass, Paul Clements, Rick Kazman, Linda Northrop, Amy Zaremski, Recommended Best Indistrial Pratice for Software Architecture Evaluation, Technical Report, Software Engineering Institute, January 1997

About the authors

Amar Ramdane-Cherif received his Ph.D. degree from Pierre and Marie university of Paris in 1998 in neural networks and IA optimization for robotic applications. Since 2000, he has been associate Professor in the laboratory PRISM, University of Versailles, Saint-Quentin en Yvelines, France. His main current research interests include: Software architecture and formal specification, dynamic architecture, architectural quality attributes, architectural styles and design patterns, E-mail: mailto:rca@prism.uvsq.fr

Samir Benarif received his BS degree at the University of Science and Technology Houari Boumediene (Algiers, Algeria) in electrical engineering and his MS degree at the university of Versailles, Saint-Quentin en Yvelines, France, respectively in 2000 and 2002. Since 2003 he has been studying for his Ph.D. at PRISM laboratory, University of Versailles, France. His investigations and field interests concern, dynamic architecture, architectural quality attributes, architectural styles and design patterns.

Nicole Lévy is a professor at the University of Versailles, Saint-Quentin en Yvelines, France. She holds a doctoral degree from the Nancy University. She is Director of the ISTY and a research staff of the PRISM Laboratory, Versailles, where she coordinates the SFAL (Spécification Formelle et Architecture Logicielle) research group. Her main research interests are formal and semiformal development methods, formalization of styles and architectural patterns.