Horizon Project

ANR call for proposals number ANR-08-VERS-010
FINEP settlement number 1655/08

Horizon - A New Horizon for Internet

WP1 - TASK 1.1: State-of-the-art in Context Aware Technologies

Report

(ANNEX B)

Institutions

GTA-COPPE/UFRJ
PUC-Rio
UNICAMP
Netcenter Informática ltda
LIP6 Université Pierre et Marie Curie
Telecom SudParis
Devoteam
Ginkgo Networks
VirtuOR

Contents

List of Figures

Chapter 1

Introduction

The goal of the Horizon Project is to conceive and test a new architecture for a post-IP environment. This post-IP architecture employs network virtualization, which includes a piloting system. The design of such pilot system is based on multi-agent systems to guide the system towards intelligent decisions.

The central idea of this project is to develop an environment where each element of the system contributes with information used to automatically update the control algorithms to reflect the changes in the environment that influence the value of the network parameters. For that, the piloting system must be "autonomous". To make decisions for enabling the system with such autonomy, we surveyed the state of the art on autonomous systems, multi-agent systems with support to piloting system, as well as some development platforms that offer such features.

This document presents the state of the art on multi-agent systems and piloting system. Chapter 2 shows the characteristics of autonomous systems. Chapter 3 introduces multi-agent systems with emphasis on its specific features to support piloting systems. Chapter 4 presents agent platforms that meet the requirements of the project, and Chapter 5 presents the conclusions.

Chapter 2

Autonomic Systems

An autonomic computing system can be described as a system that senses its operating environment, and models its behaviour in that environment. Moreover, it takes actions to change either the environment or its own behaviour. A goal-oriented system is an autonomic system that operates independently and that achieves its goals by itself without intervention, even if external environmental changes. An autonomic system has the properties of self-configuration, self-healing, self-optimization and self-protection [1]. In summary, the main characteristics of autonomic systems are autonomy and spontaneity.

2.1 Characteristics of Autonomic Systems

The concept of "The Autonomic Computing" was proposed by IBM [1], that defined four primary functionalities of autonomic computing: self-configuration, self-healing, self-optimization, and self-protection, which are described next:

- **Self-configuration** is the capacity of adapting itself to dynamically changing environments. When a standalone component is introduced, it integrates seamlessly into its surroundings environment and the rest of the system automatically adapts itself to the presence of the new components. The system has the capacity to self-adjust, and the automatic configuration of components follows high-level policies;

- **Self-healing** is the ability to discover, diagnose, and act to prevent disruptions. The system must be able to maintain all its features, possibly in a degraded mode, until all the needed resources be found. It should maintain a base of knowledge about the configuration system,

to allow diagnostic reasoning and to analyze system logs (or logs from other systems) to identify failure;

- **Self-optimization** is the capacity to tune resources and to balance workload so that the use of resources can be measured. In this way, components should be structured to improve performance and efficiency which demands the ability to monitor the environment, experiment new options, and learn to improve choices for performance optimization;

- **Self-protection** is the ability to anticipate, detect, identify, and protect against threats. An autonomous system must detect these situations and avoid disruption of usage. Such ability calls for the setting up of mechanisms and architecture for detection and protection of all network elements.

2.2 Architecture and Operation of Autonomic Systems

An autonomic computing system is made of a connected set of autonomic elements. Each element must include sensors and effectors [2]. A control loop composed of monitoring behaviour through sensors, comparison with expectations, decide making and their execution is proposed to achieve the system goal.

2.2.1 Architecture of Autonomic Elements

The Figure 2.1 shows an architecture of an autonomic element [2].

An autonomic element is composed of managed components, as well as an autonomic manager which contains several components:

- **The internal monitor** observes the state of the "managed component" and communicates the information to the component self monitor;

- **The external monitor** observes the state of the environment and communicates this state information from environment to the self monitor component;

- **The self-monitor** assesses the state of the managed component, compares it with the expected state stored in a knowledge base, so that to assess the deviation of the state of managed component can be assessed;

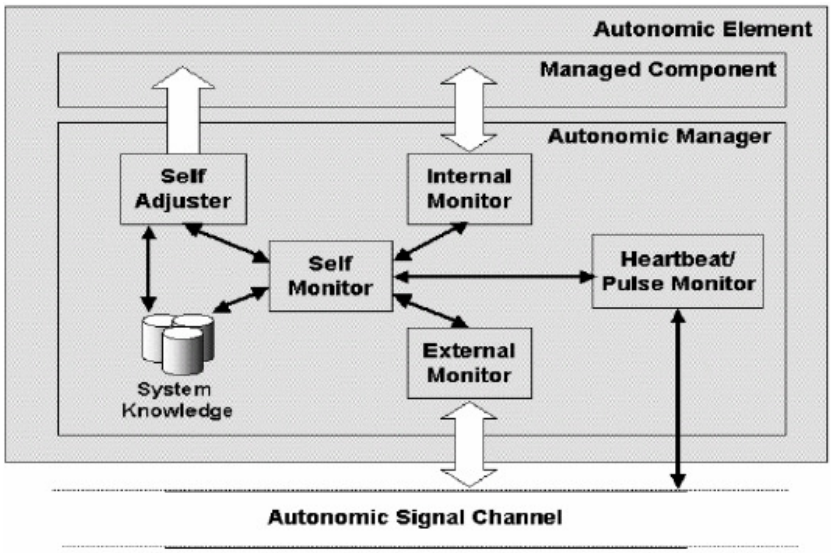

Figure 2.1: Architecture of an Autonomic Element.

- **The self-adjuster** allows the adjustment of the status of managed component;
- **The heartbeat monitor** summarizes the state of the autonomic entity and communicates with other entities responsible for the autonomic state control.

The Figure 2.1 shows that an autonomic element is composed of a managed component and a correspondent autonomic manager. The autonomic manager implements the required self-monitoring and self-adjusting capabilities. An internal monitor observes the state of the managed component and passes this information to the self-monitor for evaluation and action. The measured state is compared with the expected state held in a knowledge base. Undesirable deviations are reported to the self adjuster for action, which may result in changes to the managed component. Similarly, an external monitor observes the state of the environment via an autonomic signal channel and it may trigger internal changes. The signal channel links it to other autonomic managers. The heartbeat or pulse monitor provides a summary of the state of an autonomic element to other autonomic elements responsible for monitoring that state [2].

In summary, an autonomic system composed of a set of autonomic entities operates in a control loop which ensures the "self" properties.

2.2.2 Autonomic Control Loop

Autonomic systems work as control loops (Figure 2.2) [3]. These systems collect information from a variety of sources including traditional network sensors and reporting streams as well as higher-level device and user context. The information is analyzed to construct a model of the evolving state of the network and its services, so that adaptation decisions can be made. These decisions are actuated through the network and will potentially be reported to users or administrators. The impact of these decisions can be then collected to inform the next control cycle.

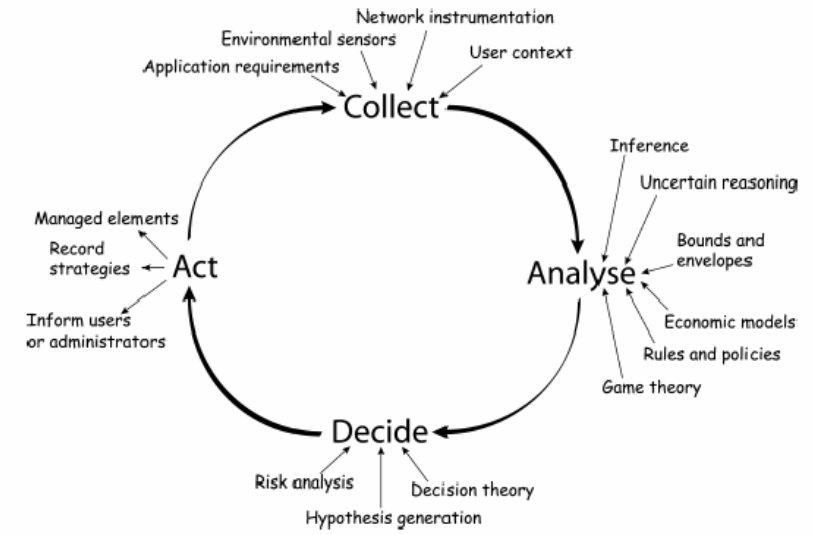

Figure 2.2: Autonomic control loop.

Several solutions have been proposed for the implementation of these systems, including intelligent agents. We present in the next chapter information about multi-agent systems.

Chapter 3

Multi-Agent Systems (MAS) Seen as Piloting Systems

The correspondence between a population of cooperating autonomous agents and the characteristics of telecommunication networks is the main motivation for using multi agent system (MAS) in telecommunication networks. Researchers on MAS have searched for challenging study cases, while the network community searches new solutions to some of traditional problems, such as provisioning of QoS and avoidance of manual network configuration [4].

Agents are usually classified in two categories: "cognitive" and "reactive"; the difference between them involves the design methods and their usage. Ferber [5] summarizes such difference by the question: Should we understand agents as entities already intelligent that are capable of solving certain problems by themselves, or should we assimilate them as very simple reactive beings that act directly to environmental changes?

3.1 Definition of Agents

There is no a universally accepted definition of the concept of "agent". One of the proposed definitions [5] [6] is that an agent is a physical or virtual entity which:

- Is capable of acting in an environment;
- Can communicate directly with other agents;
- Is driven by a set of trends (in terms of its objectives, or a function of satisfaction and even survival, that it seeks to optimize);

- Possesses its own resources;
- Is capable of perceiving (in a limited way) its environment;
- Has only a partial representation of this environment (and possibly even none);
- Possesses skills and offers services;
- Can possibly reproduce itself;
- Tending to achieve its objectives, by taking into account the available resources and skills. It perceives, represents and communicates with other agents.

Thus, an agent can be seen as an entity capable of thinking and acting independently of its environment in order to satisfy objectives which were preset (by itself or by an external entity). Some characteristics of agents are:

- **Ubiquity**, which is the capacity of sophistication and deployment of an agent-based process;
- **Interconnection**, which plays an essential role in the design of multi-agent systems;
- **Intelligence**, which can be measured by the complexity of tasks that are automated or delegated without human intervention.

3.2 Characteristics of Agents

An agent can be situated, autonomous, proactive, reactive, or social, as described bellow:

- **Situated**: the agent is able to act on its environment based on sensory inputs it receives from the environment;
- **Autonomous**: an agent should able to act without the intervention of others (human or agent) and control its own actions and its internal state;
- **Proactive**: an agent must exhibit a proactive behaviour, while being able to take initiatives at the right moments;

- **Reactive**: an agent must be able to perceive its environment and to formulate responses in the required time; and
- **Social**: an agent must be able to interact with other agents (Software or human) to perform its tasks or to assist other agents to perform their tasks.

3.3 Cognitive Agents

There is no unique definition of "cognitive agents". In [5], there are references to a 'cognitive school'. Researchers that follow this "school" are interested in agents which can make plans for their behaviors. A cognitive agent has a knowledge base including all information and know-how necessary to carry out its task and the interactions with other agents and with its environment. In other words, cognitive agents can be defined as 'intentional'. They possess goals and explicit plans to accomplish these goals. Briot [7] discusses about deliberative agents, which are equivalent to the cognitive agents defined in [5].

3.4 Reactive Agents

While cognitive agents can build plans for their behaviours, the reactive agents just have reflexes. Reactive agents are defined as a special type of agent, being cognitive agents the general case. In [6], reactive agents are defined as those which react without reference to their history. In other words, "they simply respond directly to their environment". In [7], the authors describe the architecture of reactive agents as opposed to cognitive agents. Similarly to the architecture proposed in [6], the architecture discussed in [7] defines reactive agents as those which make decisions based only on the information captured in the current execution time.

3.5 Multi-Agent Systems (MAS)

A multi-agent system (MAS) consists of a set of computer processes acting at a certain time, sharing common resources and communicating with each other. The key point of multi-agent systems is the formalization of coordination among agents. The main issues involving agents are:

- **Decisions**: what are the mechanisms of the officer's decision? What is the relationship between perceptions, representations and actions of

agents? How do they break down their goals and tasks? How do they construct representations?

- **Control**: what are the relationships between agents? How are they coordinated? This coordination can be described as cooperation to accomplish a common task or as a negotiation between agents with different interests.

- **Communication**: what kind of message they send to each other? Which syntax these messages follow? Different protocols are offered depending on the type of coordination between agents.

The multi-agent systems have applications in the field of artificial intelligence, which may reduce the complexity of the problem dividing it into subgroups. An intelligent agent is associated with each subgroup and the exchange of information between agents is used to perform coordination [5]. This is known as distributed artificial intelligence. Multi-agents are especially useful in telecommunications, such as in electronic commerce, but can also be used in other applications such as optimization of transportation systems and robotics.

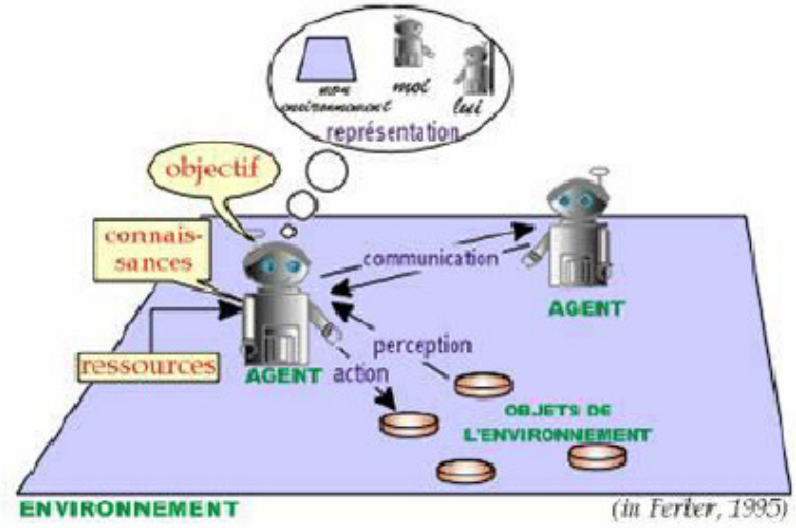

Figure 3.1: Multi-agent system.

[5] defines a multi-agent system (Figure 3.1) composed of the following:

- **An environment E**;

- **A set of objects O**. These objects are passive and can be associated with a position in environment E. They can be created, collected, modified and destroyed by agents;
- **A set A of agents**, which represent entities active in the system;
- **A set R of relations** that bind objects (and therefore agents) between them;
- **A set O of operations** that allows which agents of the set A, collect, produce, consume, transform and manipulate objects from O; and
- **Operators** responsible for representing the application of these operations and world reaction to this attempt to change.

A multi-agent system is therefore composed of several agents that operate in an environment and interact with each other. These agents are able to perceive and act on objects detectable in this environment. Their collaboration allows them to act according to their respective goals and motivations. A multi-agent system is composed of a set of agents which interact among themselves Each agent is composed of:

- **Collection Component**, which allows agents to collect information on their environment;
- **Interaction Component**, which allows agents to interact with other agents;
- **Decision Component**, which allows agents to take decisions based on their perceptions;
- **Execution Component**, which allows agents to perform its decisions.

Chapter 4

Options to Build the Autonomic Platform

Future post-IP architecture should be context-aware. So, the first work package involves the task of defining the deployment of a context-aware infrastructure. The platform to finish such functionality will be included in the network elements. Physical and logical sensors (software entity, network components, and software agents) collect context information related to the presence, location, identity and profile of users and services. A typical context-aware software involves the localization of services and users, the call of services according to user behaviour, the provisioning of information for service composition, facilitation of ad hoc communication among users, and adaptation of QoS to the changing environment. The objective is to explore two types of context aware infrastructures and to chose the best way to introduce intelligence in the Horizon platform composed of a Knowledge and a Piloting planes. More precisely, the Horizon platform will adopt a multi-agent system to offer some intelligence. The multi-agent system is formed by agents situated in all network equipment (common to all virtual instances) [8]. In this Chapter, we describe what could be an autonomic platform. For this, we analyzed three platforms that could be useful in the Horizon Project: the Ginkgo platform, the Dimax platform and the JADE platform.

4.1 Ginkgo

The Ginkgo's technology provides support to Autonomic Networking applications by employing Intelligent Agents, distributed in Network Equipments (NE) across the network. Ginkgo Intelligent Agents play a dual role in autonomic networking applications: by feeding, in real-time, the Knowl-

edge Plane with the information required by the application and by exploiting the distributed knowledge in the Knowledge Plane to manage in real-time network control mechanisms [9].

4.1.1 Situated View of the Ginkgo Agents and Knowledge Plane

In the Ginkgo model, the Knowledge Plane is distributed among Ginkgo agents as a set of "Situated Views". Each Situated View represents the knowledge of an individual agent regarding the situation of the network in its neighbourhood which is supposedly more important to the agent than the situation in remote locations. Furthermore, updating knowledge in individual Situated Views can be more easily done in real-time with limited amount of control traffic. Ginkgo agents working together maintain an always-up-to-date collective distributed knowledge of the overall network situation, as illustrated in the Figure 4.1 [9].

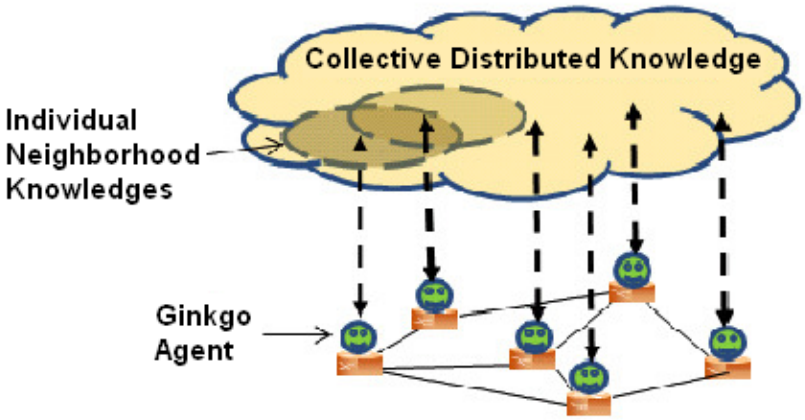

Figure 4.1: Ginkgo agents and the Knowledge Plane.

4.1.2 The Ginkgo Agent Architecture

Ginkgo Agents are made of three main types of building blocks (Figure 4.2):

- **The Situated View Knowledge Base**, which is dedicated to store the structured knowledge of the Situated View of the agents;

- **The Behaviours**, which are autonomic software components permanently adapting themselves to the environment changes. Each of these Behaviours can be considered as a specialized function with some expert capabilities. Each behaviour is essentially a sense→decide→act loop in charge of a control function;
- **The Dynamic Planner**, which is in charge of orchestrating the Behaviours. The Dynamic Planner follows a Policy provided by the user indicating how the behaviours should take into account the dynamic changes occurring in the environment.

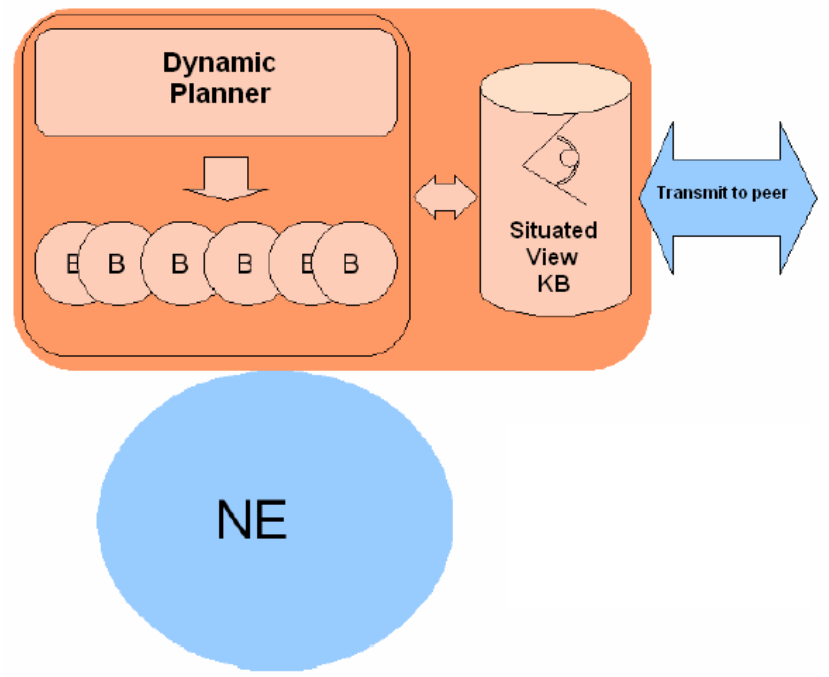

Figure 4.2: Ginkgo agents architecture.

Each of these Behaviours ("B" in Figure 4.2) can be executing some expert function. Typical functions of Behaviours are:

- Production of knowledge for the Situated View in cooperation with other Agents;
- Reasoning individually or collectively to evaluate the situation and to decide on the application of an appropriate action, e.g. a Behaviour can simply be in charge of computing the available bandwidth on the NE.

It can also regularly perform complex diagnoses or it can be dedicated to automatic recognition of specific network conditions;

- Action onto the NE parameters, e.g. a Behaviour can tune QoS parameters in a DiffServ context;
- Uploading of useful information to a network management system, e.g., a Behaviour can upload a synthetic alarm obtained from the observations of elementary ones.

Behaviours have access to the Situated View which operates within each Agent as a whiteboard shared among the Behaviours of the Agent.

The activation, dynamic parametrization and scheduling of Behaviours within an Agent is performed by the **Dynamic Planner**. The Dynamic Planner decides which Behaviours have to be active, when they have to be active and with which parameters. The Dynamic Planner detects changes in the Situated View and occurrence of external/internal events. It orchestrates the reaction of the Agent to changes the network environment. To do so, the Dynamic Planner follows a rule-based policy expressed in a simple and compact form.

The **Situated View** of each agent is a structured knowledge base representing the environment of the agent. It contains knowledge elements collected locally by the agent as well as knowledge obtained from its peers. The Situated View is updated on a periodical basis and it is used to adapt the Behaviours to changes occurring in the network and to take real-time decisions. An automatic mechanism mirrors the Situated View to the appropriate peers; the knowledge is reflected in the Situated View of the peer agents. The rate and range of this mechanism can be tuned according to the nature of the knowledge. The Situated View is organized following an ontology-based model, which helps to build well structured applications as well as to inter-operate with other systems.

4.2 DimaX

DimaX is a fault-tolerant multi-agent platform which offers several services like naming, fault detection and recovery. To make multi-agent system (MAS) robust, DimaX uses replication techniques. Moreover, DimaX provides developers with libraries of reusable components for building MAS. DimaX presents some interesting features like robustness and reusability.

4.2.1 DimaX Services

DimaX [10] is the result of the integration of a multi-agent platform (named DIMA) and a fault tolerance framework (named DARX). Figure 4.3 gives an overview of DimaX and its main components and services. DimaX is designed in three levels: system (i.e., DARX middleware), application (i.e., agents) and monitoring. At the application level, DIMA provides a set of libraries to build multi-agent applications. Moreover, DARX provides the mechanisms necessary for distributing, and replicating agents as services. DimaX server provides the following services: naming, fault detection, observation and replication.

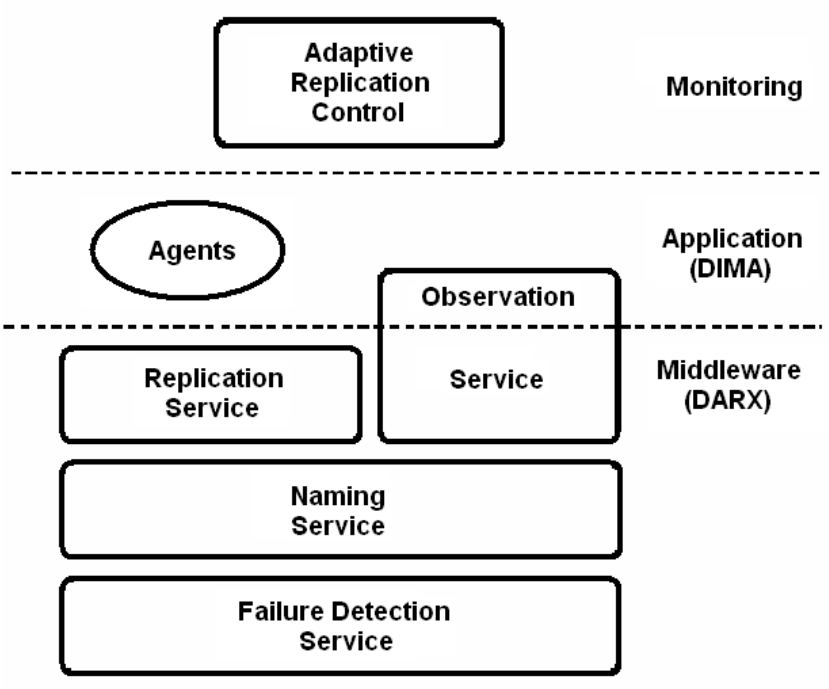

Figure 4.3: Overview of DimaX.

The **Naming Service** maintains a list (i.e., white pages) of all the agents within its administration domain. When an agent is created, it is registered at both the DimaX server and the naming server. The **Failure Detection Service** (from DARX) is based on the heartbeat technique; a process sends an `I am alive` message to other processes for informing that it is safe. When a server detects a failure of another DimaX server, its naming module removes all the replicated agents hosted at the faulty server from the list and replaces them by their replicas located on other hosts. The replacement is initiated by the failure notification.

The functionalities of the **Observation Service** are fundamental for controlling replication. An observation module collects data at two levels. The system level collects data about the execution environment of the MAS like CPU time and mean time between failures, while the application level collects information about its dynamic characteristics, such as the interaction events among agents (e.g., the sent and received messages). The observation service relies on a reactive-agents organization (named host monitors). These agents collect and process the observation data to compute local information, such as the number of exchanged messages between two agents during a given period.

DimaX uses replication mechanisms (**Replication Service**) to avoid failures of multi-agent systems. The Replication service enables to run multi-agent systems without interruption regardless of the failures. A replicated agent is an entity that possesses two or more copies of its behaviour (or replicas) in different hosts. There are two main types of replication protocols: active and passive. In active replication, all replicas process all input messages concurrently, while in passive replication, only one of the replicas processes all input messages and periodically transmits its current state to the other replicas so that consistencies are maintained. DimaX offers several libraries and mechanisms to facilitate the design and implementation of fault-tolerant multi-agent systems. Following, some characteristics of DimaX agents are briefly described.

4.2.2 DIMA Agent Behaviours

DIMA is a Java multi-agent platform. Its kernel is a framework of proactive components which represent autonomous and proactive entities. A simple DIMA agent architecture consists of: a proactive component, an agent engine, and a communication component. The proactive component (the `AgentBehaviour` class) represents the agent behaviour. This proactive component includes a decision component to select appropriate actions. For instance, a finite machine state or a rule-based system could be used to describe the decision process. The selected actions can include sending messages and a communication component is used to send and deliver messages. An Agent Engine is provided to launch and support the agent activity.

4.2.3 DarX Tasks

DARX is a framework to design reliable distributed applications which include a set of distributed communicating entities (named DarX tasks). It includes transparent replication management. While the application deals

with tasks, DARX handles replication groups. Each of these groups consists of software entities (the replicas) which represents the same DarX task. In DARX, a DarX task can be replicated several times and with different replication strategies.

4.2.4 Fault-Tolerant Agents

A fault-tolerant agent (called DimaX agent) is an agent built on our DimaX fault-tolerant multi-agent platform. Each DimaX agent has the structure of a DarX Task. However, the DarX Task is not autonomous. To make it autonomous, we encapsulate the DIMA agent behaviour into the DarX Task. This agent architecture enables the replicate of the agent. As the DARX middleware and the DIMA platform provide mechanisms for execution control, communication and naming at different levels, their integration requires a set of additional components. This set calls, transparently, DARX services (e.g., replication, naming) when executing multi-agent applications developed with DIMA. At the application level, any code modification is required. It controls the execution of agents built under DimaX and it offers a communication interface between remote agents, through DimaX servers.

4.3 JADE

JADE (Java Agent Development Framework) is a software environment to build agent systems for the management of networked information resources in compliance with the FIPA specifications for interoperable multi-agent systems [11]. JADE provides a middleware for the development and execution of agent-based applications which can seamless work and interoperate both in wired and wireless environment. Moreover, JADE supports the development of multi-agent systems based on predefined programmable and extensible agent model as well as on a set of management and testing tools [12]. A JADE environment can evolve dynamically, since agents can appear and disappear in the system according to the needs and the requirements of the application environment. Communication between peers, regardless of whether they are running in the wireless or wired network, is completely symmetric since each peer is able to play both the initiator and the responder role. JADE is fully developed in Java and it is based of the following driving principles [13]:

- **Interoperability**: JADE is compliant with the FIPA specifications [11]. As a consequence, JADE agents can interoperate with other agents, provided that they comply with the same standard;

- **Uniformity and portability**: JADE provides a homogeneous set of APIs that are independent from the underlying network and Java version. Actually, the JADE run-time provides the same APIs both for the J2EE, J2SE and J2ME environment. In principle, it is possible to application developers to decide the Java run-time environment at deployment time;

- **Easy to use**: The complexity of the middleware is hidden behind a simple and intuitive set of APIs;

- **Pay-as-you-go philosophy**: Programmers do not need to use all the features provided by the middleware. Features that are not used do not require programmers to know anything about them, neither add any computational overhead.

JADE offers the following list of features to the agent programmer:

- FIPA-compliant Agent Platform, which includes the AMS (Agent Management System), the default DF (Directory Facilitator), and the ACC (Agent Communication Channel);

- Distributed agent platform. The agent platform can be split on several hosts. Only one Java application, and therefore only one Java Virtual Machine, is executed on each host. Agents are implemented as one Java thread and Java events are used for effective and lightweight communication between agents on the same host. Moreover, parallel tasks can be executed by one agent, and JADE schedules these tasks in a cooperative way;

- A number of FIPA-compliant additional DFs can be started at run time in order to build multi-domain environments, where a domain is a logical set of agents, whose services are advertised through a common facilitator;

- Java API to send/receive messages to/from other agents; ACL messages are represented as ordinary Java objects;

- Lightweight transport of ACL messages inside the same agent platform, as messages are transferred encoded as Java objects, rather than strings, in order to avoid marshalling and unmarshalling procedures;

- Library to manage user-defined ontology and content languages;

- Graphical user interface to manage several agents and agent platforms from the same agent. The activity of each platform can be monitored and logged. All life cycle operations on agents can be performed through this administrative GUI.

4.3.1 JADE Architecture

JADE includes both the libraries required to develop application agents and the run-time environment that provides the basic services that must be active on the device before agents can be executed. Each instance of the JADE run-time is called container (since it "contains" agents). The set of all containers is called platform (Figure 4.4) and provides a homogeneous layer, that hides from agents and also from application developers the complexity and the diversity of the underlying layers (hardware, operating systems, types of network, JVM) [13].

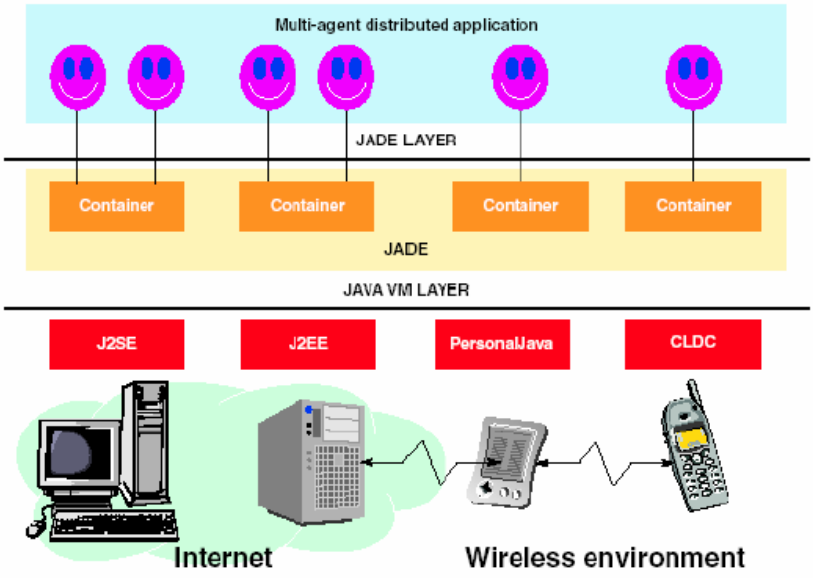

Figure 4.4: JADE architectural model.

4.3.2 Behaviours to Build Complex Agents

The developer implementing an agent must extend JADE `Agent` class and implement agent specific tasks by writing one or more `Behaviour` subclasses. User defined agents inherit from their superclass the capability of registering and deregistering with their platform and a basic set of methods (e.g. send

and receive Agent Communication Language messages, use standard interaction protocols, register with several domains). Moreover, user agents inherit from their `Agent` superclass two methods: `addBehaviour(Behaviour)` and `removeBehaviour(Behaviour)`, to manage the behaviour list of the agent [14].

JADE contains established behaviours for the most common tasks in agent programming, such as sending and receiving messages and structuring complex tasks as aggregations of simpler ones. For example, JADE offers a so-called `JessBehaviour` that allows full integration with JESS [15], a scripting environment for rule programming offering an engine using the `Rete` algorithm to process rules.

4.3.3 JADE Tools for Platform Management and Monitoring

Beyond a runtime library, JADE offers some tools to manage the running agent platform and to monitor and debug agent societies; all these tools are implemented as FIPA agents themselves, and they require no special support to perform their tasks [14]. Examples of JADE tools are:

- **Remote Monitoring Agent (RMA)**: The general management console for a JADE agent platform. The RMA acquires the information about the platform and executes the GUI commands to modify the status of the platform;
- **Directory Facilitator (DF) GUI**: The DF agent also has a GUI, with which it can be administered, configuring its advertised agents and services.
- **The Dummy Agent** is a simple tool for inspecting message exchanges among agents, facilitating validation of agent message exchange patterns and interactive testing of an agent.
- **The Sniffer Agent** allows to track messages exchanged in a JADE agent platform: every message directed to or coming from a chosen agent or group is tracked and displayed in the sniffer window, using a notation similar to UML Sequence Diagrams.

The JADE is an interesting option that offers interoperability through the FIPA specification. It offers a friendly and robust support for the development of multi-agent systems that meets the requirements of the Horizon Project.

Chapter 5

Conclusions

This document presents a report on the state of the art on autonomic systems as well as multi-agent systems. These systems are well adapted to complex environments and are composed of highly dynamic networks that implement the concept of self-piloting systems. We gave an overview of three platforms for building agents that can be useful for the development of the Horizon Project.

Acknowledgement

We would like to thank Carlos Roberto Senna, Daniel Macêdo Batista, Edmundo Roberto Mauro Madeira, and Nelson Luis Saldanha da Fonseca for their work in improving the final version of this report.

Bibliography

[1] IBM, "An Architectural Blueprint for Autonomic Computing," Jun 2006. White Paper. 4th Edition.

[2] R. Sterritt and D. Bustard, "Towards an Autonomic Computing Environment," in *DEXA '03: Proceedings of the 14th International Workshop on Database and Expert Systems Applications*, p. 699, IEEE Computer Society, 2003.

[3] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli, "A Survey of Autonomic Communications," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 1, no. 2, pp. 223–259, Dec 2006.

[4] T. Bullot, R. Khatoun, L. Hugues, D. Gaïti, and L. Merghem-Boulahia, "A Situatedness-Based Knowledge Plane for Autonomic Networking," *International Journal of Network Management*, vol. 18, no. 2, pp. 171–193, Mar 2008.

[5] J. Ferber, *Les Systèmes Multi-Agents: Vers Une Intelligence Collective*. Dunod, 1995.

[6] M. Wooldridge, *An Introduction to Multi-Agent Systems*. John Wiley & Sons, 2002.

[7] J. Briot and Y. Demazeau, eds., *Principes et Architecture des Systèmes Multi-Agents*. Hermes, 2001.

[8] Horizon Project, "Horizon Project: A New Horizon to The Internet," 2010. `http://www.gta.ufrj.br/horizon/`. Accessed at Apr 2, 2010.

[9] Ginkgo Networks, "White Paper – Ginkgo Distributed Network Piloting System," Sep 2008. `http://www.ginkgo-networks.com/IMG/pdf/WP_Ginkgo_DNPS_v1_1.pdf`. Accessed at Apr 2, 2010.

[10] N. Faci, Z. Guessoum, and O. Marin, "DimaX: A Fault-Tolerant Multi-Agent Platform," in *SELMAS '06: Proceedings of the 2006 International Workshop on Software Engineering for Large-Scale Multi-Agent Systems*, (New York, NY, USA), pp. 13–20, ACM, 2006.

[11] FIPA, "The Foundation for Intelligent Physical Agents," Oct 2010. `http://www.fipa.org`. Accessed at Apr 2, 2010.

[12] R. H. Bordini, M. Dastani, J. Dix, and A. E. F. Seghrouchni, eds., *Multi-Agent Programming Languages, Platforms and Applications*. Springer, 2005.

[13] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa, "JADE A White Paper," Sep 2003. Volume 3, n. 3, `http://jade.cselt.it/papers/2003/WhitePaperJADEEXP.pdf`. Accessed at Apr 2, 2010.

[14] F. Bellifemine, A. Poggi, and G. Rimassa, "Developing Multi-Agent Systems with JADE," *Intelligent Agents VII Agent Theories Architectures and Languages*, vol. 1986, pp. 89–103, 2001.

[15] JESS, "Jess, the Rule Engine for the Java Platform," Nov 2008. `http://herzberg.ca.sandia.gov/jess/`. Accessed at Apr 2, 2010.