MULTIMODAL INTERACTION SEMANTIC ARCHITECTURE FOR AMBIENT INTELLIGENCE

Sébastien Dourlens

MULTIMODAL INTERACTION SEMANTIC ARCHITECTURE FOR AMBIENT INTELLIGENCE

DOCTORAL THESIS OF VERSAILLES SAINT QUENTIN UNIVERSITY - LABORATOIRE D'INGENIERIE ET SYSTEMES DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

By

Sébastien Dourlens

Defended on 14 May 2012

Thesis Committee

Reviewer:	Professor Alain ABRAN, Departement of Software Engineering and Information Techology, Ecole de Technologie Supérieure, Université du Québec, Canada
Reviewer:	Professor Flavio OQUENDO, Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA), Université de Bretagne Sud, France
Examiner:	Associate Professor Yasuhisa HIRATA, Departement of Bioengineering and Robotics, Tohoku University, Japan
Examiner:	Professor Nicole LEVY, Centre d'Etude et De Recherche en Informatique et Communications (CEDRIC), Conservatoire National des Arts et Métiers (CNAM), France
Examiner:	Associate Professor Frédéric MIGEON, Institut de Recherche en informatique de Toulouse (IRIT), Université Paul Sabatier, France
Examiner:	Researcher and Expert Senior Jean-Denis MULLER, Commissariat à l'Energie Atomique (CEA), France
Thesis Director:	Professor Amar RAMDANE-CHERIF, Laboratoire d'Ingénierie des Systèmes de Versailles (LISV), Université de Versailles Saint Quentin, France

ABSTRACT

There still exist many fields in which ways are to be explored to improve the human-system interaction. These systems must have the capability to take advantage of the environment in order to improve interaction. This extends the capabilities of system (machine or robot) to better reach natural language used by human beings. We propose a methodology to solve the multimodal interaction problem adapted to several contexts by defining and modelling a distributed architecture relying on W3C standards and web services (semantic agents and input/output services) working in ambient intelligence environment. This architecture is embedded in a multi-agent system modelling technique. In order to achieve this goal, we need to model the environment using a knowledge representation and communication language (EKRL, Ontology). The obtained semantic environment model is used in two main semantic inference processes: fusion and fission of events at different levels of abstraction. They are considered as two context-aware operations. The fusion operation interprets and understands the environment and detects the happening scenario. The multimodal fission operation interprets the scenario, divides it into elementary tasks, and executes these tasks which require the discovery, selection and composition of appropriate services in the environment to accomplish various aims. The adaptation to environmental context is based on multilevel reinforcement learning technique. The overall architecture of fusion and fission is validated under our framework (agents, services, EKRL concentrator), by developing different performance analysis on some use cases such as monitoring and assistance in daily activities at home and in the town.

Keywords:

Multimodal Interaction Architecture, Events fusion and fission, Environment Knowledge Representation Language, Semantic Agents, Services, Knowledge-Based Agents, Multi logics, Ubiquitous Computing, Ambient Intelligence, Web services.

RESUME (French Abstract)

Il existe encore de nombreux domaines dans lesquels des moyens doivent être explorés pour améliorer l'interaction homme-système. Ces systèmes doivent avoir la capacité de tirer avantage de l'environnement pour améliorer l'interaction. Et ceci afin d'étendre les capacités du système (machine ou robot) dans le but de se rapprocher du langage naturel utilisé par les êtres humains. Nous proposons une méthodologie pour résoudre le problème d'interaction multimodale adaptée aux différents contextes en définissant et modélisant une architecture distribuée qui s'appuie sur les standards du W3C et des services Web (agents sémantiques et services d'entrée / sortie) qui travaillent dans un environnement d'intelligence ambiante. Cette architecture est réalisée en utilisant le modèle des systèmes multi-agents. Afin d'atteindre cet objectif, nous avons besoin de modéliser l'environnement en utilisant un langage de représentation des connaissances et de communication (EKRL, Ontologie). Le modèle de l'environnement obtenu est utilisé dans deux principaux processus d'inférence sémantique: la fusion et la fission des événements à différents niveaux d'abstraction. Ces opérations sont sensibles au contexte. Le système de fusion interprète, comprend l'environnement et détecte le scénario qui se passe. Le système de fission interprète le scénario, le divise en tâches élémentaires et exécute les tâches qui nécessitent la découverte, la sélection et la composition de services appropriés dans l'environnement pour répondre aux différents objectifs. L'adaptation au contexte de l'environnement est basée sur la technique d'apprentissage par renforcement multi-niveaux. L'architecture globale de fusion et fission est validée et développée dans notre framework (agents, services, concentrateurs EKRL) par l'analyse de différentes performances sur des cas d'utilisation tels que la surveillance et l'assistance dans les activités quotidiennes à la maison et en ville.

TABLE OF CONTENTS

ILLUSTRATIONS

TABLES

ACKNOWLEDGEMENTS

There are a number of people to whom I owe much gratitude for assisting me, in one way or another, with this thesis.

Thanks to Professor Amar Ramdane Cherif for giving me the opportunity to undertake this work and for his wisdom and direction in supervising this thesis.

Special thanks to all the members of this doctoral thesis committee. I'm particularily grateful to Alain Abrain and Flavio Oquendo for having kindly accepted to review my thesis report. I am also indebted to Yasuhisa HIRATA, Nicole Lévy, Frédéric Migeon and Jean-Denis Muller for having accepted to examine my thesis. I would like to thank particularly Professor Nicole Lévy for her useful comments, encouragements and suggestions.

Thanks to the Director and administrative people of the Versailles Saint Quentin University.

Special thanks to the researchers of the PRISM Laboratory and the LISV Laboratory.

Thanks to my friends and colleagues for their support.

Thanks to all my family for their love.

CHAPTER 1

INTRODUCTION

« Some men see things as they are and ask why. Others dream things that never were and ask why not. » George Bernard Shaw

1.1 Motivations

Our main motivation is to enhance the multimodal interaction between any system and the human in the human environment. One goal to achieve is to adapt the concerned system to the human and its environment in the opposition to the creation of systems and environments where people must constantly learn and adapt to these systems. And in the ideal case, the system should learn from the interactions with the user. A second challenge is to obtain an architecture as inter-connected as possible and most suitable with the human environment.

The modelled architecture should be easily integrated in the human environment by following these brand new paradigms:

- *Ubiquitous network* is used for an expanding network or architecture.
- *Pervasive computing* is the trend towards increasingly ubiquitous (another name for the movement is ubiquitous computing) and connected systems in the environment, a trend being brought about by a convergence of advanced electronic - and particularly, wireless - technologies and the Internet.
- *Pervasive environment* is an environment where communicating objects recognize and locate each other automatically. The objects interact with each other without special action from the user.
- *Ambient Intelligence* is embedded in a distributed network of intelligent devices (sensors and effectors) interconnected in the environment (Ramos et al. 2008, Augusto et al. 2010). This intelligence can drive machines, monitor activities and inform users. It also moulds context-aware environment to our immediate needs, habits, preferences and orders. Ambient intelligence integrates ubiquitous network, pervasive computing, intelligent user interfaces and security.

Our main problem is to design an architecture for interaction that are able to answer these major requirements.

Until 2008, Interaction management was concerned by 3 main parts:

- *Human interacts with Human* which is a naturally way by default (Human-Human Interaction).
- *Human interacts with Machine* and the best interaction should be the as natural as possible. It is Human-Machine Interaction (HMI). And, in particular, when machine is a computer, Human-Computer Interaction is an important part of system design, evaluation and implementation. Quality of system depends on how it is represented and used by users (ergonomics). Therefore, enormous amount of attention has been put into better designs of Human computer interfaces (HCI) and user interfaces (UI). There are very few successful HMI interfaces (Dumas et al., 2009). In case of Human-Robot Interaction (HRI), which is an application of HMI, the machine is simply a robot. HRI is an important domain concerned in human robot cooperation (Fong et al., 2003; Goodrich and Schultz, 2007). In case of ambient environment, the house can also be called a robot or a machine. To avoid this need of complication, in this thesis, we will make no distinction between machine, system, robot, and parts of a system.
- *Machine interacts with Machine.* It is Machine-Machine Interaction (MMI) and the interaction is not natural but more and more efficient

with network protocols and digital communication languages. It's used in multiagents systems and complex systems to manage the communication and cooperation between system parts.

Since 2008, the new direction of research is to replace common regular methods of interaction with intelligent, adaptive, multimodal, natural methods. Ambient intelligence or ubiquitous computing which is called the third researches wave is trying to embed the technology into the human environment so to make it more natural and transparent at the same time (Karray et al. 2008).

It is this key domain where we develop systems to understand the human environment and where systems interact with the environment (users, other systems, parts of the system) in a bidirectional way. The environment integrates both machines and human. Interfaces are not necessarily a screen, a mouse or a keyboard but more multimodal inputs/outputs systems.

In this new interaction, we decided to develop a semantic architecture able to reinforce the interaction between the systems of the environment. It is much more complex because it is fully embedded in the environment and is not limited to a single system and one user. One important requirement for such architecture is to deal with the information coming from the environment and the possible actions to act and interact with this environment. In multimodal interaction, the system is based on combination of multiples modalities of interaction by simultaneous use of different input/output channels. Modalities may be based on sensors (mouse, keyboard, pen, motion tracking, haptic, pressure, taste), visual (facial expression analysis, body movement tracking, gesture recognition, gaze/eyes movement tracking) or auditive (speech recognition, speaker recognition, emotion analysis, noise, signs detection).

We can decompose this multimodal interaction architecture in 4 main operations: environment sensing, environment understanding or modelling, deciding and acting. The need for multimodal interaction systems is to enhance error avoidance and ease the error resolution, to accommodate a wider range of users, tasks and environmental situations, to cater the need of individual differences and preferences, and to permit flexible and improved use of modes including alternation and integration. Multimodal interaction has great potential in terms of application areas and thus needs extensive inter-disciplinary research for addressing issues and challenges.

The great challenge of multimodal architecture creation is to build reliable processing systems capable of analyzing and understanding the environmental meanings. This opens a number of associated issues, such as heterogeneous modalities information fusion, architectures for service composition, interaction management, machine learning for multimodal interaction, modelling languages, frameworks, etc. This thesis does not intend to cover exhaustively all the issues related to multimodal architecture design.

To summarize, this thesis is to contribute in solving issues such as large quantity of incoming events to manage from multiple sources, environment modelling, environment understanding, complexity and heterogeneity of systems, and multimodal communication. The main objective is to model and implement a new multimodal architecture for interaction in pervasive context (pervasive environment and computing) with composition and adaptation mechanisms based on a semantic approach.

We want to answer several questions:

- How to model and understand the environment, and its systems and events?
- How to design complete interaction architecture for ambient intelligence?
- How to deal with the large amount of information?
- What is the interaction protocol we will use?
- How to manage and update context?
- How can interaction architecture adapt to context change?

First of all, we will need to describe knowledge representation and how it is entangled with the systems and components of the architecture, humans included. Recent researches have been made to store low level data like events or high level information like scenario in ontology structure. There are two main advantages of this storing structure: - natural language use and - hierarchically filled concepts. These semantic ontologies have a better expression of the relations interconnecting several classes of concepts. Instances are directly stored under the class following the concepts list. We will model components of ambient intelligence to manage multimodal interaction including decision processes for a system to act in a human environment by taking in account the different possible contexts. Our semantic components will allow the development of an architecture to understand the environment and decide accordingly. Connectors such as our environment knowledge representation language, compatible and close to natural language, and W3C standards like EMMA, will allow communicating with web services controlling hardware parts. We notice that we will use human interfaces, sensors and actuators but it is not our purpose to develop or improve them because we manage information at a high level of abstraction. Secondly, we assemble the components of the architecture. The environment is a complex system composed of multiple systems in interaction. Another interesting paradigm to design architecture is multi agent systems (Ferber, 1999) where agents are simple or complex components of that architecture. And in order to develop a pure parallel architecture, where all services are independent and components are autonomous, new architectures could use web service composition. These web services are made to communicate with extended languages in the respect of W3C standards recommended for more interoperability. We will focus our research on mechanisms like modalities selection, adaptation to context and reconfiguration of the architecture. Moreover, some mechanisms are presented to manage awareness and learning. Finally, we present the architecture implementation, the measures of performance and the development of applications.

1.2 Multimodal Interaction Problem

The interaction process holds a set of problems stemming from an incomplete *a priori* knowledge about the environment, hazards, strategies of exploration, insufficient sensory information, to an inherent inaccuracy in the systems. By combining different modalities, the system can effectively reduce recognition uncertainty, thereby improving robustness while adapting the tasks to do in a dynamically changing environment. Therefore, a great simplification of the interaction processing can be accomplished, as it has been already demonstrated by several existing multimodal systems.

Multimodality allows humans to move seamlessly between different modes of interaction, from visual to voice or touch, according to changes in context or user

preferences. A system must provide multimodal interfaces, which try to integrate speech, written text, body language, gestures, eye or lip movements and other forms of communication in order to better understand the human and to communicate more effectively and naturally.

A multimodal system is an important element of a successful human-robot interaction. When systems enter the human environment and come in contact with users, they need to be able to interact with users and ease the burden of knowledge transfer from the user to the system. The system must be capable to establish a common focus of attention and be able to use and integrate spoken instructions, visual perceptions, and non-verbal clues like gestural commands. Human-robot interaction can become more intuitive as the level of flexibility in the multimodality system interface increases, for example we can combine natural language and hand gestures to interpret both complete and fragmental commands or we can use speech, posture, and object recognition to navigate a mobile robot to an object of interest. In these cases, humans and robots act as peers exchanging information in dialogue to achieve goals. At the same time, the requirements of the domain place some constraints on the choice of modalities and the degree of freedom in expressing the user's intentions. The multimodal system must capture the user's actions in speech and gesture within the domain of operation and then attempt to match them to elements in the system's domain knowledge base.

The system can identify objects and tasks via a multimodal architecture which may interpret the deictic gestures and speech inputs of the user. Since the user input can be vague, inaccurate, and often contradicting. A fusion/fission multimodal system can be extended to be an intention aware system that can be used to reduce unnecessary and often redundant instructions by being aware of what the user really wants. In addition, system can use its own hardware and others parts of the network resources to dialog, act and interact with human too.

After the management and filtering of the large amount of information, we have to understand what and how meaning can be extracted and if useful meaning is relevant and can be stored in memory for any further use. It deals with the problem of time and availability of information, and how the information must be managed now and frequently. All this process seems to consume a lot of time and computer resources (memory and processor). These events can appear in same time, more precisely, in concurrent time. It means not only one data can give a meaning but several like for example the act to say good bye to someone, two events will often appear, an arm move and a speech.

How to make the fusion to understand this composed event which permits to understand the action done? Now, modalities and knowledge representation of a situation are improved due to the expansion of the technology, and researchers work a lot on Natural Language and XML descriptions of a scene. So we can suppose meaning will be easier to extract and use to interact.

According to all these problems and evolution, we will improve the concept of fusion and fission of events for the interaction in ambient intelligence.

1.3 "Put That Here" Scenario

To illustrate the multimodal interaction problem, we will take the famous example of "Put That Here" (Bolt, 1980). The goal of this application is for a

humanoid robot to interact with a user, visually and vocally, understanding the environment where the robot is. This environment contains the robot, some objects and the human. The user shows an object to the robot, he asks the robot to take this object and show the robot where to put it. Lots of research issues must be solved like human speech and gestures recognition, gesture imitation, speech synthesis to validate the demand, location recognition, object to grasp recognition, moving the object action, and why not communicating with the object if possible. In fact, in this thesis, we don't work on creation and the measuring of sensors or actuators and their performances. We consider them well conceived by specialists even if they are uncertain. We also choose high level sensors and actuators where data and features extraction are already effectively done. For example, the Microsoft Kinect is a high level sensor able to recognize human gestures capturing several skeletons; 3D human position in a room, capture sounds using 4 microphones (Figure 1.1).

Figure 1.1 – Microsoft Kinect and skeleton capture

It is not the aim of this thesis to redevelop such economic and efficient enough devices. However, we are interested by managing events coming from sensors capturing what is happening in the environment, understanding this environment to evaluate the situation, choose and decide how to act. That's our interaction problem.

In this example, we have two different input modalities (audio and visual) produced by three sensors: Human Gesture recognition (MS Kinect), Human voice recognition (MS Speech API) and Object recognition (Video camera embedded in the Aldebaran Nao robot). They send three events: "User points an object to a location" to point where to catch or to put on, "Object is a box" once concerned object is recognized and "User speaks" to tell user is telling words "Put", "That" and "Here" or something else. The Nao robot has the necessary output modalities to move its head, catch and move the objects with a robotic arm and may also talk to the user with a vocal speech synthesis function.

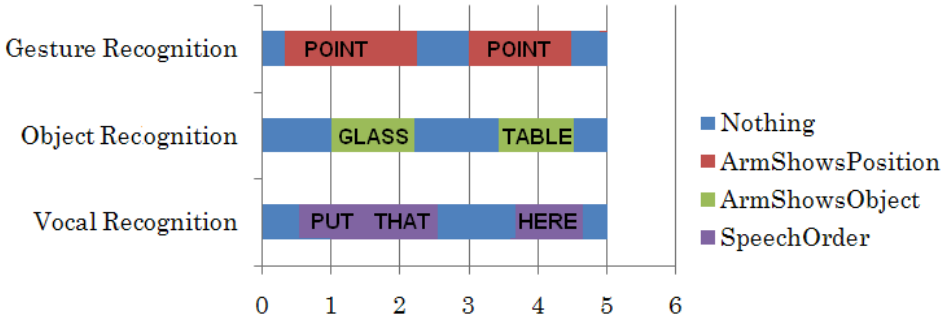


Figure 1.2 – Temporal Constraints to respect

To understand the human desire and realize the task, our robot need to deal with temporal constraints related to synchronization of events in respect of time and ordering in the best case (Figure 1.2). Any other events that don't match this coordination must be ignored and don't provoke a deadlock situation for the robot. We can see that some conditions must be taken in consideration:

- *To model* the problem and find a proper architecture
- *Modalities management*: to use several modalities together to reinforce the interaction
- *Events Completeness*: to check if all events required are present to decide to act;
- *Consistency* to check if all entities or objects are known to be recognized;
- *Modalities arrivals order*: to verify if events sent are respecting the expected order;
- *Temporal aspects* of vocal and gestural commands, taking into account the time needed by a modality to detect a command and the time to merge data;
- *Context adaptation*: time may change according to the user.

1.4 Methodology and Contribution

Our aim is to reinforce interaction with multi-modalities systems. The issue is very hard because human interaction requires the understanding of the human environment. This is an important challenge for complex systems engineering. Lots of information from different sources must be processed and organized. Knowledge representation and reasoning must be studied. The main question we will answer in this dissertation is how to develop the most flexible multimodal architecture that combines advantages of both top-down methodologies (modelling functionalities) and bottom-up methodologies (exploiting, checking, learning from experience and quality returns).

Our contribution can be decomposed in these following pragmatic parts:

- A methodology to model interaction architecture of complex systems
- Modelling and understanding the environment using a knowledge representation language
- Implementing semantic agents able to store, reason and communicate
- Designing services components able to communicate with agents, and to compose and adapt our architecture in a pervasive computing way
- Building a sensors/actuators services concentrator embedded in an electronic board (integrating a Linux operating system and an administration web site)
- Providing powerful mechanisms of composition, selection, adaptation and learning
- Checking performances
- Developing several applications

Our methodology is presented in Figure 1.3 and shows the order of the chapters which are much detailled in the next section. This methodology is a standard approach to explore the state of the art in scientific literature in order to obtain common definitions and methods in different domains and choose a way, the background theories and stringently examine algorithms, for high level

functions and components to solve our problem. The second step is to compile requirements of the systems of the environment, to give our definitions about knowledge, to build a knowledge representation language for the description of the environment, to model our architecture and design its components. Third step is to model an architecture capable of organizing components with complex mechanisms to improve interaction. Fourth step is the development of a platform to build prototypes of architectures and its evaluations. We added a comparative study of our model of semantic architecture with some other existing components, architectures and platforms introduced in first step. In the fifth step, we present the implementation procedure and validate our architecture prototype on different uses cases. Sixth step is the general synthesis of our work and proposes future work.

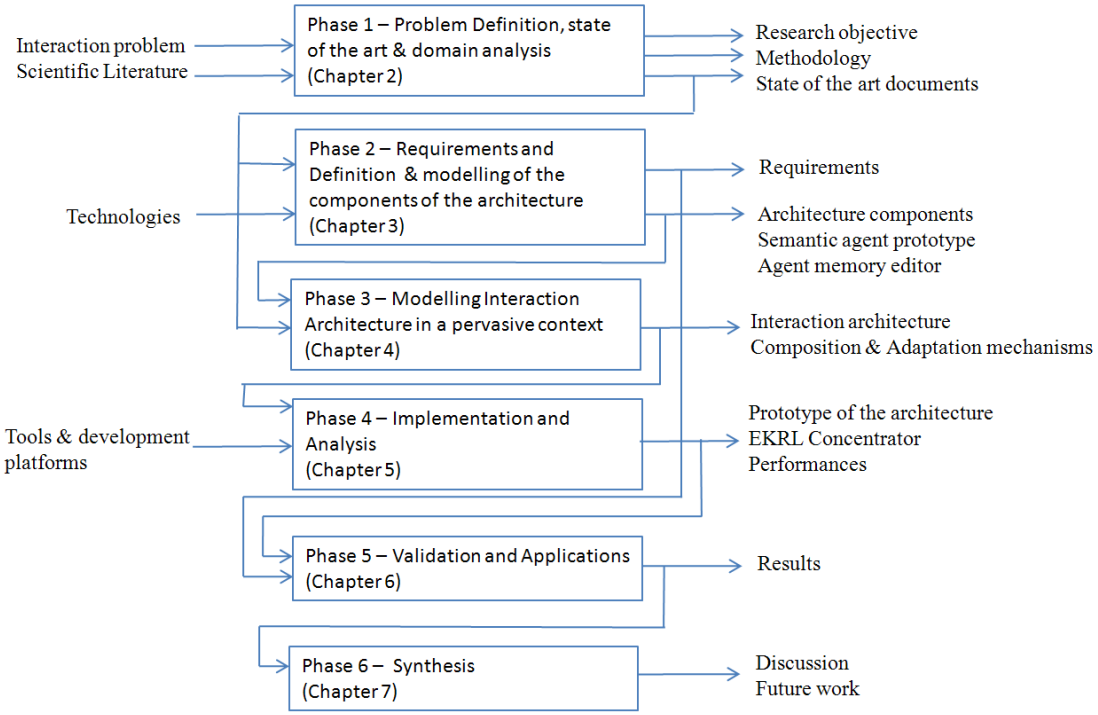


Figure 1.3 – Methodology

Artificial Intelligence and Computational Intelligence domains are studied because they try to understand and reproduce the human behaviour and thinking. We also have studied architectures based on semantic agents and cognitive architectures able to solve our interaction problem. Exploration of multidisciplinary scientific domains has required considerable effort in the understanding of the relationships between:

- Knowledge engineering: ontological representation & reasoning (semantic and understanding) for environment modelling;
- System and software engineering: Architectures for Multimodal interaction, Ambient Intelligence, Ubiquitous network, W2C open standards, Robotics or mechatronics;
- Intelligence: Artificial Intelligence, Computational Intelligence, Cognitive architectures for interaction.

1.5 Thesis Outline

Chapter 2 provides background for many of the subsequent chapters by surveying the four main domains concerned by this thesis: multimodal interaction with human and all systems in the environment; knowledge representation and reasoning for the description and the understanding of the environment; symbol grounding and artificial intelligence aspects; ubiquitous computing, ambient intelligence and web services in the respect of W3C recommended standards; and finally, a review of existing architectures for interaction and cognitive architectures.

Chapter 3 describes the requirements of our architecture in systems, our definition of conceptual and behavioural knowledge and the OOFDA model for interaction. Interaction is decomposed in semantic fusion and fission operations. This chapter delves thus into the special challenges of representing and reasoning with some core domains of distributed knowledge. We present our Environment Knowledge Representation Language (EKRL) becoming the common language to all architecture components allowing architecture components to communicate, store and reason on situation with a semantic wealth. We raise a variety of interesting ontological issues such as modelling information of dynamic environment such as time, space, causation, action, scenario or task, scales and contexts at different level of granularity. These challenges are ubiquitous across application areas, so solutions must be general and composable. Then we model the components of the architecture as generic as possible. In particular, input/output services interconnected with systems in the environment are presented and semantic agents are built to store, reason on knowledge and decide how to make the architecture interact with the environment.

Chapter 4 explains how to build complete interaction architecture from designed components in pervasive contexts. Powerful mechanisms applied to the architecture like discovery, composition, modalities selections and multi context awareness are formally explained. They are necessary to develop ambient intelligence architecture in order to manage complex tasks and environment and reinforce the interaction. Composition of numerous software or sensor-motor functions individually is modelled in a unified interaction model composed of the two main operations fusion and fission. To understand and adapt the global behaviour of the architecture, we propose semantic learning techniques.

Chapter 5 presents our framework implementation and explores its integration in several machines, third party platforms and the EKRL concentrator exchanging messages between services, devices, machines and Internet. An analysis of performance is performed to evaluate the number of events which can be managed, the memory access time, the network load and cognitive load. Algorithmic complexity of agent inference engine is analysed to evaluate the required power and memory resources. We also formally propose a temporal validation of the agents' inference engine in the case of "PutThatHere" scenario and in the general case of reasoning on any behavioural models. Finally, a comparative study with other multi-agent architectures and semantic agent components is discussed to show the difference with our approach.

Chapter 6 provides an implementation procedure for users and a mixture of interesting use cases like daily activities monitoring and assistance, healthcare and crossroad assistance. These applications show how to apply our framework

to interaction scenario, and demonstrate how our architecture is easy and powerful to design complex applications by focusing on concepts and models in the knowledge of agents without programming.

Chapter 7 is a synthesis of our contribution. We present our solutions to fulfil the requirements and we propose future work.

CHAPTER 2

BACKGROUND THEORIES

« The whole history of science has been the gradual realization that events do not happen in an arbitrary manner, but that they reflect a certain underlying order, which may or may not be divinely inspired »
Stephen William Hawking

2.1 Introduction

In the objective previously defined in the chapter 1, we need to explore the scientific state of the art in different domains of research to extract common definitions and theories useful to solve our interaction problem, model the environment and design the components of our architecture.

Actually, the design of the architecture for interaction with distributed components requires a large effort of search and preciseness. For this main reason, this chapter looks like disparate sections divided into five sub problems and four surveys. But it was very important for us to explain the basis of where components come from and why we decide to model them as they are. And the architecture will clearly makes the link between these parts emergent of these multiple research fields of study. Sometimes, it is necessary to give many explanations to go deeply into the subject and other times, only certain ideas or brief replies are appropriate; in the latter case, a quick review is sufficient. We will follow this thread:

1. Before modelling the environment and a new architecture for the interaction, it is necessary to find a definition of what are the multimodal interaction and the different contexts. We also define the two main interaction processes: fusion and fission. Fusion will permit to extract the meaning of information coming from the different input modalities connected in the environment while fission will permit to decide and select the output modalities to act in this environment.
2. A second step will be a short explanation of the symbol grounding and how Artificial Intelligence is an underlying domain to understand and reason about the environment. This section is about languages and symbolic representation of systems coexisting in the environment. In particular, some researchers suggest that the intelligence and the adaptation of a system emerge from the physical interaction with the environment.
3. We naturally arrive to the third step. The concepts and behaviours of systems should be described and the formalization of description logics and languages for the knowledge representation is required. They are very important because they are used to build standards of communication between components of complex and extended architectures.
4. In the fourth step, we explain the new paradigm of ambient intelligence where components of the architecture are parts of ubiquitous computing and environments. Composition and adaptation of the architecture can be realized with the support of web services. The emergence of semantic web propels the need of knowledge representation language. A deep link between the previous symbolic representation, the formal languages and the distributed components will permit to define a powerful communicating architecture.
5. The fifth step consists to have an overall look at the current existing architectures for interaction under three points of view: human machine architectures, computational cognitive architectures and developed semantic components.
6. The last step refers to developments tools and platforms to implement our architecture and applications.

2.2 Interaction and Context Representation

Human-Machine Interaction is an important Robotics research area in the context of Robotics manipulation, navigation and assistance especially with the increasing uncertainties and complexities of the real world. This thesis is dedicated to enable systems to explore and learn about environment and exploit it, multimodal interactions between them and the objects they are manipulating or avoiding, and between the humans or others systems in this environment. Human-Machine Interaction involves acquisition context, context awareness, interpretation context and execution context.

2.2.1 Multimodal Interaction

A system is composed of a multimodal architecture (Figure 2.1). It contains sensors and actuators to drive this architecture. So, a large amount of events to exchange and to process must be taken into account. In this thesis we want to conceive efficient interaction architecture in ambient intelligence to reduce the above complexity. Information must be well organized and meaning of situation must be quickly extracted to take decision. Meaning of the situation in the interaction requires developing a description of the current relationships among entities and events in the environment context. The extraction of this meaning is very important for the interpretation. Scenarios of multimodal interaction to be realized will contain a multimodal precondition part called *fusion*, and a post condition part called *fission*.

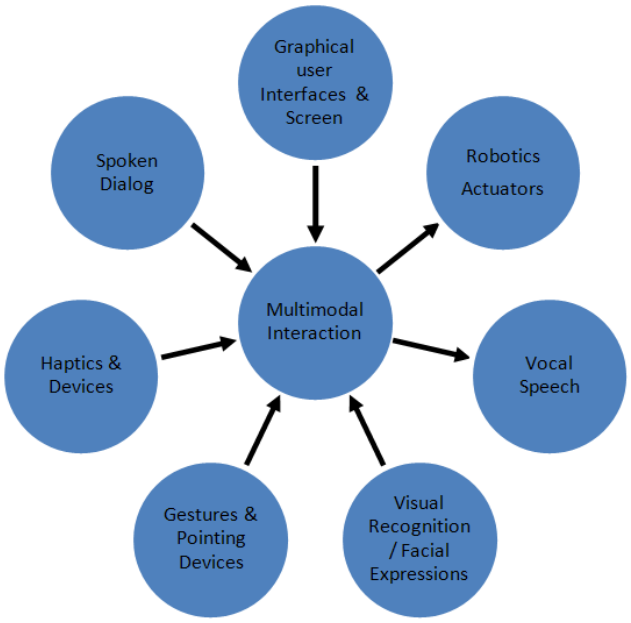


Figure 2.1 – Interaction Multimodal

Definitions of these two important processes of interaction are well presented in (Landragin, 2007) but to resume his view:

- Multimodal fusion starts from low level integration (signal information) to high level storage of the meaning (semantics event information) by composing and correlating previous data coming from multiple sources (sensors, interaction context, software services or web services) in the case of a human-robot or human machine interactions. *Multimodal Information fusion* refers to particular mathematical functions and

algorithms for data combination. Multimodal fusion is a central question to solve and provide effective and advanced human-computer interaction by using complementary or redundant modalities. Multimodal fusion helps to provide more informative, exact, complete, reliable interpretation.

- Multimodal fission, in the opposite way, is the process to physically act or show any reactions to the inputs or the current situation. According to decision rules taking following the fusion process, the fission will split semantics results of the decision into single actions to be sent to the actuators.

Dependency between modalities allows reciprocal disambiguation and improves recognition in the interpretation of the scene. Essential requirements for multimodal fusion and fission engines are the management of modalities with events, cognitive algorithms, formal logic and context representation of robotics systems or simulation environment.

This type of intelligent architectures matches the 3 modules (reactive-deliberative-reflective) architecture of Minsky (1974) and Searle (1969). Many researchers insert deliberation or decision process in the multimodal processes, at the end of the fusion process or at the start of the fission process. In this thesis, we propose to separate semantics aspects, cognitive aspects and fusion/fission processes. Semantics layer will be seen as a communication layer close to a language and cognition into intelligent agents able to communicate and manage memory storage and retrieval.

2.2.2 Interaction Context

The robot needs to be endowed with natural interaction capabilities by using a multimodal system that combines multiple input modalities such as natural speech, pen-based input, hand gestures, facial gestures, eye gaze, body language, or tactile input. Furthermore, the interaction process poses a set of problems stemming from incomplete a priori knowledge about the environment, hazards, strategies of exploration, insufficient sensory information, inherent inaccuracy in the robotic devices and the mode of operation. By combining different modalities, the robot can effectively reduce recognition uncertainty, thereby improving robustness while adapting the manipulation or the locomotion tasks in a dynamically changing environment. Therefore, a great simplification of the interaction processing can be accomplished, as has been demonstrated by several multimodal systems.

In order to interact and recognize relevant events, we can store simple scenarios and complex scenarios. A scenario corresponds to the sequence of simple physical actions/motions, or a state-action graph, executed by simple effectors using sensory inputs to accomplish a simple manipulation or exchange with the environment. Each scenario needs lots of information related to the environment, to all objects and to present Humans. We call this an *interaction context*. In this context, we will give symbolic representations for physical interactions; we use symbols which do not refer simply to physical objects but rather to the embodied interactions between the robot and the objects in its manipulation environment. This interaction context will be modelled using the concept of ontology. Ontology will keep meaning of any situation, current or

planned interaction and running scenario in a representation language close to natural language.

Situation Awareness model (Figure 2.2) and *User Fusion Model*, using *Knowledge Representation Language* in a *Software Agents* organization permitting reasoning, are very close to our design view of an ideal Robotics Platform (Blasch et al., 2006). We will adapt this work to *Ambient Architecture* and *Human Robots Interaction*. Coradeschi and Loufti (2008) propose a review of *Perceptual Anchoring,* a transversal domain where it is necessary to associate a symbol to signal sensor level representations of a physical object by using context and higher level semantics information. This is important for us to have symbolic information sent by intelligent sensor but we propose another way of doing this.

The pervasive computing community increasingly understands that developing context aware applications should be supported by adequate context information modelling and reasoning techniques. Bettini et al. (2009) propose a survey on context-aware applications. These applications adapt to changing context information: physical context, computational context, and user context/tasks. Context information is gathered from a variety of sources that differ in the quality of information they produce and that are often failure prone. Context-aware applications should be supported by adequate context information modelling and reasoning techniques. These techniques reduce the complexity of context-aware applications and improve their maintainability and evolvability. They discuss the requirements that context modelling and reasoning techniques should meet, including the modelling of a variety of context information types and their relationships, of high-level context abstractions describing real world situations using context information facts, of histories of context information, and of uncertainty of context information.

Gupta et al. (2009) presents a promising method to observe human-object interactions. Their fusion system on image and video uses spatial and functional compatibility for recognition. Graphical and Bayesian models are used to understand human scenes and events. *Object Classification* and *Action Recognition* are just done by stochastic values. We will deeply improve these two operations by using a *Context Ontology* to store concepts, values for classification and predicates of models for actions.

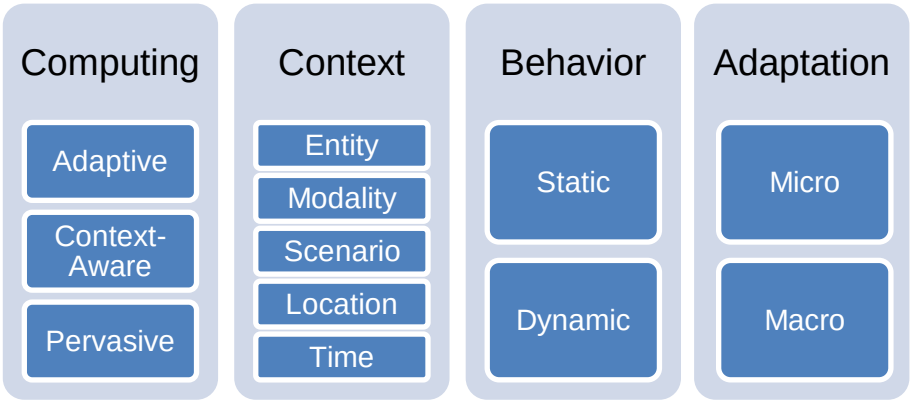


Figure 2.2 – Context-aware pervasive computing

Soylu et al. (2009) propose a review of context-aware pervasive computing (a.k.a. ubiquitous computing, ambient intelligence) with software and ontological

engineering. This paper is a sound basis for our work in terms of dynamic context and adaptivity, and awareness. they present a review of selected literature of context-aware pervasive computing while integrating theory and practice from various disciplines in order to construct a theoretical grounding and a technical follow-up path. This paper is not meant to provide an extensive review of the literature, but rather to integrate and extend fundamental and promising theoretical and technical aspects found in the literature. Their purpose is to use the constructed theory and practice in order to enable anywhere and anytime adaptive systems in the environment. They particularly elaborate on context, adaptivity, context-aware systems, ontologies and software development issues.

2.2.3 Context Classification

Context awareness requires that contextual information is collected and presented to the application of adaptation. Given the heterogeneity, diversity and quality of this information, it is desirable to make a classification or a categorization to facilitate the process of adaptation. In this area, several researchers have proposed various approaches of categorizations.

Schilit et al. (1994) and Dey (2001) have categorized the context into two classes: the *primary context* that contains information about the location, identity, time and activity (status); the *secondary context* can be inferred from it (e.g. the location, you can deduct those nearby).

Chen and Kotz (2000) have proposed two categories: the *active context* that influences the behaviour of an application and *passive context* which is necessary but not critical for the application.

Petrelli et al. (2000) have indicated the *material context* (location, object, machine, physical platform) and the *social context* (social aspects as the relationship between individuals).

Hofer et al. (2002) have developed a classification into two classes: the *physical context* that can be measured by physical sensors and *logical context* that contains information about the interaction (the emotional state of the user, its goals, etc.).

Razzaque et al. (2005) have proposed as follow a categorization in six classes:

1. *User Context*: It permits to obtain information on the users. User profile for example is part of it. It may contain information about its identification, its relations with other users, the list of tasks, and others;
2. *Physical Context*: It offers the ability to integrate information related to physical environment, like location, humidity, temperature, level of noise, and so on ;
3. *Network Context*: It also provides environment information, but these related mainly to the computer network. For example: connectivity, bandwidth, used protocol, names, and so on.
4. *Activity Context*: lists the events that took place in the environment and their timestamps. Example: entry of a person, it snows, and so on.
5. *Material Context*: It permits to identify all physical objects and systems in the environment which can be used. It includes for example the profile

and machines activities in the environment (identification, location, level of battery, and so on);

6. *Service Context*: It informed on what can be achieved. For example: Information on the functionality that the system can offer. The last categorization has the advantage to cover components considered by the first categorizations made. However, if we refer to the six questions proposed by Brezillon et al. (2004) and which would provide the minimum information on the context.

Miraoui and Chakib (2007) give 2 classes:

1. The information *triggered a service* that aggregates information on changing values causing the automatic activation of services provided by an information system making decision systems;
2. The information *quality change of services* that include information that the change in values leads to change the format of a service (quality).

This categorization has two main advantages: it is *simple* because it has only two classes; it is *complete* because it covers all aspects of context, particularly the six issues (Brezillon et al., 2004). “All definitions can be assembled around six questions: Who? What? When? How? Where? and Why?”. The question "Why?" is clearly ignored in this categorization.

2.2.4 Context Use

The context is taken into account in various Computer Science fields including *Natural Language*, *Machine Learning*, *Multimodal Interaction*, *Information Retrieval*, *Ambient intelligence* and even in *Security*. Just as in the interaction between humans, the purpose of taking into account the context is to enhance the adaptability and decision aid system. The *Ambient Computing* is very closely linked to the context because of the fact of the heterogeneity and ubiquity of communicating entities in this type of environment. These two aspects of the architecture require the adaptability of delivered services and more information on the used media depending on location, activity, i.e. the context. We speak here of *context aware* systems because they are sensitive to context and they are able to use the context of an entity to adapt their operational functions to provide better services to humans or others entities.

We talk here about *ambient systems sensibility*. Chalmers (2004) identifies five main context uses in computing systems as follow:

1. *Context Sensors*: where the context is captured and information describing the current context (temperature, location, …) are prepared and well presented to the users or others services;
2. To associate the context to data, this operation is called *contextual improvement* (or augmentation). For example, *records* on inspected objects can be associated to their location; *meeting information* can be linked to people in the meeting and location where it has taken place;
3. To permit the *discovery of contextual resources*, for example, to switch on the light in the nearest human location;
4. In the case of *context triggered events* to start actions like loading maps data when arriving in a city area;

5. *Contextual mediation*: It consists to use the context to modify a service. For example to describe the limits and preferences in a large amount of data, and then display only the most appropriate.

Dey (2001) and Chen and Kotz (2000) have covered recent research works relative to context, by underlying on their applications, the contextual information used and how to use it. The result is two reviews of scientific literature that most of researchers in Ambient computing often recall and especially the location of the user contextual information. Oddly, in rare cases, a temporal index is used and the location of nearby objects. This could be explained by the difficulties associated with the retrieval of contextual information and its treatment.

Sonntag et al. (2007) uses the W3C standards (EMMA, OWL-S, MPEG7, WSDL) to develop a context-aware multimodal mobile interface embedded in a PDA to the semantic web. This is a great job integrating a *Question Answering* system and *Natural Language Processing* system using speech and gestures to ask and retrieval information about web services, semantic web pages or the Internet network. Semantic web and in particular *Knowledge Ontology* is used to control interaction sequence for dialog and find links between modalities (video, audio, voice). This ontology makes part of SUMO project that we will present in section 2.4.6.

Our main interest is to take account the context to reinforce the adaptability of multimodal architecture in the ubiquitous environments.

2.3 Symbolic Representation

Knowledge Representation and Reasoning is at the heart of the great challenge of Artificial Intelligence: to understand the nature of intelligence and cognition so well that (computerized) systems can be made to reveal human-like abilities. As early as 1958, John McCarthy contemplated Artificial Intelligence systems that could exercise common sense. From this and other early work, researchers gained the conviction that (artificial) intelligence could be formalized as symbolic reasoning with explicit representations of knowledge, and that the core research challenge is to figure out how to represent knowledge in computers and to use it algorithmically to solve problems.

2.3.1 Problems of Traditional AI

However, the original intention of artificial intelligence was not only to develop clever algorithms, but also to understand natural forms of intelligence, which requires a direct interaction with the real world. It is now generally agreed that the classical approach has failed to deepen our understanding of many intelligent processes. How do we make sense of an everyday scene or recognize a face in a crowd, for example? How do we manipulate objects, especially flexible and soft objects and materials like clothes, string, and paper? How do we walk, run, ride a bicycle, and dance? What is common sense all about, and how are we able to understand and produce everyday natural language? Needless to say, trying to answer these questions requires us to consider not just the brain, but how the body and brain of an intelligent agent interact with the real world.

Recent developments in artificial intelligence and cognitive science suggested a new paradigm (called “embodied intelligence” (Pfeifer and Scheier, 1999) where

behaviour is not only the result of centralized control, but also emerges from the interaction of central control, body (morphology and materials) and the environment (Brooks, 1988). Development of intelligence based on inputs, possible actions (mental, embodied in the physic of the system, or external in the environment), centralized control (reactive or deliberative) and even distributed control. Pfeifer theory is only the old State-Cognition-Action paradigm where cognition improves from scratch using perception. It was also the first idea proposed by cyberneticists (Dupuy 2000) in their mechanical and physical point of view of living beings but most of their theories were vanished behind the connectionism paradigm. Perceiving the world also implies the notion of consciousness (Bronowski 1970). He is certainly right for animals and body parts adaptation (mechanical parts) to avoid millions of year's evolution. It is also simpler to highlight intelligent behaviour when embodied (so observable or measurable) and it is easier to perceive things to understand them. I personally think artificial intelligence firstly requires a good brain with an efficient memory and a good reasoning system to store, interpret, learn and create new knowledge to improve thinking. We believe behavioural aspects to be only the outcomes of intelligent reasoning and not the intelligence itself but it is necessary to have inputs to understand the relationships between entities and their behaviours and outputs to act on them. One way to improve its intelligence is to equilibrate both bottom-up (perception to logic) and up-bottom (logic to action) approaches, not only one of these two to avoiding instabilities (i.e. possible schizophrenic states following uncertain events). That's also the main idea of the paper of Roy Deb presenting a theoretical framework to ground the meaning of verbs, adjectives and nouns to physical referents using a unified representational scheme where all words are composed of sensory-motor primitives called *semiotic schema* (Deb 2005). In this holistic view, a goal appears more complex to achieve and many tasks have to be solved in parallel, logical at high level, and perceived or executed at low level. We may talk about a main goal like a living equilibrium composed of motivation sub goals like maintaining of energy, health, attention or satisfaction. We will propose a mechanism to manage this. As in classical AI, meaning is concerned by rules (atoms and formulae) but not just rules; it is as much concerned by relationships of composition and recursion of these rules. Knowledge representation in our work should take into account this point.

2.3.2 The Symbol-Grounding Problem

The term "symbol" in this context is taken to mean, roughly, discrete information elements that may be given word-like labels that make sense to humans. It may be a concept, an action or predicate or relationships between the two previous. Pure symbolic (from classical AI) is considered "ungrounded" approach. In contrast, the "grounded" scientist treats language as part of a larger cognitive system in which semantics depends in part on non-symbolic structures and processes including those related to perception and motor planning. Examples include Bates' grounding of language in sensory-motor schemas (Bates, 1979) and Barsalou's proposal of a "perceptual symbol system" that grounds symbolic structures in sensory-motor simulation (Barsalou, 1999; Barsalou, 2003). The grounded camp pays more attention to how symbols arise from, and are connected to interactions with the physical and social environment of the symbol user. The ungrounded camp has the advantage of mature computational modelling tools and formalisms. Although these have

proved to be immensely useful, there are also clear limits to the explanatory power of any approach that deals strictly with relations among word-like symbols.

The symbol-grounding problem refers to how symbols relate to the real world (Harnad 1990). In traditional AI, symbols are typically defined in a purely syntactic way by how they relate to other symbols and how they are processed by some interpreter (Newell and Simon 1976; Quillian 1968); the relation of the symbols to the outside world is rarely discussed explicitly. In other words, we are dealing with closed systems, not only in AI but in computer science in general. Except in real-time applications, the relation of symbols (e.g., in database applications) to the outside world is never discussed; it is assumed as somehow given, with the (typically implicit) assumption that designers and potential users know what the symbols mean (e.g., the price of a product). This idea is also predominant in linguistics: it is taken for granted that the symbols or sentences correspond in some way with the outside world. The study of meaning then relates to the translation of sentences into some kind of logic-based representation whose semantics are clearly defined (Winograd and Flores 1986, p. 18).

Logical statements are different at different level of abstraction, granularity or point of view (for example, at several orientations of moves correspond to a straight walk), that means that pure logic is not sufficient to explain a behaviour. Mathematicians are now working on multimodal logics with a multiplicity of operators to be adapted to the numerous forms of language. Predicates logic and Description logic bring a part of the solution to the lake of representation but they are also limited to the first order logic (FOL) to ensure the completeness. In reality, higher order logic is more close to the reality but cannot be fully proven in general but may be proven in the simplified context. That's why, we used a knowledge representation language (KRL) to use predicates in higher level logic and multimodal logic at different levels of abstractions. The objective is to reduce the reasoning time and memory space necessary to solve a problem. This dimensionality curse is also a problem. So another solution to limit the search in state and action space is the *bounded rationality* theories of Herbert Simon (1982). As explained by Russell and Subramanian (1995), "The classical idea of perfect rationality, which developed from Mill's utilitarianism, was put on a formal footing in von Neumann's decision theory (von Neumann and Morgenstern 1944). It stipulates that a rational agent always act so as to maximize its expected utility. The expectation is taken according to the agent's own beliefs; thus, perfect rationality does not require omniscience". Recently, (Nobre et al. 2009) connect theories of bounded rationality with theories of fuzzy systems of Zadeh in order to justify advantages of the participation of cognitive machines in organizations. The connections are derived by explaining why cognitive machines can extend limits of knowledge (lack of information) and limits of information processing and management (lack of cognition and computational capacity) of humans when participating in organizations.

2.4 Knowledge Representation & Reasoning

2.4.1 Semantic web RDF and OWL

Semantic Web is about putting together 3 things: the *representation of the meaning* of things that are published on the Web (called *resources*), the *representation of relations* that link things together, the *standardization* of the previous two. We need to represent something about entity's semantics in a way that allow computers to do something useful for the humans. We also need to share such representations, i.e., make them popular, widely adopted and standard, so that all collectively can benefit from that, in a way that would not be possible in isolation, reinforcing domain knowledge.

RDF Resource Description Framework

The *resource description framework* is a language for representing information about resources in the World Wide Web (Figure 2.3). It is particularly intended for representing metadata about web resources, such as the title, author and modification date of a web page, copyright and licensing information about a web document, or the availability schedule for some shared resource. However, by generalizing the concept of a *web resource* as defined above, RDF can also be used to represent information about things that can be identified on the web, even when they cannot be directly retrieved on the web. Since the approach of situational statements is based on RDF, a profound investigation of the resource description framework has been undertaken. A detailed introduction can be found in the so-called "RDF-Primer" document (Manola and Miller, 2004)

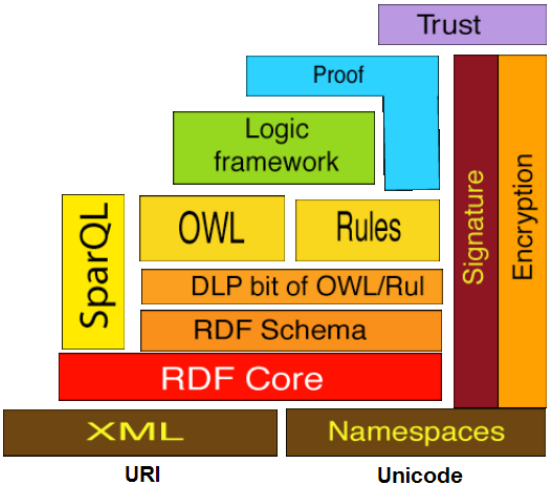


Figure 2.3 – Semantic Web layers (W3C Schema)

In (Stuckenschmidt and Van Harmelen, 2005) it is pointed out that, if we wanted to describe information on the meta-level and to define its meaning, we have to look for further approaches than XML and the RDF standard has been proposed as a data model for representing meta data. (Brickley and Guha, 2004) and (Lassila, 2000) are important papers on RDF. Nonetheless, this abstract data model finds a concrete syntax representation in XML. The corresponding schema language to define the vocabularies is called RDF Schema, or RDFS (Manola and Miller, 2004). In RDFS one can define which properties apply to which kinds of objects and what value ranges hold. Furthermore one can describe the relationships between objects.

OWL Ontology Web Language

The prominent ontology web language OWL has more facilities for expressing semantics compared to XML, RDF, and RDFS. (McGuinness and Van Harmelen, 2004) provides an overview. Furthermore, it has greater machine interpretability, according to Stuckenschmidt and van Harmelen (2005). OWL adds more vocabulary for describing properties and classes and among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes. OWL can especially be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. OWL is a revision of the DAML+OIL web ontology language, in which, the first user model ontology was defined (Heckmann, 2005). Chen and Kotz (2002) combine semantic web and OWL with ubiquitous computing. Eberhart (2003) defines an ontology based infrastructure for intelligent applications. The general user model ontology GUMO and the UbisWorld ontology are defined as OWL applications. In OWL, *concept* of Description Logic becomes *class*, *role* becomes *property* and *individual* become *object* or *instance*. We have developed OWL v2 import/export software to our KRL format to keep completeness and constituency of relationships between concepts in respecting standards. We can then connect our system to OWL base.

Comparing RDF, OWL, UML and OCL

RDF is a set of concepts or a language used for describing knowledge. OWL is a set of concepts or language for modelling that knowledge. UML is a semi formal language for class-based modelling and OWL comprise some constituents that are similar in many respects like classes, associations, properties, packages, types, generalization and instances (ODM, 2008). Despite of the similarities, both approaches present restrictions that can be overcome by integration. On one hand, a key limitation of UML class-based modelling is that it allows only static specification of specialization and generalization of classes and relationships, whereas OWL provides mechanisms to define these as dynamic. It means that OWL permits recognition of generalization and specialization between classes as well as class membership of objects based on conditions imposed on properties of class definitions. OWL offers a more expressive and extensible manner of modelling data and provides versatile ways to describe classes and, based on such descriptions, it allows type inference. Indeed, OWL provides various means for describing classes, which may also be nested into each other such that explicit typing is not compulsory. One may denote a class by a class identifier, an exhaustive enumeration of individuals, property restrictions, an intersection of class descriptions, a union of class descriptions, or the complement of a class description. OWL provides important features complementary to UML and OCL/MOF (OCL, 2010) that would improve software modelling: it allows different manners of describing classes; it handles these descriptions as first-class entities; it provides additional constructs like transitive closure for properties; and it enables dynamic classification of objects based upon class descriptions. OWL ontologies can be operated on by reasoners providing services like consistency checking, classes satisfiability, concept and instance classification. Specification of OWL2 (Cuenca Grau et al., 2008) contains several sub-languages, or profiles (Calvanese et al., 2009), which offer increased tractability at the expense of expressivity (Parreiras, 2009). Ontologies and web service architecture are now used to design and adapt MAS

(Goumopoulos and Kameas, 2009) bringing interoperability (Obrst, 2003). EKRL can easily integrate OWL. In (Dourlens and Ramdane-Cherif, 2011c), we mention Architecture Description Languages (ADL). One ADL most interesting is π-ADL (Oquendo, 2004) (Oquendo, 2005), a formal language with its primary focus on the software architecture description at the mathematical sense and the ability to reason about its behaviour. π-ADL formal foundations are based on higher-order typed π-calculus. π-ADL encompasses the architecture centric constructs for representing the structure, as well as behaviour details.

2.4.2 Multimodal Mark-up Languages

Since 1995, we have seen the emergence of a third generation of chatterbots, based on the use of markup languages. Artificial intelligence Markup Language (AIML), the first of these languages, was used in the construction of ALICE[1]. It is the owner of two recent Loebner prizes[2]. Despite its success, the systems developed using AIML are still present. To use AIML, we need to define lots of categories[3] and it is too simple (links are repetitive, deductive logic, templates are too close to the form of the sentences) but in many case, it can be useful for chatterbot on a precise subject of talk and following a dialog.

The idea of an extended language proposed by Makatchev and Tso (2000) was good but did not survive. It has not been used because it was not clearly defined and not so complete to be useful. The articles about Robot Markup Language (Kwak et al., 2006) and Multimodal Presentation Markup Language (Kushida et al., 2005) are interesting but limited to their applications. For the second one, authors have conducted researches into multimodal presentations that make use of anthropomorphic character agents as a new type of multimodal media that can be used to effectively communicate information in conjunction with the World Wide Web (WWW) and they have developed a markup description language for this purpose, the Multimodal Presentation Markup Language (MPML). They have developed a Multimodal Presentation Markup Language for Virtual Reality VRML (MPML-VR), which is a markup language that enables the description of multimodal presentations in a three-dimensional VRML space and that inherits features from MPML. They have also implemented an MPML-VR Agent system, an anthropomorphic agent system that functions on the VRML platform. By using the three-dimensional VRML space, presentations such as showing products in three dimensions, virtual exhibition halls, and virtual museum environments become possible. The audience is able to move their location at will and view the contents from an arbitrary position. In this paper we report the MPML-VR markup language, MPML-VR Agent system that we have created for the VRML environment, and the MPML-VR Viewer, a presentation viewing system. VHML[4],

Virtual Human Markup Language, uses and built in 2000 on Sable and AIML to develop an animated software assistant like Microsoft Agent, for a more usability and help, but more conversational and multimodal. VHML describes new languages to accommodate functionality that is not catered for like DMML

[1] http://www.alicebot.org
[2] http://www.loebner.net/Prizef/loebner-prize.html
[3] http://www.alicebot.org/aiml/aaa/
[4] VHML project homepage: http://www.vhml.org

Dialogue Manager Markup Language (AIML), FAML Facial Animation Markup Language, BAML Body Animation Markup Language, SML Speech Markup Language (Sable), EML Emotion Markup Language.

Recently, W3C worked on Extensible MultiModal Annotation markup language (EMMA)[5], a new XML Markup Language to manage the communication between multimodalities. For example, it is used in mobile phone for voice interface with the user. It can be used with VoiceXML[6] concerning voice activity.

Robotics Markup Language (Roboml) (Makatchev, 2000) is no more maintained but has some applications in assistant for Japanese websites. It is not so Robotics oriented. RobotML (Park et al., 2007) is a Robotics Markup language from the Munich University of Applied Sciences in 2005, XML is used to describe robot joints and can be read by some Robotics simulators. Robotics multimodal parts are well described. More recently, MultiML (Giuliani and Knoll, 2008) (not OWL compatible) and EMMA (Johnston, 2009; Johnston et al., 2009) (OWL compatible) are the most advanced XML languages for Multimodal fusion and fission incorporating multimodalities, contexts, time but these languages don't deal with the interaction meaning but they bring a bridge to semantics interpretation, they want to represent modalities and recognize scenarios using their own XML tags. In addition, Giuliani and Knoll (2008) also present related work like MURML (Kranstedt, 2002), MMIL (Landragin et al., 2004), Cogest (Gibbon et al., 2003) and MIND (Chai, 2002) languages or systems built for conversational agents without standard ontologies. In Multimodal Interpretation for Natural Dialog (MIND), Joyce Chai focused on intention and attention, it doesn't use KRL and ontologies but it was close to our work in the interpretation of meaning. These XML-based languages are limited to first order logic reasoners. To understand the comparison with our work done in this thesis, we really want to open a door on an independent environment representative language where meaning appears from the linked concepts using predicates to represent knowledge using context and narratives (Sengers, 1998; Singh, 2005) but without complex human common sense. Reiter (2000) greatly proposes the original idea to use narratives as programs. This idea is at the heart of the design of our architecture which will be defined and conceived differently. We search to find some good cognitive or semantic component for our architecture to achieve goals in human environment using multimodal interaction.

2.4.3 Ontologies

Definition

Sowa (1999) gives the following definition for ontology: *"The subject of ontology is the study of the categories of things that exist or may exist in some domain. The product of such a study, called an ontology, is a catalogue of the types of things that are assumed to exist in a domain of interest D from the perspective of a person who uses a language L for the purpose of talking about D."* ... *"Ontological analysis clarifies the structure of knowledge. Given a domain, its ontology forms the heart of any system of knowledge representation for that domain. Without ontologies, or the conceptualizations that underlie knowledge,*

[5] EMMA homepage: http://www.w3.org/TR/emma
[6] VoiceXML homepage: http://www.w3.org/Voice

there cannot be a vocabulary for representing knowledge ... Second, ontologies enable knowledge sharing."

We adopt Gruber's definition referring to ontology as an explicit (not implicit) specification of conceptualization (Gruber, 1993), with the Guarino's interpretation of conceptualization (Guarino, 1995).

Formal Ontology

An explicit ontology may take a variety of forms, but imperatively it will include a vocabulary of terms and specification of their meaning (i.e. definitions). Ontologies are distinguished along a spectrum of formality and referring to a different degree of formality by which a vocabulary is created. In the literature, ontologies fall in many types based on different spectrum (Guarino, 1998; Sowa, 1984, 1999; Obrst, 2003). Therefore, the same term *ontology* can be used to describe models with different degrees of structure.

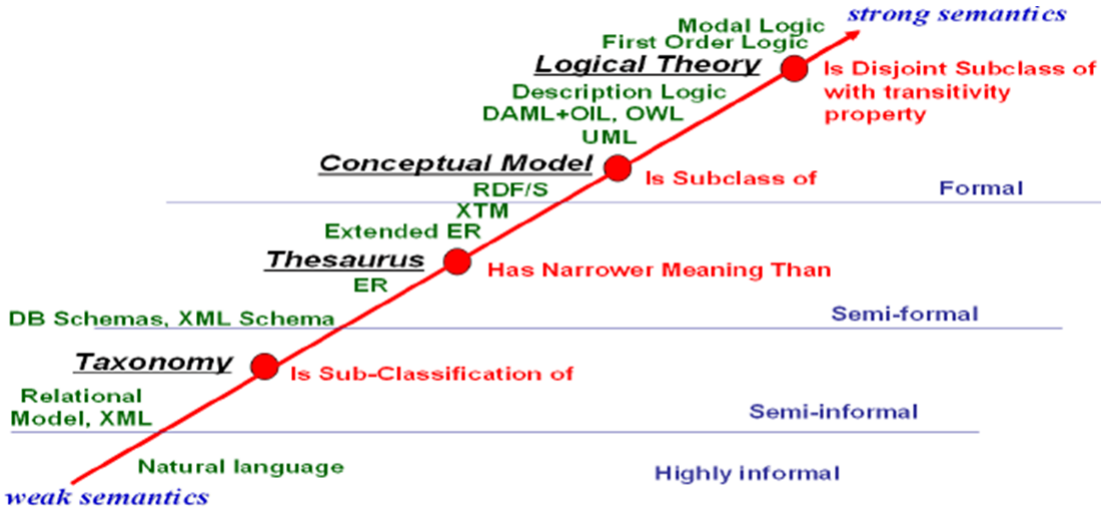


Figure 2.4 – Ontology spectrum updated from Obrst (2003)

Figure 2.4 shows the following spectrum:

- *Highly informal*: expressed loosely in natural language. An informal ontology contains a list of types that are either undefined or defined only by statements in a natural language.
- *Semi-informal*: expressed in a restricted and structured form of natural language. In this zone, we identify weak structure such as taxonomies.
- *Semi-formal*: expressed in an artificial formally defined language. this zone includes elements with more structure: in the bottom database schemas and metadata schemes (e.g. ICML, ebXML, WSDL), on the top of conceptual models (OO models, UML, XML topic maps, ...) and in-between thesaurus (e.g. Wordnet, Verbnet, NLP, OpenCyc, Lexicon) (Lenat, 1990; Panton et al., 2006)
- *Rigorously formal*: usually expressed in a logic-based language (e.g. first order logic) with theorems and proofs mechanism. Fundamentals of formal ontology include axiomatic theories organized into partial order (lattice). Axiomatic theories or simply axioms use first order logic for a rich expression of the constraints between the entity and relation types. We distinguish between theorems (proven) and theories (not proven). Using axiomatic-

deductive method provides deduction and derivation (non monotonic reasoning) which are based on completeness and decidability.

Types of Ontologies

According to the level of granularity in (Guarino, 1998) and (Gomez Perez and Benjamins, 1999), ontologies levels of abstraction can be classified into six types:

- *Top-level ontologies* describe very general concepts independent of a domain like time, space, matter, object, action
- *General ontologies* define a large number of concepts relating to fundamental human knowledge
- *Domain ontologies* describe the vocabulary related to a specific domain like medicine or automobile
- *Task ontologies* define concepts related to the execution of a task or activity such as diagnosing or selling
- *Application ontologies* define concepts essential for planning a particular application. They describe concepts depending both on a particular domain and task which are often specializations of both the related ontologies
- *Meta-Ontologies*, generic, or core ontology define concepts which are common across various domain; these concepts can be further specialised to domain as specific concepts.

Uses of Ontologies

Ontologies are used for:

- *Communication*: share understanding, as blackboard, unifying framework to enable communication, interaction & cooperation. 3 schemes are possible :
    1. Each services shares its own ontology
    2. Multiple ontologies can be attached to a single indexing global ontology and be used by multiple services with or without their own ontology
    3. Global ontology is alone and shared by several services with or without their own ontology

  As now processors and memory are grouped in clusters with a equilibrated load balancing, this question on the architecture is not so important but if communication can be interrupted or a cluster is broken, in this last case backup must be enabled or big ontology must contain a copy of local one.
- *Interoperability*: ability of systems to cooperate for a common purpose
- *Modelling*: specification and conceptualization, reliability, reusability. Using ontology to link Declarative (Domain knowledge) & Procedural (problem solving) knowledge!
- *Reliability*: reliability defines the ability of a system or components to perform its required functions under stated conditions. We can differentiate between the roles that ontology might play for reliability of software systems. Indeed, informal ontologies can improve the reliability by serving as a basis for manual checking of the design against the specification. Formal ontologies enable the use of semi-automated consistency checking of the software system with respect to the declarative specification.
- *Reusability*: The accomplishment of reusability largely depends on the sharing of a similar conceptualization. By characterizing classes of domains

and tasks within these domains, ontologies provide a framework for determining which aspects of a system are reusable between different domains and tasks. Indeed, a clear semantic is needed of the concepts related components being reused. Thus, ontology-driven Information System can import and export modules and components between their systems.

From an operational point of view, the development and the exploitation of ontology remains a complex task in a global process of knowledge engineering. Upstream, extracting and structuring large sets of concepts with increasing sizes represents one of the major difficulties. Downstream, retrieving subsets of concepts requires approaches that are not time-consuming and are efficient in terms of semantic relevance of the results. We will attempt to improve these performances. For information retrieval, we will try to find a fastest query process. For refinement and enrichment, most of researchers use distance measures or estimation of semantic similarities between pairs of concepts.

Lots of applications and case study using ontology exist and we can't exhaustive but we want to pay attention at the use of Ontology to produce automatic video annotation (Newbold et al., 2008), they used it to track “illegally” parked vehicles with partial observation of the captured scene. In (Chen and Kotz, 2002), authors develop the Context Broker Architecture (CoBrA), it's a framework built around a Context Ontology and an Ontology Inference Engine in OWL for context-aware pervasive computing environments. The concepts of *Agent*, *Role*, *and Activities* are well defined. Chella et al. (2002) present a method to model robotics environments integrating multi agents systems into ontologies.

Ontology Mapping

Pirrò and Talia (2007; 2010) worked on the ontology mapping problem. They developed UFOME a friendly environment to ensure extensibility and usability. There also the works of INRIA and UVSQ researchers like Euzenat on ontology alignment and matching (Euzenat and Shvaiko, 2007; David et al., 2010) who developed an Alignment API, Georges Gardarin who developed JANUS an automatic ontology builder (Bedini et al., 2008), and (Grigori et al., 2010) studied matching and ranking BPEL processes for services discovery where processes are in different OWL-S ontologies. Further in this direction, (Besana et al., 2009) propose to share choreographies comparing their OpenKnowledge framework to BEPL4Chor and Web Services Choreography Description language (WS-CDL) and *Let's Dance*, a language for service behaviour modelling.

2.4.4 Knowledge Representation Languages

Knowledge Representation Languages (KRL) are semantic languages allowing to build and extract the meaning of the situation past or current. Some examples of these languages are KRL, KIF, NKRL, INFOMAP. They are all based on the frames theory of Minksy (1965) and more precisely the slots of Quillian (1968). They may bring: Entity recognition, Semantic role labelling, Question answering making better use of relations among concepts to catch user or context semantics. There are two types of QA systems: FAQ systems: match the user query to a closest query (or query type) stored in the system and free queries using a corpus-based answer extraction. Fillmore (1985) proposes to

analyse frames and the semantics of understanding. The Berkeley FrameNet[7] project was born from this excellent idea but till now, they have no result maybe due to the mass of relationships. KIF and NKRL languages are interesting to realize the predicate calculus used to conceptualize the behavioural conceptualization. The *Suggested Upper Merged Ontology* (SUMO) nd its domain ontologies form the largest formal public ontology in existence today. They are being used for research and applications in search, linguistics and reasoning. SUMO is the only formal ontology that has been mapped to the entire WordNet lexicon. SUMO is written in the SUO-KIF language. SUMO is free and owned by the IEEE (Figure 2.5).

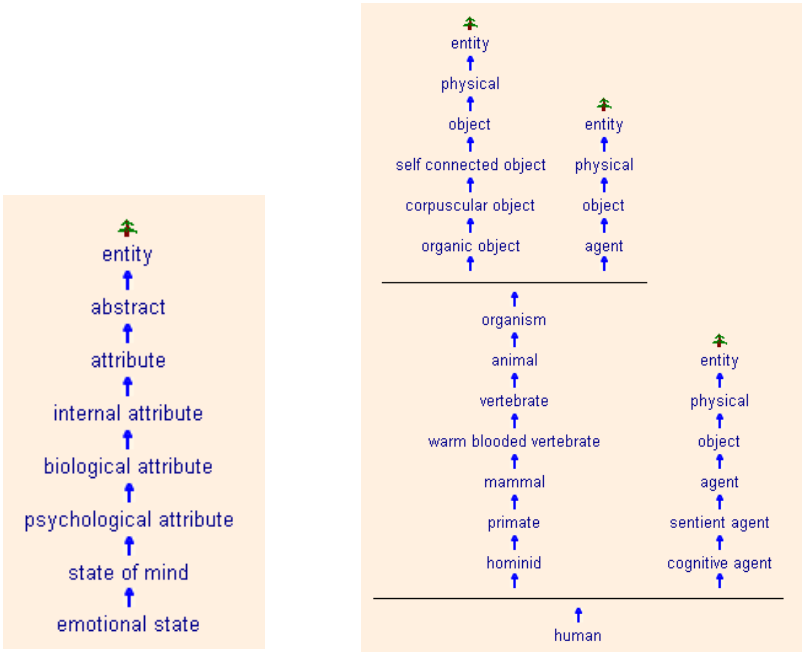


Figure 2.5 – Superclasses of Emotional State &- multiple inherited superclasses of the Human concept. (SUMO[8])

A Knowledge Interchange Format (KIF)

KIF[9] is the product of the Interlingua Working Group, chaired by Fikes and Geneseth (part of Knowledge Sharing Effort KSE initiated by DARPA[10] in 1990). The work on KIF has grown out of the recognition that an Interlingua needs to be a language with the following general properties:

- A formally defined declarative semantics.
- Sufficient expressive power to represent the declarative knowledge contained in typical application system knowledge bases; and
- A structure that enables semi-automatic translation into and of typical representation languages.

KIF is an extended version of first order predicate logic. The version 3.0 has the following features:

[7] FRAMENET http://framenet.icsi.berkeley.edu
[8] SUMO webpage: http://virtual.cvut.cz/kifb/en/
[9] KIF Standford page: http://www.ksl.stanford.edu/knowledge-sharing/kif
[10] DARPA homepage: http://www.darpa.mil

- List-based linear ASCII syntax suitable for transmission on serial media.
- Model-theoretic semantics with axiomatic characterization of a large vocabulary of object, function, and relation constants.
- Function and relation vocabulary for numbers, sets, and lists.
- Support for expression of knowledge about the properties of functions and relations. Functions and relations are included in the universe of discourse as sets of lists so that they can be arguments to relations. A `holds` relation is included that is true when its first argument denotes a relation that has as a member the list consisting of the items denoted by the remaining arguments. So, for example, one could define transitivity as follows:
```
(<=> (transitive ?r)
(=> (holds ?r ?x ?y)
(holds ?r ?y ?z)
(holds ?r ?x ?z)))
```
- A sublanguage for defining objects, *n*-ary relations, and *n*-ary functions that enables augmentation of the representational vocabulary and specification of domain ontologies. Definitions can be *complete* in that they specify an equivalent expression or *partial* in that they specify an axiom that restricts the possible denotations of the constant being defined. For example, the following is a complete definition of the unary relation `bachelor`:
```
(defrelation bachelor (?x) :=
   (and (man ?x) (not (married?x))))
```
and the following is a partial definition of a binary relation above which specifies that above is transitive and holds only for ``located objects'':
```
(defrelation above (?b1 ?b2)
:=> (and (located-object ?b1) (located-object ?b2))
:axiom (transitive above))
```

- Support for expression of knowledge about knowledge. KIF expressions are included as objects (i.e. lists) in the universe of discourse, and functions are available for changing level of denotation. For example, the following sentence says that Lisa has the same belief as John about the material of which things are made:
```
(=> (believes john '(material ,?x ,?y))
(believes lisa '(material ,?x ,?y)))
```

KIF is intended to be a core language which is expandable by defining additional representational primitives.

Knowledge Representation Languages (KRL)

Quillian (1968) introduced the frame and slots structure containing predicates, roles and attributes but without the use of ontologies previously introduced and limited to first order logic. It was based on Frege's logic. Frege has developed a second-order predicate calculus and used it to define mathematical concepts, and to state and mathematically prove propositions. Briggs (1985) suggests that knowledge representation research began with ancient Indian analysis of Shastric Sanskrit in the first millennium BC.

The representation approach of Zarri (1996) is more turned to express narratives (NKRL) from natural language or web semantic pages, and the processing of the meaning of temporal events. NKRL innovates with respect to

the usual ontology paradigm by associating with the traditional binary ontology of concepts, an ontology of events, i.e., a new sort of hierarchical organization where the nodes correspond to *n*-ary structures called *templates*. Instead of using the usual *object (class, concept) – attribute – value* organization, templates are generated from the combination of quadruples where each of them connect together the *symbolic name* of the template, a *predicate*, and the *arguments* of the predicate introduced by named relations, the *roles*. The quadruples have in common the *name* and *predicate* components. If we denote then with L*i* the generic symbolic label identifying a given template, with R_k the generic role and with A_k the corresponding conceptual argument, the NKRL core data structure for templates has the following general format:

$$(L_i\ (R_1\ A_1)\ (R_2\ A_2)\ \ldots\ (R_n\ A_n)) \qquad \textit{(Formula 1)}$$

Predicates pertain to the set {BEHAVE, EXIST, EXPERIENCE, MOVE, OWN, PRODUCE, RECEIVE}, and roles to the set {SUBJ(ect), OBJ(ect), SOURCE, BEN(e)F(iciary), MODAL(ity), TOPIC, CONTEXT}. An argument of the predicate can consist of a simple 'concept' (according to the traditional, *ontological* meaning of this word) or of a structured association ('expansion') of several concepts. The NKRL standard *ontology of concepts* is called HClass, *hierarchy of classes*. Templates are included into an inheritance hierarchy, HTemp(lates), which implements then the new *ontology of events.* Figure 2.6 is a sample of a template and an instance of a BT service to a customer.

name: Move:TransferOfServiceToSomeone

father: Move:TransferToSomeone

position: 4.11

natural language description: 'Transfer or Supply a Service to Someone'

MOVE SUBJ *var1*: [*var2*]

OBJ *var3*

[SOURCE *var4*: [*var5*]]

BENF *var6*: [*var7*]

[MODAL *var8*]

[TOPIC *var9*]

[CONTEXT *var10*]

{[modulators]}

var1 = human_being_or_social_body

var3 = service_

var4 = human_being_or_social_body

var6 = human_being_or_social_body

var8 = process_, sector_specif ic_activity

var9 = sortal_concept

var10 = situation_

var2, *var5*, *var7* = geographical_location

c1)

```
MOVE SUBJ BRITISH_TELECOM
     OBJ payg_internet_service
     BENF (SPECIF customer_  BRITISH_TELECOM)
     date-1: after-1-september-1998
     date-2:
```

Figure 2.6 – NKRL Example

According to Zarri (2009a), usual ontologies – both in their 'traditional' and 'semantic web' versions like RDF and OWL – are not very suitable for dealing with elementary or complex events. Basically, ontologies organize the 'concepts' – i.e., the important notions to be represented in a given application domain – into a hierarchical structure where the nodes (the concepts) are defined by a set of binary relationships of the 'property/value' type (e.g., a 'frame'). This approach is largely sufficient to provide a *static*, *a priori* definition of the concepts and of their properties. The only negative points are translating natural language assertions (to add new events) and questions (for the query systems) in NKRL are not automatic. Anyway, the work done by Zarri is quite innovative for any event-driven application architecture. Now, numerous works use conceptual and behavioural ontologies and inference engines we will present later but they are focusing on the query of semantic web information rather than environment understanding. Dourlens & Ramdane-Cherif (2010; 2011) presented their previous work on modelling a semantic architecture using KRL. Sabri et al. (2011a and 2011b) published a similar work. Their modelling of a context aware architecture is based on the Zarri's NKRL inference engine. They are service-oriented but not standards.

2.4.5 Event Representation

Moens and Steedman (1988) elaborate Vendler's work (Vendler, 1967) on temporal and aspectual categories, and discuss the following types of events:

- *Culmination*: a punctual or instantaneous event, which is accompanied by a transition to a consequent state. Ex*: Harry (has) reached the top*.
- *Point*: an indivisible event (not necessarily instantaneous) with no consequent. Ex: *John hiccupped*.
- *Process*: an event which is extended in time but that does not lead to any particular conclusion or culmination. This type of event can take a for-adverbial phrase (e.g., *He climbed for an hour*), but will not allow for a non-adverbial phrase (e.g., **He climbed in an hour*).

Culminated process: state of affairs that extends in time but does have a particular culmination associated with it at which a change of state takes place. This type of event can take an in-adverbial phrase (e.g., *He climbed to the top in an hour*), but will not allow a for-adverbial phrase (e.g., **He climbed to the top for an hour*). Ex: *Harry climbed to the top*. Ex: *Harry climbed*.

Table 2.1 shows schematically how events can be characterized by atomicity or extension over time and by whether they have a consequent state or not. Events can be inter-related through culmination, preparatory processes, iteration, and progressive links between them.

	Events		States
	atomic	extended	

+Consequent	CULMINATION (recognize, spot, win the race)	CULMINATED PROCESS (build a house, eat a sandwich)	understand, love, know, resemble
-Consequent	POINT (hiccup, tap, wink)	PROCESS (run, swim, walk, play the piano)	

Table 2.1 – Events and States (Moens & Steedman, 1998)

The relations between subparts of events lead to the notion of a single event structure, called a nucleus. The nucleus can be seen as a tri-partite event structure comprised of a preparatory process, a culmination and a consequent stage. Verbs can be classified according to which stages of the event structure they participate in. Verbs of motion such as run and bounce, have a preparatory stage, but no culmination or consequent; verbs of contact such as hit and kick have both a preparatory stage and a culmination (when the contact between objects is established); and verbs of change of state, such as break participate in all three stages. These groupings can be easily related to the verb classes from Levin. Moreover, this decomposition also meets well the needs of animation, which requires a detailed characterization of each stage of an action in order to instruct a virtual human.

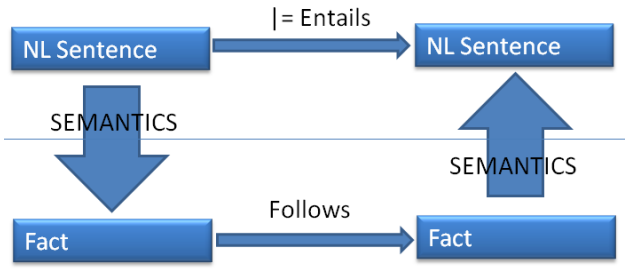


Figure 2.7 – Facts in the world representation

Vaculín and Sycara (2007) have done a good work on representation of events on an ontology. Events Composition and Recognition are embedded (Artikis et al., 2010). As Zarri, Dinarelli et al. (2009) worked on integration of dialog modality in an ontology of events where events looks like frames.

Semantic data and models representation (Fiure 2.7) can also be made directly at a perception level. The interpretation of the natural language grammar is done with a string parser using statistical methods like *Combinatory Categorial Grammar* (CCG) from Steedman and Baldridge (2005) and Bos (2005). These parsers are usually employed in Automatic Speech Recognition (ASR) sensor or following them to ensure the robustness of the recognition (Gaitanis, 2008). In this thesis, we don't consider the pre-processing of sensors and post-processing of actuators, we will limit our work to high level approaches.

2.5 Ambient Architecture

To design an ambient architecture for interaction, we need to define the basic requirements like extendibility of the architecture, discovery of components, distributed computing, composition, adaptation and interoperability. We study

ubiquitous computing, ambient intelligence and web services to take advantage of their intrinsic properties.

2.5.1 Ubiquitous Computing

Mark Weiser's classification of a ubiquitous computing system is based on two fundamental attributes: namely *ubiquity* and *transparency* (Weiser, 1993). Ubiquity denotes that the interaction with the system is available wherever the user needs it. Transparency denotes that the system is non-intrusive and is integrated into the everyday environment. In (Maffioletti, 2001) the requirements for a ubiquitous computing infrastructure are analyzed and the general idea of a middleware that provides the basic functionality for modelling interactive environment applications is presented. Current research in ubiquitous computing leads toward the development of interactive environments that enable the mobility of both users and computing devices. The vision of ubiquitous computing relies according to Coen et al. (1999) on the presence of so-called *intelligent environments* enriched by computers embedded in everyday objects like blackboards, tables, and chairs, and enriched by sensors able to catch information from the context. According to Maffioletti, these environments represent the spatial boundaries of applications integrated in our everyday context, they represent the physical space where the applications are placed and executed. It is distinguished between a *service dimension* that represents the number of services that is available in the system and a *device dimension* that represents the number of devices incorporated in the environment. Intelligent environments have a large number of hardware and software components that need to cooperate. They tend to be highly dynamic and require reconfiguration and resource management on the fly as their components and inhabitants change. The real power of the concept comes not from any one of these devices; it emerges from the interaction of all of them. The hundreds of processors and displays are not a *user interface* like a mouse and windows, just a pleasant and effective *place* to get things done.

2.5.2 Ambient Intelligence

Interaction architecture come back to hardware architectures to extend them to ambient intelligence and software engineering can help in particular the use of multi agents systems (MAS) to model and simulate entities of the environment. The need of intelligent components deals with artificial intelligence. Combining AI and pervasive architecture allows building intelligent architectures. Moreover, intelligence is integrated in the environment and agents may are parts of this environment (Becerra and Kremer, 2011). Two categories characterize Ambient Intelligence:

- A device-centric, technology-driven view: concerning physical interaction and processing capabilities for a new generation of networked devices (smart objects) and environments.
- A service-centric, user-driven view: exploring human, social and economic impact of new services made possible by the diffusion of ambient communication and interaction technologies.

In ambient intelligence, people abilities are enhanced through a digital environment that is aware of their presence and context and is sensitive, adaptive and responsive to their needs, habits, gestures and emotions. Human

environment involves real-world problems, which are characterized by incompleteness and uncertainty. Generally, we deal with information; some part of it might be correct, some part might be incorrect, and some part might be missing. The question is how to proceed with an elaborated reasoning process dealing with these information problems.

2.5.3 Web Services and Semantic Web Services

The concept of *web services* (Alonso et al., 2004) provides a new solution to solve the integration problem among heterogeneous application systems. According to Kim et al. (2004), the web services concept can be seen as a kind of standardized software technology that can integrate and share various systems. This web services concept has an advantage of flexibility by perfectly defining standard specifications for mutually sharable data among distributed systems. So the web services provide the advantage that they can transparently access any web servers in any place with any device and at any time. The web services architecture combines three essential roles: *service provider*, *service registry* and *service requestor*, see the following Figure 2.8.

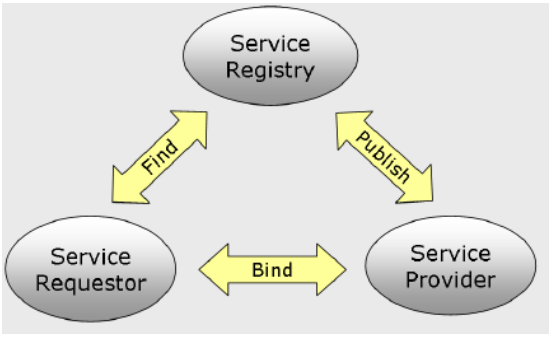


Figure 2.8 – Web Services Architecture

The *service provider* publishes the availability of their resources using WSDL, Web Services Description Language that defines the usage of web services and is used in order to describe the interface name, argument and return value of programs. The *service registry* is acting as a blackboard of services using UDDI, Universal Description Discovery and Integration, with the purpose of building a distributed global registry that could be accessed through the web environment. The web services standard architecture is composed of XML, UDDI, WSDL and SOAP (Simple Object Access Protocol), a protocol that enables users to mutually communicate their services under distributed environment with the well established XML. REST (Fielding, 2000) is also growing because it directly uses HTTP requests and responses to transfer the XML messages. Qi et al. (2008) explored the deploying and managing web services issues and brought solutions.

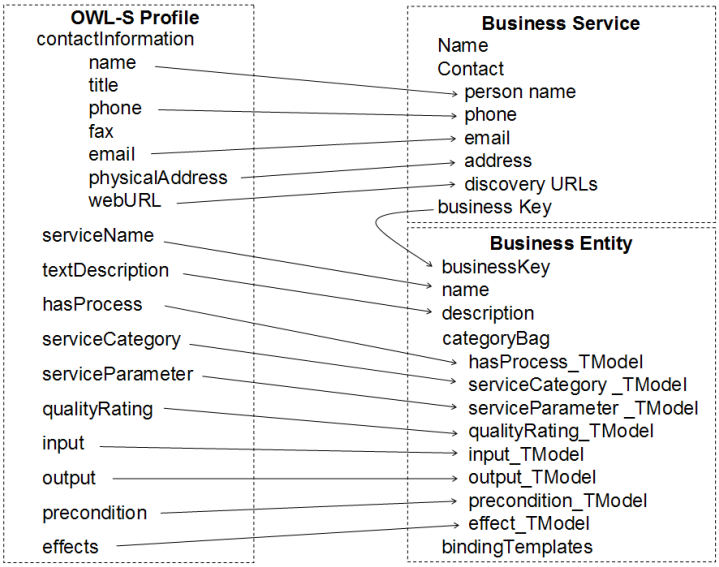


Figure 2.9 – The OWL-S to UDDI mapping according to Martin et al (2004)

The idea behind *Semantic web services*[11] (McIlraith et al., 2001) is that the semantic web should also enable greater access to services on the web. In (Martin et al., 2004) it is argued that semantic web services are developing the means by which services can be given richer semantic specifications.

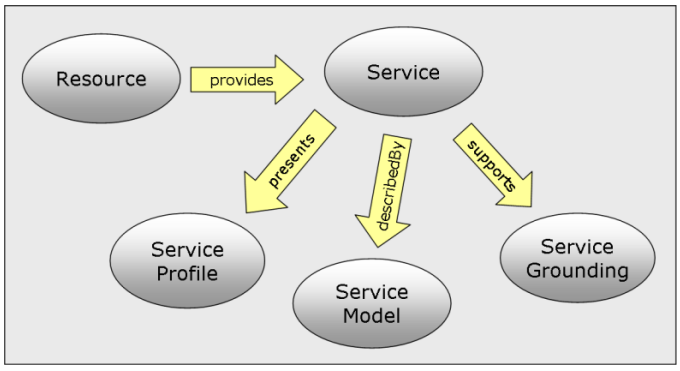


Figure 2.10 – Semantic Web Service Ontology

This richer semantics can enable fuller, more flexible automation of service provision and use, and support the construction of more powerful tools and methodologies. The semantic web should enable users and computer systems to discover, invoke, compose, and monitor all web resources that offer particular services. The DARPA Agent Mark-up Language program (DAML)[12] is about to develop and refine a service ontology that is called OWL-S. This OWL-based web service ontology is described in its release 1.1 at OWL-S Coalition in 2005. UDDI is widely used by businesses to register their presence on the Web by specifying their points of contact both in terms of the ports used by the service to process requests and in terms of the physical contacts with people that can answer questions about the service. Figure 2.9 shows the mapping from OWL-S

[11] Semantic Web Services Interest Group: http://www.w3.org/2002/ws/swsig
[12] DAML Semantic Web Services homepage: http://www.daml.org/services/owl-s/

to UDDI and thus integrates both approaches. Figure 2.10 shows the top level of the semantic web service ontology: a Resource provides a Service. This service presents the *ServiceProfile* that defines what the services do. This service is described by the *ServiceModel* which defines how the service works. And finally the service supports the *ServiceGrounding* that shows how to access it.

2.5.4 Semantic Grid or Ubiquitous Services

Another late-braking, interesting research issue is the so-called *Semantic-Grid*. It can be seen as the application of semantic web technologies to Grid computing[13], for example, Foster et al. (2002) propose an overview. The semantic grid[14] can be defined as *an extension of the current Grid in which information and services are given well-defined meaning, better enabling computers and people to work in cooperation*, (Wikipedia). The vision points towards a generically useable infrastructure, which is comprised of easily deployed components with flexible collaborations and computations on a global scale. The key to this is seen in an infrastructure where all resources, including services, are adequately described in a form that is machine-processable, to say, the goal is semantic interoperability. According to Zhuge (2005), the semantic grid is an internet centred interconnection environment that can effectively organize, share, cluster, fuse, and manage globally distributed resources based on the interconnection semantics. The semantic grid approach promises to provide effective middleware technology for ubiquitous user modelling. However, because of its early stage of development, the investigation and the integration are postponed to future work.

Kim et al. (2004) classify *Ubibots* (ubiquitous robots) in 3 categories: *Sobot* (software robot), *Embot* (embedded robot) and *Mobot* (Mobile robot). They conceived an autonomous agent architecture (called *Rity*) divided in 5 modules: perception, internal state, behaviour selection, interactive learning and motor. A first scenario presents the abilities of Rity to recognize and react properly to its master. The second demonstrates the possible transfer of Sobot to any other Ubibot so Rity becomes omnipresent. Many modalities are taken in account.

Very close to our idea and operational, Ha et al. (2005) have developed ubiquitous companion robot with a robotic service framework. Services are robotic Agents. The robot architecture comprises a planning service, semantic web services, environment knowledge base sharing, plan composition and plan execution services to make it automatically interoperable. For one process or one sequence to execute, they put in a very simple XML code file: conditions, plan and service functions to run.

As a foundation for our component design, we consider agent as a web service and we join the Agent-based Modelling idea of Macal and North (2006) where any agent is:

- *Identifiable*
- *Situated*
- *Goal-directed*
- *Autonomous* or self-directed

[13] Grid Computing uses the resources of many separate
[14] Semantic Grid homepage: http://www.semanticgrid.org

- *Flexible*, with the ability to learn and adapt its behaviour or connections.

We must consider agents who have attributes, behavioural rules, memory, resources, decision making abilities and adaptation rules, but in our case, memory and rules are stored into the component or another shared component, this memory component will also store social interaction, collaboration and group behaviour schemes. We also are in agreement on the work of Asaad and Mubarak (2004) using an ontology.

2.5.5 Composition and Adaptation

Composition approach now deals with applications, services and modalities use (Constantinescu et al., 2004). There are *Quantitative load* with lots of formats, languages, applications and services, and *Qualitative load* with more complex, sophisticated and intelligent applications (Sansonnet, 2005). Input and Outputs Modalities are now vocal speech, Graphical user interface, tactile screens, Lights, mouse pointer, recognition of voice, recognition of gestures, numerous actuators and sensors. Applications are more online, interactive, conversational, social and collective. Events choice needs composition and adaption depending on quality use.

Our architecture design must be ambient and dynamic so ubiquitous services are required. Our software agents controlling sensors and actuators will need a composition and an orchestration of services. As webservice components, they will bring an automatic services composition and discovery. It will permit lots of changes in a real time adaption of the functionalities depending on the faults, the energy saving, the load of work to do. This automatic adaption will permit a better reliable architecture. Robot will be able to use any database, actuators, software service on the network. During its navigation through network or real space, system will discover and use other available services. In addition, to achieve this dynamic adaption goal, we will use *Reconfiguration Scenarios*. Scenarios could be integrated to the models of the Ontological knowledge base. They could depend on context and qualitative or functional criteria checking conditional events ad using our fusion component.

Smith and Becker (1997) have worked on an ontology to build planning and scheduling systems, it is more concepts and instances ontology to model activities and constraints, it is a part of the OZONE framework. Sabouret et al. (2009) propose a dynamic composition for ambient intelligence environment.

Orchestration will be made by the application of schemes stored in the memory, the same orchestration system used to manage events and any of all schemes in the Ontology. Orchestration is closer than planning and execution components and will be very well explained in the next chapters. As good examples, we remind the work of Medjahed et al. (2003), Sycara et al. (2003) and Yu et al. (2006), they use semantic web language to compose and manage webservices. We will also use ontology to store any representation of webservices management schemes.

Acampora and Loia (2005) use agents and Fuzzy Mark-up Language (FML) to make services integrate fuzzy rules and decision for Context-Aware Adaptivity to fuzzy control an intelligent home environment. Idea of integrating fuzzy rules for taking decision is good but we will manage fuzziness (or granularity) of values and actions (spatial and functional distribution) in an ontology to manage

accuracy and execution of scenarios. Fuzzy logic can be also used to adapt strategies or schemes to possibilities and accept uncertainties.

2.6 Existing Architectures for Interaction

2.6.1 Introduction

Most often, HMI architecture or cognitive systems are manually built in a fixed manner, by combining a set of disparate modules, in order to solve a unique problem and validate the found solution. Even the starting assumptions and data are very strict. Generally, the application works well but there is no flexibility, and if a module has not the expected performance then it is replaced by another and the work of research is repeated again. Our approach is quite different. We want to find the architecture to be able to solve many interaction problems starting from general engineering requirements. This imply to model enough generic components able to work together to realize the function of a module. In our case, these components use semantics approach, knowledge representation language and previously examined standards to concretely determine the interaction architecture. Thus, we limit our architectures state of the art to architectures respecting these basic requirements. In fact, there are little semantic architectures because, until now, semantic components were reserved to web designers in HCI (W3C recommendation) and knowledge engineers, and carefully preserved by mathematical logicians or artificial intelligence researchers. Software and system engineering researchers appropriate this knowledge with the advent of ontologies of domain worn by the prospect of web 3.0.

2.6.2 HMI Architectures for Interaction

Krahnstoever et al. (2002) proposed a framework using speech and gestures to create a natural interface. The output of their framework was to be used on large screen displays enabling multi-user interaction. Fusion was done using a unification-based method. Cohen et al. (2002) worked on Quickset, a speech/pen multimodal interface, based on Open Agent Architecture, which served as a test bed for unification-based and hybrid fusion methods. Bourguet (2002) endeavoured in the creation of a multimodal toolkit in which multimodal scenarios could be modelled using finite state machines. This multimodal toolkit is composed of two components, a graphical user interface named IMBuilder which interfaces the multimodal framework itself, named MEngine. Multimodal interaction models created with IMBuilder are saved as a XML file. Flippo et al. (2003) also worked on the design of a multimodal framework, geared toward direct integration into a multimodal application. One of the most interesting aspects of their work is the use of a parallel application-independent fusion technique. The general framework architecture is based on agents, while the fusion technique itself uses frames. Configuration of the fusion is done via an XML file, specifying for each frame a number of slots to be filled and direct link to actual resolver implementations. Lastly, Bouchet et al. (2004) proposed a component-based approach called ICARE thoroughly based on the CARE (Coutaz et al., 1995) design space. These components cover elementary tasks, modality-dependent tasks or generic tasks like fusion. Finally, communication between components is based on events. The components-based approach of

ICARE has provided inspiration for a comprehensive open-source toolkit called OpenInterface (Serrano et al., 2008).

OpenInterface components are configured via CIDL XML files, and a graphical editor. Frameworks positioned at the data level, such as OpenInterface, Squidy and other data stream-based approaches, rely on the linear chaining of processing components.

	ICARE – OI	OpenInterface	IMBuilder/ MEngine	Flippo et al.	Krahnstoever	Quickset	Phidgets	Papier-Mâché
Architecture traits								
Finite state machine			x					
Components	x	x					x	
Software agents				x		x		
Fusion by frames					x			
Symbolic-statistical fusion						x		
Reusability easiness								
No programming kit					x	x		
Low-level programming/ API				x			x	x
Higher-level Programming								
Visual Programming tool	x	x	x					
Characteristics								
Extensibility		x	x	x		x		
Pluggability							x	
Reusable components	x	x				x		
Open Source	x	x						x

Table 2.3 Characteristics of different tools for creation of multimodal interfaces.

Table 2.3 summarizes the different characteristics of the systems described above: extensible systems (i.e. toolkits) have the potential ability to add other input modalities in a practical way. Pluggability refers to the ability of a toolkit to insert itself into architecture without having to rewrite everything. The other characteristics are self-explanatory. Regarding the table, each of these

approaches has components with specific characteristics and objectives. It is obvious there is a necessity to unite these components.

2.6.3 Cognitive Systems for Interaction

Cognitive systems for interaction are expected to learn through experiences, find correlations, create hypotheses, remember, and learn from the outcomes. They mimic the human brain's "structural and synaptic plasticity" (Gluck and Pew, 2009). The processing is distributed and parallel, not centralized and serial. With no set programming, the computing cores that the researchers have built can mimic the event-driven brain, which wakes up to perform a task. The goal is to create a system that not only analyses complex information from multiple senses at once, but also dynamically rewires itself as it interacts with the environment, learning from what happens around it. The aim is to solve problems related to navigation, machine vision, pattern recognition, associative memory (where you remember one thing that goes with another thing) and classification. Some similarities appear in building and modelling the architecture. Functional components are identical like memorizing, matching, selecting and acting. But the goal is different because, from this system, should appear a global intelligence and the system should be able to reproduce human functions in order to study the human brain and psychology. Cognitive architectures often look like the human physiology which has been built with a long period of time and even came from the human evolution.

Four good surveys (Dumas et al., 2009), (Sun, 2009), (Thorisson and Helgasson, 2011) and (Samsonovich, 2011) present and evaluate the current cognitive architectures. We retrieve the following cognitive architectures: 4D/RCS (Albus and Barbera, 2005), ACT-R (Anderson and Lebiere, 1998; Bothell, 2010), ART (Grossberg, 1987), BECCA (Rohrer, 2011), biSoar (Kurup and Chandrasekaran, 2007), CERA-CRANIUM (Arrabales et al., 2009), Chrest (Gobet and Lane, 2010), Clarion (Sun, 2007), CogPrime (Goertzel, 2009), CoJACK (Evertsz, 209), Disciple (Tecuci et al., 2009), Epic (Kieras, 2009), FORR (Epstein and Petrovic, 2010), GLAIR (Shapiro and Bona, 2009), GMU-BICA (Samsonovitch and De Jong, 2005), HTM (Hawkins, 2005), Leabra (O'Reilly, 1996), LIDA (Franklin and Patterson, 2006), NARS (Wang 2007), Nexting (Vashist and Loeb, 2010), Pogamut (Gemrot et al., 2009), Polyschemes (Cassimatis et al., 2004), Recommendation Architecture (Coward, 1990), REM (Murdock eta al., 2001), Soar (Laird et al., 1987; Young and Lewis, 1999), Ymir (Thorisson, 1999). Bach (2008) developed MicroPSI a broad architecture of agents for motivated cognition; it is also a cognitive system. SOAR has no declarative knowledge representation. ACT-R does not use semantics that's why Oltramari and Lebiere (2011) have extended ACT-R with semantic resources. They are not multi modalities and not progressive; most of them are only based on visual insights. In contrast, most of them contain a specific semantic memory mixing low level and high level data. Only Ymir and LIDA uses distributed modules.

Cognitive architecture for general intelligence like human intelligence specifies a goal but not a methodology. And existing methodologies in Artificial Intelligence have evolved to be incompatible with goal. We need method to integrate over boundaries into other disciplines and domains (Bach, 2008).

2.6.4 Multi Agents Frameworks

We searched for existing components or architecture able to work in a distributed way and exploiting information from knowledge base or ontologies. Rob (2009) surveys agent-based modelling and simulation tools. Sterling and Taveter (2009) propose an interesting book about agent-oriented modelling. Most of them are multi-agents which are integrated in systems for interaction with environment (Weyns et al., 2003; Daghan et al., 2007). Sakarkar and Upadhye (2010) explored software agents and ontology.

Bayardo et al., (1997) use InfoSleuth agent system to develop an open and dynamic environment. InfoSleuth can connect ontological concepts and execute matching services. Ontologies are not embedded into it.

Tambe et al. (2000) develop TeamCore, a multi agent's architecture integrating heterogeneous variety of agents and humans to work together. They manage the performance and preference to improve coordination between agents and human.

AMELI (Esteva et al., 2003) is a toolkit for the specification and verification of agent mediated e-institutions that based on a dialogical framework. In this framework, all observable activities from the agents are seen as messages in the context of a scene.

Gil Iranzo (2004) provides a set of agents negotiating in a semantic web architecture and the intellectual property rights and the agents knowledge are stored in a domain ontology. Environment and agents can be analysed statistically and get new knowledge from their interactions.

Heckmann (2005) worked on ubiquitous user modelling and provides UBISWORLD a framework for interaction between mobile (multi agents) and human in daily life. This offers a great opportunity to reach better adaptation with ongoing evaluation of user behaviour and sharing personal information and preference when querying a information in an application. AmbieAgents (Lech and Wienhofen, 2005) is an infrastructure which can run in a variety of configurations in distributed environments. Agents are also embedded in mobile phones.

Lin and Hsu (2006) developed iCare agents in mobile phone to store information about user activities and assist users in pervasive computing. Specifically, the iCare research seeks to create intelligent digital technology that can engage and excite people into active participation of desirable physical and mental activities at home that are considered healthy, creative, productive, educational and enjoyable.

In (Gateau, 2007), agents have the ability to reason about normative dimension of organizations. Agents have a representation of norms and can deliberate on respect or violations of norms. The multi agents system is in interaction with other services using multiple wrappers.

Mastrogiovanni et al. (2007) propose presents a distributed knowledge representation and data fusion system designed for highly integrated Ambient Intelligence applications. The architecture, based on the idea of an ecosystem of interacting artificial entities, is a framework for collaborating agents to perform an intelligent multi-sensor data fusion. In particular, they focus on the cognitive

layers leading the overall data acquisition process. Their work rests on ETHNOS Framework of (Piaggio et al, 2000).

Alain et al. (2009) developed GEORAL a multimodal dialogue application on JADE (Bellifemine et al., 2007) and DORIS platforms, and FIPA recommendations (FIPA, 2001; Briola et al. 2008). The application is dedicated to manage the speech interaction between agents and human-machine interfaces.

Billington et al. (2009) integrate the reactive nature of finite state machines and the reasoning capabilities of non-monotonic logics to produce intelligent autonomous robots. Their robotic player integrates vision, sound recognition, motion control and the reasoning to perform competitively as a player in a complex game with incomplete information.

Kameas et al. (2009) propose ATRACO systems which are dynamic compositions of distributed, loosely-coupled and highly cohesive components that operate in dynamic environment, Agents support adaptive task realization and enhanced human-machine interaction while ontologies provide knowledge representation, management of heterogeneity, semantically rich resource discovery and adaptation. Agents and ontologies are distributed and pervasive but the framework seems not enough flexible, and ACL and ontologies are not unified with behavioural templates. The architecture doesn't permit to understand the environment and reinforce the interaction.

Kapahnke et al. (2010) developed ISReal which is a comprehensively integrated application of semantic web technologies, semantic services, intelligent agents, verification and 3D graphics for intelligent 3D simulation. They focus on the interplay between its components for semantic XML3D scene query processing and semantic 3D animation service handling, as well as the semantic-based perception and action planning with coupled semantic service composition by agent-controlled avatars in a virtual world. They demonstrate the use of the implemented platform for semantic-based 3D simulations in a small virtual world example with an intelligent user avatar.

Subercase and Maret (2010 & 2011) provide a framework for programming MAS using semantic web technologies. Agents are programmed as extended finite state machine using SWRL (Horrocks et al., 2004) programmed a semantic agent with SAM a semantic agent model. Agent behaviour is then a part of agent's knowledge which makes it exchangeable with other agents.

To show the advantages of our approach, we will compare these systems or components with our architecture in Chapter 5.

2.7 Existing Development Platforms

JADE[15], Java Agent DEvelopment framework is a Java powerful multi agent platform which permits to design and control any level components of a system. It can be good for modelling and prove an efficiency of an architecture but not so suitable for Robotics.

[15] JADE homepage: http://jade.tilab.com

Robotics Platform is often seen as the hardware final system where the software code will be embedded so that the software is very much related to the platform but all software editors give an extended definition including the development and simulation environment with the hardware platform. Thus in reality, there is not only one platform, we will retain the platforms list of Table 2.3.

In addition to the platform, we have framework, there are like development platforms including ready-to-use software components and libraries.

Protégé[16] is a open source framework to design Knowledge Ontology using the OWL v1 et v2 languages, some additional plug-in like JESS[17] are available to add Reasoning. They are good tools to model our robotics knowledge but not to implement in a webservice composition and a robotics simulation.

	Platform Type	Product Examples
Software	Modelling behaviours	MS VPL, Aldebaran Choregraphe, JADE
	Software Development	MS Visual Studio, Gostai URBI, JADE
	3D Development	Solidworks, 3DS, Director, Xara3D
	Physics & 3D Simulation and Prototyping	Intel PHYSX, HAVOK, Webots, DirectX
	Components/Service Execution	DSS, CCR, .NET,...
Hardware	Target Hardware Embedded in a Robot	Meccano Spykee, Tux Droid, Car, Computer, Aldebaran Nao, ...

Table 2.3 – Platforms List

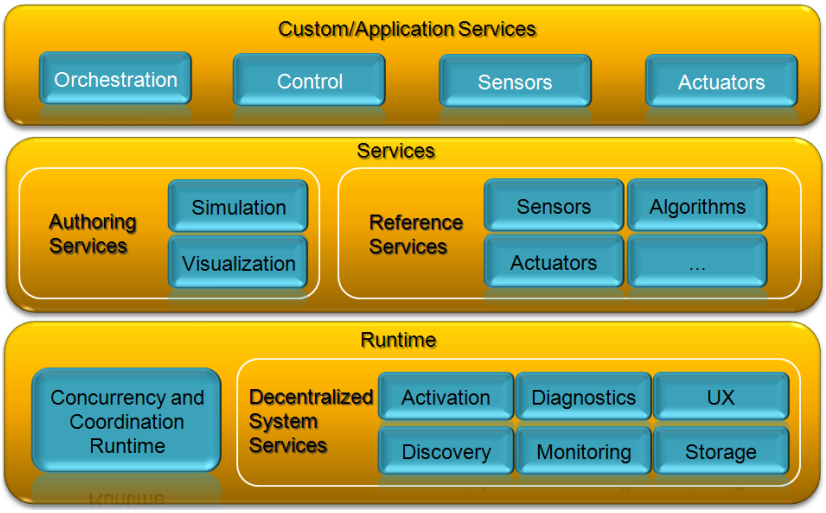


Figure 2.11 – Microsoft Robotics Architecture

[16] Protégé homepage: http://protege.stanford.edu
[17] JESS homepage: http://herzberg.ca.sandia.gov/jess

OpenHRP3[18], Open Architecture Humanoid Robotics Platform, is a Japanese software platform for humanoid robotics, and consists of a dynamics simulator. It provides various software components and calculation libraries that can be used for robotics related software developments. It is distributed under the Eclipse public license (EPL v1). We really think a robotics platform for simulation and real developments are needed like MICROSOFT Robotics Studio (physics, graphical & webservices composition), Urbi[19] or ALDEBARAN[20] Choregraphe (simple components in Python & Urbi). Urbi is a universal interface for Interactive systems, it's a language and a programmable platform to develop and run C++ components.

Microsoft robotics Studio parallel or concurrent applications can be developed in any languages using or not the .NET framework, a visual robotics programming language (VPL) is added to fast design a control system by graphically linking sensors, actuators or code boxes. It includes two programming systems: a decentralized system services (DSS) to manage services and a concurrency and coordination runtime (CCR) to execute services with concurrency. See the architecture in figure 2.11. They simulate and properly prepare the programs to be embedded in a target real-time OS. As Applications, GOSTAI, the French conceptor of Urbi, has made a tourist Robot able to assist users in their visit of the Sciences City of Paris. Since January 2009, GOSTAI is developing a robotics cloud computing to bring payable services to robots like web video surveillance compatible with Spykee robot. ALDEBARAN, a French company, is the hardware conceptor of the humanoid robot Nao not yet sold. Urbi and MSRS are good to design scenarios & events, parallel tasks, to use visual programming and simulate low level hardware parts. In the other hand, they need more high level intelligent software to realize intelligent applications.

We may add VRML, Virtual Reality Mark-up Language, is very good and can be easily coupled with a 3D generator. It describes very well the environment with agents on a web virtual space. Several others formats exist like 3DML, 3DXML developed by Dassault Systems and the ISO standard X3D. One of the most used and powerful 3D engine is Microsoft DirectX.

We must add that more and more systems use graphics multi cores (GPU) array available on graphics boards. These graphics boards are used as powerful parallel processing systems. For instance, Sharp (2008) has implemented a fast decision tree algorithm and a work of (Diamos et al., 2011). Some others like (Farabet, 2011) develop their own dedicated processor.

2.8 Conclusion

In this chapter, the study of these theories is very interesting for our work and is now mature. We have to draws our own definition and conclusion on how to integrate them and model components of our interaction architecture respecting formalisms and standards. This state of the art was necessary to understand all required basis: the multimodal interaction issues and context management; the definition of fusion and fission processes which we want to implement; artificial

[18] OPENHRP3: http://www.openrtp.jp/openhrp3/en/download.html
[19] Urbi homepage: http://www.gostai.com
[20] ALDEBARAN homepage: http://www.aldebaran-robotics.com

intelligence and symbol grounding problem to define reasoning components able to understand the environment; the description and knowledge representation languages and logic in order to model our architecture and the systems present in the environment in a semantic and more natural way; and finally the definition of ambient intelligence, ubiquitous services and pervasive context to integrate a modularly composition and orchestration mechanisms. We presented several existing interaction architectures to compare later to our work . We finished by a short tour around several development platforms useful to implement our architecture. In the next chapter, we will define requirements as a synthesis of this chapter and make our design choices. We will model our interaction architecture comprising semantic agent and services as basic components and a new environment knowledge representation language as connector. Following chapters will present this architecture in the pervasive context of ambient intelligence.

CHAPTER 3

SEMANTICS COMPONENTS FOR ARCHITECTURE

« No theory of reality compatible with quantum theory can require spatially separate events to be independent »
John Stewart Bell

3.1 Introduction

In this thesis, we want to define new active components to represent the environment, to reason on it and then reinforce multimodal interaction. After a general study of existing domains and components, we are now able to give our definitions of all information needed to model a compliant architecture and the knowledge representation in our components. In this chapter, we will define and show the design of our architecture and methodology.

Architectural Level	*Systems Development and Management* *Objective, Quality, Conception, Configuration, Deployment, Monitoring, …*	
Dynamics Level	*Coordination* *EKRL, Petri Nets*	*Adaptation* *Selecting, Filtering, Change*
Component Level	*Functional Components* *Perception, Reasoning (Decision), Action*	
Semantic Level	*Semantic Agents and Services Integration* *Query, Answer, Event Matching, Memory*	
Communication Level	*Network and System Abstraction* *Infrastructure, Connection Ports*	

Figure 3.1 – Architecture Modelling Levels

Figure 3.1 presents the five directions to develop and manage our architecture:

1. *Network and System abstraction* are the possible connections, communications protocols and links between components.
2. *Semantic agents and services integration* are system integration of component in an interaction application and information models to achieve one or several modalities
3. *Functional Components* like Perception, Memorizing, Reasoning and Action are the job operations or functional parts designed to reach a goal in one step of a global process. It needs composition and communication.
4. *Coordination and Adaptation* are a composition of services and agents working together to realize a cooperative operation. This operation can be an adaptation of the basic architecture or any other tasks to perform.
5. *Systems development and management* contains all objectives, functional and non-functional requirements, conception constraints, possible configurations, deployments and monitoring tools. It can manage present or future input/output components that will be integrated in the service composition. Generic components can be specialised in agent, service, sensors or actuators with a messaging system and the surrounding environment (objects and physical phenomena).

Our models architecture can be seen as a development platform integrating physical and logical components. All components will be represented and well defined in this chapter.

This architecture is composed of seven complex functional components with a specific operational role presented section 3.5.3. Each component at any level is built from a basic and generic component; this component is a webservice with networking, software and hardware capabilities. With these generic services, we will be able to design, recompose, reuse, improve and adapt our intelligent ambient architecture.

Our main purpose in the chapter is the design of an interoperable, scalable and efficient architecture composed of services. This design must take account of these following points:

- *Oriented systems* to manage hardware, real-time constraints and software required for intelligent components
- *Oriented ubiquitous* Webservice with an automatic Services Composition and Discovery
- *Ambient intelligence* to integrate and use others components for changing environment
- Centralized, Shared and diffuse architecture

This chapter proposes requirements to build architecture and systems included in the environment. It clarifies basic information concepts, knowledge definitions and environment representation. It describes modelling of our global architecture and components like services and semantic agents.

3.2 Requirements

3.2.1 Knowledge Functional Requirements

Req.1 Environment Modelling

This requirement concerns complexity of knowledge representation, knowledge fusion, reasoning, planning, execution control and learning in distributed configuration, algorithms to manage a high volume of data by expert systems, logics like First order Logic (FOL), High Order Logic (HOL) or Fuzzy logic (FL). Hughes (1989) explains why arbitrary higher-order functions are essential for decomposing programs into modular parts.

System is designed for acting in the real world and thus usually faces with a large set of high-dimensional input signals. In order to further process these high-volume datasets, abstraction processes that extract relevant information from the input data or events are necessary. (*Amount of information)*

There is an ongoing debate whether and when to generate symbolic descriptions of the relevant information in the input space. A drastic compression of the input data is needed, e.g., by the generation of abstract models for categorization. These models can for instance be beneficial in dealing with missing information or to introduce context within the symbolic domain. Furthermore, symbolic descriptions are the basic means of communication about the perceived entities or events and can be used to interact with other systems or humans (*Symbolic description*)

From a system's perspective, this calls for the possibility to exchange these descriptions in extensible representations between the different cognitive modules. Within every artificial cognitive system processes exchange information and work on representations of this data. As soon as learning and adaptation is intended, for instance to dynamically add new environment features and behaviours that are extracted by perceptual modules, representations must be able to dynamically evolve. Hence, data structures must represent information in an extensible way (*Extensible Representations)*

In addition to the benefits that openness provides for an architecture, the exchanged data items need to be self-descriptive and dynamically interpretable by components that manage features and knowledge about the exchanged data. This requirement implies the understanding of the situation (*Interpretable Representations)*

In order to provide a basis for a seamless communication with humans, improvement in interaction and communication capabilities of cognitive systems is extremely important. A basis for these capabilities and further advanced capabilities of cognitive system is thus some kind of a memory structure or a federation of different memories. This is due to the fact that memorization capabilities are prerequisites of learning and adaptation in cognitive beings, particularly if learning processes are active over a longer period of time. In cognitive functions, memories manage information and knowledge from various knowledge sources like spatial and contextual information, as well as scene and event descriptions. Hence, an important feature of memory systems is the ability to relate new information to already existing information. Additionally, accounting for the fact that memory is basically a limited resource, processes that distinguish relevant from irrelevant information and act upon that decision like forgetting or compacting are necessary. The processes in architectures for cognitive system require some kind of memory (*Curse of dimensionality, Learning and Forgetting*)

Req.2 Uncertainty of the environment

Uncertainty is in sensing and in the result of actions over the environment inherent to robots. It is posing serious challenges to existing methodologies for MAS which rarely take uncertainty into account. This requirement concerns the capability for our architecture to perceive correctly the environment, then the understanding and the interpretability of the environment *(Uncertainty)*

Req.3 Multiviews of environment

Consistently handle different and even opposite views of the world (Bessam and Kimour, 2009). To improve efficiency by limiting the search and recognition, and evaluate a minimum of rules to decide and act in complex environment with a complete or partial description where the interaction occurs. Multiple contexts exist like, for instances, context of user and other entities, or current task contexts *(Contextual Representation)*

3.2.2 Architecture Functional Requirements

The focus is set on the specification of hardware and software deployment, e.g., specifying the distribution of software components in architecture of systems. As the functionality of systems is different, the architecture is also typically specific

to each class or family of systems. The resulting challenge is to raise the level of *descriptiveness* and to facilitate the modelling of different components of the systems like structural composition or dynamic execution in a distributed system.

Req.4 Communication, Coordination and Adaptation

Consistency of a set of components concerns the design of functional software or hardware architectures, networking infrastructure, distributed systems (coordination, orchestration), distributed knowledge and processing. The use of multi-modalities and multiple machines to increase power, flexibility and redundancy is always profitable and may also improve capacities of sensing or action and security in case of failure.

Dynamic coordination is necessary in a technical architecture that exploits parallelism and provides an avenue for managing the dynamic behaviours that can be executed in the system. While the focus of sequencing is the mapping of serial behaviours to a synchronized series of system actions, coordination goes beyond this and provides structures for executing complex behaviours and tasks that depend on the runtime dynamics, for instance on the current perceptual state of the system or temporal aspects (*Coordination)*

In time and possibly in space of several modalities is important to analyse the correlation between events coming from each modality. If two events of two modalities occur in approximately same time (*Synchronization*)

Instead of predefined feed-forward processing chains, a CVS uses multiple sensors and recognition pathways to gather information about its environmental context. In order to build architectures for such systems, flexible means of managing the interconnection between different cognitive processes are required, for instance to realize hybrid architectures that allow for sensory bottom-up as well as actuators top-down processing (*Flexible Orchestration*)

A distributed computing system is a set of computer programs, executed on one or more systems, and coordinating actions by exchanging messages. Distributed configuration is considered when there are multiple autonomous systems interacting in a common environment, and especially if they have to cooperate in order to achieve their common and individual goals *(Distributed System)*

Transparency in a distributed system describes the degree to which the differences between a local and a remote interaction are masked out, e.g., for users working with an application on the system level or software developers utilizing a middleware in a specific programming language. While even more dimensions of transparency have been defined in the literature, most important dimensions of distribution transparency are Access, Location, Migration, Relocation, Concurrency and Failure (Tanenbaum and Van Steen, 2002) and their relevance to the integration approach in the context of its network functionality. (*Transparency*)

In contrast to some processes tightly coupled hardware systems that need real-time performance, the higher level perceptual components in our system may process the percepts in soft real-time. Because of the limited resources on the mobile or robotics platform, large parts of the processing therefore need to be distributed to external processing nodes *(Distributed and Pervasive Processing)*

A scenario can get worse as the number of components increase and/or whenever two sets of components are using the same media or network *(Noisy and limited bandwidth communication)*

As networking infrastructure becomes more and more commonplace, it is important that the architecture properly support working across Ethernet networks. Furthermore, remote process control and shared data across networks is desirable (*Networking support*)

Req.5 Integration, Monitoring and Testing

We need to integrate several methodologies that handle the subsystems of each individual system (extended to system of systems, objects, modalities and agencies in a cooperative setting) in a consistent manner; such that the integration becomes the most important problem to be solved, ensuring a timely execution of planned tasks.

Although this can be difficult to measure, the goal should be to provide convenient and flexible mechanisms to integrate the modules developed under this architecture with external libraries and applications, for example, customized software developed by third parties (*Ease of integration with external applications*)

The system should provide tight perception action feedback loops to react promptly to unexpected situations as well as higher-level planning for efficient use of resources over longer time frames. Plans should guide, not control, reactive components *(Reactive and Deliberative)*

3.2.3 Non Functional Requirements

Non-functional requirements which need to be considered in a suitable integration approach: Quality attributes on the implementation level like high availability and external as well as process constraints, e.g., with regard to legal requirements or the development process. Compared to the functional aspects, non-functional aspects place constraints on the realization or feasibility of certain conceptual design decisions within the integration architecture with regard to, e.g., performance, reliability or scalability.

Req.6 Modularity and Reusability

Multimodal system architectures reveal a richer set of classical modelling and development approaches. They are employed to build hybrid architectures combining data-driven bottom up with goal-directed and knowledge-based top-down processing. It can even be necessary to connect different processes at runtime, therefore yielding fully dynamic system architecture. The architecture should include support and guidelines for a modular code structure that allows only the applicable portions of the code to be installed, executed, or updated. This includes providing the support infrastructure for easily composing the modules to form a seamlessly integrated system (*Modularity)*.

A software architecture yields modular decomposability when it facilitates the comprehensible decomposition of a problem into a smaller number of easier sub problems that are still manageable by the integration architecture. In order to satisfy this constraint, the resulting partitioning should allow for independent,

parallel development and interconnection through a structure as simple as possible (*Modular Decomposability)*

This property of modularity relates to reusability of already existing building blocks of functionality in different contexts. An architecture satisfying composability shall allow to freely integrating existing building blocks in novel applications that were not foreseen during the initial development of these modules (*Modular Composability)*

If software favours modular understandability, it shall be comparatively easy for humans to understand the system-level functionality of a software module. It needs to be possible to understand the interactions of a module without getting to know all other modules within an architecture. In order to allow for better understandability, our assumption is that traceability of the dynamic behaviour of modules is an equally important requirement (*Modular Understandability*)

The aim of this property is to limit the impact of change. A software architecture conforms to modular continuity when a change in one of the domain modules yields only a minimal number of changes in other modules. This property poses questions of versioning and backward compatibility of interfaces. *(Modular Continuity)*

While continuity is concerned with the impact of a change, this property is concerned with the impact of failure. It states that the number of modules of an architecture that are affected by an abnormal condition within one module shall be minimal. This challenge relates to the question of combined critical dependencies introduced previously and the aim to achieve a high degree of robustness (*Modular Protection)*

It should allow dynamic reconfiguration of the system, including adding, removing, upgrading, or reconnecting components to the system. This includes the infrastructure for maintaining and updating the system over time (*Dynamic reconfiguration)*

An architecture must permit to execute multiple computations and parallelize the processing of decomposed modules, e.g., by using multiple processing units on a single physical computer system or by distributing the computations over a set of several networked processors or computers The integration architecture needs to support means for the development of concurrent software systems, because of the inherent parallelism in the application domain. It should provide higher level abstractions for dealing with these challenges than regular programming language constructs. The resulting programming model must support synchronous or asynchronous use (*Concurrency and Parallelism*)

This implies that modules can be reused in a variety of applications and should support working across different configurations, for example, as sensor type and placement is modified (*component or component reusability)*

Req.7 Complexity and Heterogeneity

We have complexity classes from theory. We build complex systems that do amazing, but often unpredictable, things. Is there a meaning of system complexity that spans the theory and practice of computing? Do current systems have to be so complex? Can we build systems with simple designs, which are easy to understand, modify, and maintain, yet provide the rich complexity in functionality of systems that we enjoy today? Managing complexity is one of the

most critical aspects of integration for robotic systems. One way is towards keeping much focused, simple devices. The trick for the system integrator, of course, is how to manage system complexity to make it acceptable. For robotic systems, unfortunately it can be difficult to provide precise measures of complexity. For the current context, we will generally think of two types of complexity: components level and task or system level complexity. In both cases, we tend to focus on two types of measures, namely, quantity or variety. Complexity may also refer the complexity of the environment (know/unknown, certainty/uncertainty, determinist/non determinist).

The *Heterogeneity* requirement is subjacent to coordination and adaptation. Components can be heterogeneous (network protocols, services, modalities, devices) and it can often reveal some communication problems. Cohesion and consistency make software, such as wrappers or converting systems, necessary.

Req.8 Portability

The standards compliance of an integration approach describes what standards are defined or supported by an integration approach. This aspect also relates to the openness of an approach *(Standard Compliance)*

This aspect describes for which hardware and software platforms implementations of an integration concept are available. In many cases, it is desirable to use the software developed for an application across different hardware (e.g., CPUs, GPU, robots, and embedded board) and software (e.g., operating systems) platforms. On the hardware side, this means that the software should be easily configurable for different robot configurations, such as sensor type and layout, motor type, and overall mobility. In the commercial sector, an important aspect of this is to support implementations that can run on embedded microprocessors with limited memory and processing power. A small dependency graph regarding external libraries and components is desirable for reasons of software complexity and maintainability *(Portability and platform independence)*

The system should enable access to the implementation of the architecture through a well-established *application program interface* (API), common language and protocols compatible with standards, and should provide flexibility in allowing customizations that respect the overall architecture design. *(Open and flexible)*

The ability to integrate additional services and modules as well as being itself integrable with other frameworks by concise definition of the integration interfaces and the used protocols is an important requirement for the overall approach. Ideally, the approach shall be based on well-known standard protocols and techniques that are beneficial for building distributed systems. Additionally, the framework shall support portability in terms of hardware platforms, operating systems and programming languages *(Openness)*

Req.9 User Preferences and Responsiveness

In an awareness application, the latency between visualization and head movement must match the user's expectations with regard to comfortable interaction speed. We also talk about acceptance criteria. *(Comfort, Responsiveness)*

Req.10 Scalability

Another requirement for the utility of an integration approach is the level of scalability. It should be easy to expand the system by adding new software modules and hardware components. Also, the overhead of supporting modular components should increase reasonably with the scale of the system. A system is said to be scalable if it can handle the addition of users and resources without suffering a noticeable loss of performance or increase in administrative complexity. Architecture needs to provide at least two properties of scalability: On the one hand, scalability is provided if an increased number of components reasonable for the given domain performing different tasks do not degrade overall system performance. Multi-core and Multi-thread architectures are needed to gain more computational resources. Conversely, the overall system performance shall scale up if the processing of a single component is distributed to a number of different components. Additionally, geographical scalability must be supported at least in terms of addressing the needs of the integration environment *(Scalability)*

Req.11 Reliability

It is an additional requirement introduced here that is obviously important, even in the context of experimental systems research. As a non-functional aspect, it relates to almost any of the functional aspects (*Reliability*)

3.3 Components Specifications

Our objective is to reinforce the interaction machine-human and machine-machine by means of multiple input and output modalities. The goals for our research can be divided into these main categories:

1. Definition of relevant information to represent environment
2. Modelling an interaction architecture able to understand what is happening in the environment with the fusion and fission processes
3. Definition of functional features to model which the architecture will manage
4. Designing components able to model the environment
5. Maintaining multimodal communication
6. Discovery of services for adaptation and ambient intelligence
7. Extending architecture to ubiquitous network and pervasive environment

To achieve these goals, we have to define the environment knowledge representation (EKRL). We have to model the components to manage this knowledge. Then we must integrate these components in an adaptable architecture capable of improving the ambient intelligence. Management of complexity and heterogeneity of systems will be solved by: reducing the amount of events managed by agents, forcing the use of EKRL for the communication into our architecture and develop a concentrator as a wrapper with virtual and real services (sensors, actuators and web services), and adapting the architecture by the discovery and composition of components respecting criteria chosen by user.

3.3.1 Information

To represent the environment, it is necessary to understand the nature of information present in the human environment. Wisdom (Know and know-how) and Behaviours are stored in an ontology and permit reasoning on Knowledge. For the perception and understanding, we are agree with Shedroff (1999), see Figure 3.2, *data* (covering creation, gathering and discovery) to *information* (covering presentation and organization), *knowledge* (covering conversation, narrative process and integration) and finally *wisdom* (covering interpretation of the meaning, evaluation and retrospection).

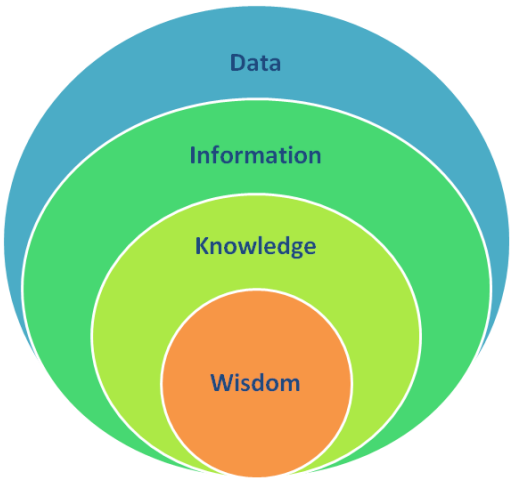


Figure 3.2 – Information Levels

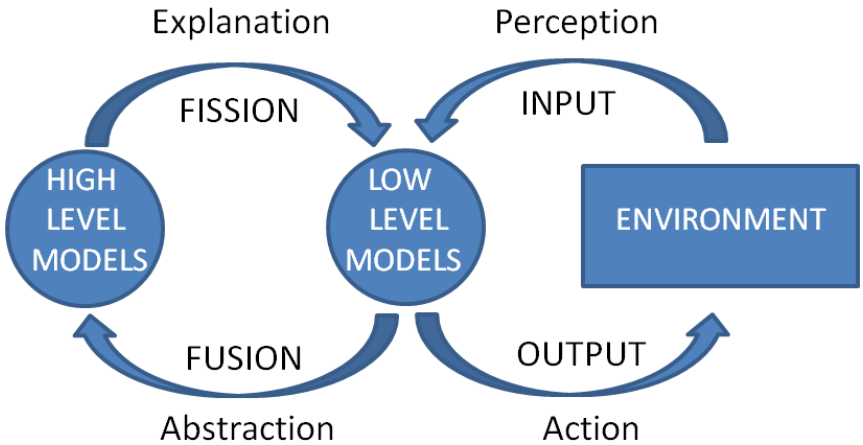


Figure 3.3 – Three facets of meaning (Complex and Primitives behaviours, and Environment)

Our definition of meaning and understanding is well explained by Malle (2004). The two processes of fusion and fission lie on the definition of the meaning (Russel, 1921; Landragin et al., 2004). To analyze and model the meaning of interactive behaviours produced or interpreted by humans there are (at least) three main facets we need to consider. First, behaviours refer to objects, actions, properties and relations in the physical environment – words have referential meaning. Second, interactive behaviours (low level) are produced intentionally to achieve effects on the interpreter (human or software) – interactive

behaviours have functional meaning. Finally, models of concepts or behaviours (high level), resulting of a composition of primitives models and concepts, have connotative meaning where connotation is defined as the name of a concept, a word or a model invokes in a person or an agent in addition to its literal or primary meaning.

The most important question in this thesis is how to model the environment. Environment is a composition of natural and synthetic systems. What has to be modelled? Figure 3.4 answers this question (Dourlens, 2011c).

Modeling	Information
Structural or static	Classes, Components, Composition
Behavioral	State, Transition, Scenario, Activities
Interaction or dynamic	Graphs, Communication, Interaction overview, Temporal
Architectural	Scenarii management, External view (meta level)

Figure 3.4 – Environment Modelling

We have presented in the previous chapter an interesting set of standard tools to model architecture, components, Interaction and Evolution. The evolution is toward environment or complex systems with EKRL (Figure 3.5).

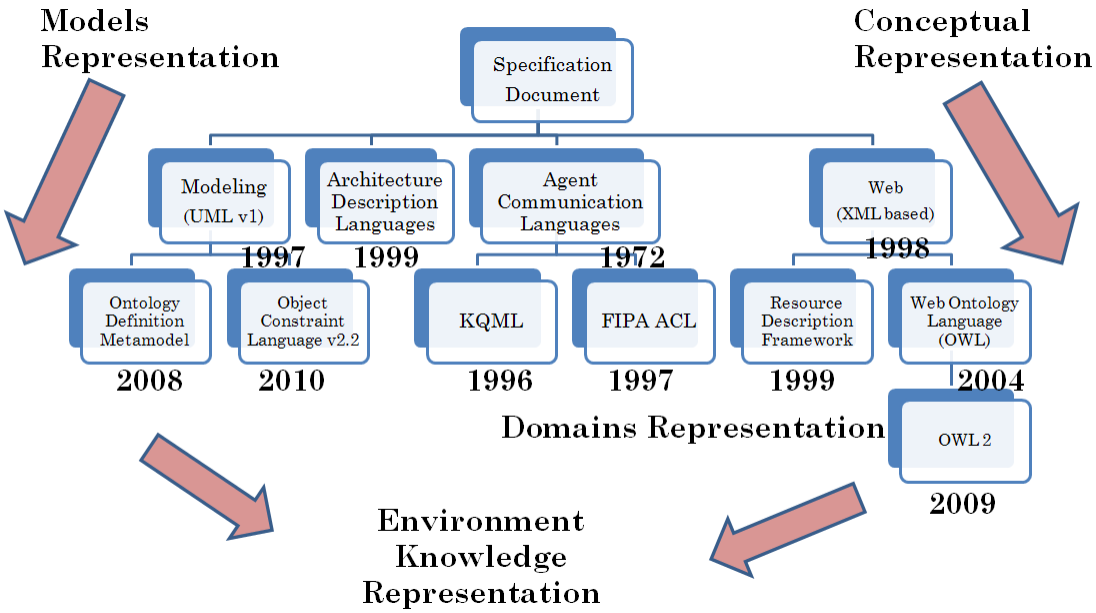


Figure 3.5 – Standards Modelling

Understanding the environment can be summarized in 3 questions:

1. What are the entities or symbols? We need to store and query conceptual information about entities in the environment.
2. What are the current actions or properties realized by these entities? We need to store and query behavioural information about facts.
3. What are the current scenario running or planned? We need to store and query scenario, programs or rules, possibly recursive (High Order Logic).

In his thesis, Heckmann (2006) talks about *User Modelling* and proposes these 4 observations:

- Information describes an ordered reality that varies across time and space.
- Information describes an ordered reality that varies from culture to culture depends on living places, concerned domains and used languages.
- Information describes an ordered reality that varies from person to person depends on family or close relationships and emotional aspects absorbed.
- Information describes ordered beliefs that can be "found" only by those with the proper observing skills and technologies.

We prefer to talk about *System Modelling* where system can be done by a human, a software service, an object or a machine. From these 4 observations, we can define our 5 basis assumptions:

- ✓ A1 "*Information is related to time, space or location, domain or language or relation, and social or technological networks*"
- ✓ A2 "*Information is related to our personal perception and memory in term of history*"
- ✓ A3 "*Information can be related to others information*"
- ✓ A4 "*All information can be valued*"
- ✓ A5 "*Information is related to queries*"

A1 is a direct assumption from observations, we assume all known resources are in the network, given by other parts or stored into memory storage service. In A2, we think about the type of existing information depending on the underlying goal which wanted to be transmitted. Information is often a narrative text (a news or an event) bringing a change of perception or state of a current situation and caring a meaning or a knowledge to reason on it. This new information can even bring negative information to create a misunderstanding in our mind or a change in our beliefs depends of our knowledge that permits to accept or reject it. In A3, from the nature of the language and perception, we know links exist to structure our thought, our discourse or any knowledge for a qualitative point of view. In A4, we assume some information can be more important or accurate, equal or lesser than another for a quantitative point of view. In the A5, we assume, in an information exchange (dialog or any communication protocol), Information can be enhanced depending on an asked question; this question is a need for a specific information or to establish a link between information. This last point is very important to reason on this information, to refine information or to reinforce knowledge.

3.3.2 Ontology

From previous assumptions, we have the basis to define items needed in our work. Definitions follow:

- ✓ D1 "*Information can be stored as a descriptive formal language L of classes of concepts C and classes of event models P*".
- ✓ D2 "*Classes and Instances are atoms and Attributes or Roles are formulae of our Description Language L*".
- ✓ D3 "*Information can be perceived from and sent to a social interaction or a technological network, a sentence or an assertion will be stored in a Predicate relation denoted* $p \in P \subset L$."

- ✓ D4 "*Concepts (static) or Instances (dynamic) are text or numeric values related to information in the natural language L*".
- ✓ D5 "*Valued information is related to a proper scale, granularity or uncertainty*".
- ✓ D6 "*Accuracy of information is related to independent scales and metrics*".
- ✓ D7 "*Role names (denoted r) are atoms of the predication and syntactic relations between $r \in R$ and the predicate $p \in P \subset L$*".
- ✓ D8 "*Arguments A are specific concept instances associated to a role.*"
- ✓ D9 "*An atomic event e or action a is a narrative predicate relation containing roles and arguments*" *$L((R_1,A_1)\ (R_2,A_2)\ldots\ (R_n,A_n))$ where an event e is a predicate p from P in the language L.*
- ✓ D10 "*A state denoted $s \in S$ or a scenario (composed of several states) denoted $x \in X$ is a specific predicate p to link several events (fusion) and/or actions (fission) representing a graph, a scheme or a program*"

We will define an ontology with two kinds of classes: *models of events* and *concepts*. In opposition to OWL (section 2.3.3), we will use the terms: *concept* for the classes of concepts, *models* for the classes of models, *property* for a property of a concept, a *role* for a property of a model, *concept instance* for an instance of a class of concept, and *fact* for an instance of a class of model.

3.3.3 Reasoning

Now, definition permits to present information that will give the memorization function. Information retrieval will be also explained with our fast tree matching using a deep link between questions and templates predicates.

- ✓ D11 "*Rules with antecedent (precondition or premise) and consequent (post condition or conclusion) are also stored in ontology as a specific model of events (a predicate p) to evaluate answers to a request, to query contexts of a situation or actions to perform. We call them Rule Models*".
- ✓ D12 "*Situation x can be defined as compound of states (being like that), activities (doing that), accomplishments (narrative), achievements (goals reached) on a time period*".
- ✓ D13 "*Context is a restrictive field of interaction in time, space and domain (subject of talk)*".
- ✓ D14 "*Meaning m or situational context x is intrinsic of the ontology structure and events filling only if ontology is enough consistent*".

Rule models are used by the inference engine to perform *matching* with the predicate logic. *Deduction* process uses axioms and concepts (definitions) to produce tautological results in the choice of instances already written in the premises (consequent of the law); it can be used to add new concepts and new instances. *Induction* process tries to find or to generalize a law from a given meaning to new instances but most of the time it's an approximation.

Varieties of meaning exist (Millikan, 2004). *Meaning of the situation* must be quickly extracted to take a reactive decision. This meaning is very important to obtain a correct interpretation. Meaning of the situation requires developing a description of the current relationships among entities and events in the environment. Meaning of entities is easy to find because of the subsumption relationship between concepts. The ontological storage of events and extraction of the meaning in the memory of agents to understand what is happening is very

important for the interpretation. A meaning function that maps elements of the symbolic/conceptual algebra (i.e., nodes in an abstract tree that is a part of an ontology) to their meanings in the semantic algebra: relationships between nodes (concepts or events) in a role of a predicate (event model) in KRL. Each event is said to denote its image under the meaning function. In practice, the meaning function is specified by a collection of so-called valuation functions (roles relationships), one for each symbolic domain or contextual information defined by the abstract syntax of the language. Not all functions can serve as a meaning function; the function must be a homomorphism between the symbolic algebra and the semantic algebra. This is just the technical condition that constrains the meaning of an abstract syntax tree node to be determined from the meaning of its sub nodes. It can be stated more formally as follows:

Suppose f is a meaning function (the common function of each event evaluation can be an *inference* function accomplished by an inference engine) and c is a concept or an event, with children $c_1, \ldots, c_k$. then

$$m_c = f(m_{c1} \ldots m_{ck}) \qquad \textit{(Formula 2)}$$

where $m \in M$, f is a semantic formula on each role R_i of c and determined by the class of c.

The reason to restrict meaning functions to homomorphisms is that their structure-preserving behaviour greatly simplifies reasoning. This design choice accounts for a property of denotational semantics we call *compositionality* that is summarized by the motto "*the meaning of the whole is composed out of the meaning of the parts*." A key consequence of compositionality is that the meaning of the environment remains the same when one of its events is replaced by another event with the same meaning. Compositionality also facilitates the implementation of programming systems. The core symbolic processing procedures of interpreters and translators based on denotational semantics have a natural recursive structure that mimics the recursive structure of the valuation functions and the abstract trees they manipulate. Moreover, if meanings are produced by a generic software component, then compositionality is also applied to the modelling of the system's structure which is composed of these software components. We can now call them *semantic components*.

Semantic contexts can be physical, lingual, social and cultural as said by cognitive grammar researchers. These contexts are essential to understand the human environment and act in accordance with human norms (defined by human).

3.3.4 Models

To realize situated interaction, we will store all models in our memory component including discourse models, domain models, task models and available media models. We use OWL semantic relationships between concepts in predicate roles. XML W3C Standards compliant appliances or sensors are already designed and operational (Sirin et al., 2005). Our roles allow defining source of information directly in the event. They include systems that automatically process input like language, gesture, or graphics and that render multimodal output. There are systems that can automatically design graphics from structured data or coordinate the layout of multimedia content in time and space.

- ✓ D15 (Ubiquitous System) "*A ubiquitous system consists of a heterogeneous set of services from any agents (computing or not), a set of supported tasks, and some infrastructure on which all agents rely on in order to carry out their tasks. These services, web or not, are discoverable.*"
- ✓ D16 (Service Model) "*A Service model is a knowledge source in a system which contains explicit assumptions on all aspects of the agent that may be relevant to the behaviour of the system. These assumptions can be separable by the system from the rest of the system's knowledge."*
- ✓ D17 (Service Modelling Component) "*An agent/service modelling component is that part of a system whose function is to incrementally construct an agent model; to store, update and delete states and roles; to maintain the consistency of the model; and to supply other components of the system with assumptions about the user."*
- ✓ D18 (State and Action Model) "*A state action model is a template predicate respecting D9"*
- ✓ D19 (Scenario Model) "*A scenario or activity model x is a graph stored as a predicate p respecting D10."*
- ✓ D20 (Human-Service Interaction) "*Human service interaction is defined as the interaction of a human with the environment and with artefacts which is aimed to accomplish a goal. Within this process the system acquires implicit inputs from the user and may present implicit outputs to the user through the environment."*
- ✓ D21 (Service Discovery Model) "*D15 and D19 implies a Service discovery model is used for an exchange between agents or more precisely to extend their abilities to sense or to act in cooperation with others agents or services*"

We notice D19 is an arranged definition from Schmidt (2003) about the extended concept of *implicit Human-Computer Interaction.*

Until now, we expressed a modelling of ubiquitous domain-independent component. We will see further how these definitions impact the context of Robotics interactions. Here, we extended the notion of Human-Machine Interaction or Human Robot Interaction to Services Interaction in an ambient network.

- ✓ D22 (Situation-Awareness) *Situation-awareness denotes the combination of agent-adaptivity, resource-adaptivity, context-awareness, service-discovery-awareness and memory-awareness.*
- ✓ D23 (Situated Interaction) *Situated interaction denotes the explicit, implicit and indirect interaction between an agent and an intelligent environment (or systems and their interaction services), that take situation awareness into account.*

Note that our definitions don't distinct user context, environment context and system context. It's because in our formalism of predicates we don't distinct the services: human, robot, software agent or environmental objects. But anyway, they are taken into account in the concepts and instances associated to events and then can be differentiate in term of question or requests to our memory component. In the same way, we won't distinct virtual objects or systems from physical one so they will work without any difference. Be sure our query system will differentiate them by their attributes or composition.

In addition, our definitions permit the interaction in simulations, real world or virtual reality. Furthermore, it is already sufficient to uniquely identify everyday objects and map them to their virtual counterparts which do the information processing part for the original object. In ambient network, now lots exist like sensors (Temperature, Light, Mouse click, Application events, Heartbeat sensor, vocal recognition, alarm, motion camera, energy level) and services (Heater, computing tasks, clock, coffee machine, mails vocal reader,…).

3.3.5 Behaviours

We will see later the impact of D18, D19 and D20 on Markovian Decision Processes (Papadimitriou and Tsilsilis, 1987; Putermann 1994, Kaelbling, 1998) to reason and to make planning in interaction.

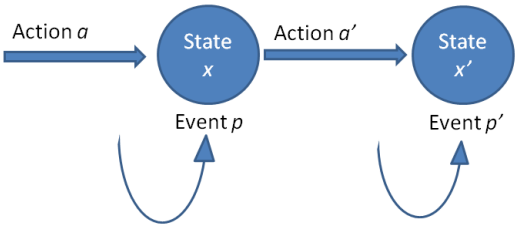


Figure 3.6 – Markovian Decision Process

Situation can be explicated by a composition of events or states classified on Table 3.2. To apply rules on a precondition is the same thing as to apply an evaluation of a predicate function on a list of arguments in events instances. As defined in D9, in a perpetual changing environment, a state denoted *s* is like a request on happened events *e* for a chosen instant time with possibly the use of constraints. An action *a* is always applied before the state change and the event appears after the change once a sensor has detected it. We can choose common representation for action, state, event (models) or fact (happened event). This will be a predicate of roles and arguments that we can called a primitive state *s*. If we want to know all of what happened in the environment at a chosen instant time, we must memorize that information and query this memory. A composed event will be a higher level fact deduced from the presence of lower level (primitives) events. These new events must be produced at the arrival of new primitive facts. Our approach is quite similar to Hierarchical task network (HTN) approaches introduced by Nau (1994), later by Ghallab et al. (2004) and recently Kaelbling and Lozano-Perez (2011).

(Sansonnet et al., 2005) showed we need five criteria of feasibility:

- *Representation*: formal representation of desired features
- *Reasoning*: heuristic to reason on formal representation
- *Dialog management*: User-system management module working in cooperation with the progress of a task.
- *Interpretation*: ability for the agent to understand the meaning of interaction (language or actions) of the user in all their dimensions : provision of information and demands, but also its mental states (what he is thinking of the system), emotional (e.g. angry with the agent or in the other hand non-motivation)
- *Production*: Ability for the agent to express pertinent information and behaviours to be understood and appreciated by the user.

Some experiences are defined to evaluate the influence of the presence (or absence) of a software or hardware function on the User-System interaction, under four main evaluation criteria:

- *Efficiency*: Efficient performance measure of user-agent in the progress of a task
- *Usability*: Is it easy for the user to understand how works the system and how to easily use it?
- *User-friendliness*: User feeling about pleasant to use (attraction, engagement, aesthetics, comfort).
- *Believability*: User feeling on what the agent can understand and solve his problem and help him.
- *Trust*: 'User feeling on what agent will behave as friendly and cooperative entity.

For Hangos and Cameron (2003), during the model design phase, the behavioural model of a system will be determined. It will represent the functional relationships between inputs and outputs, which were assigned in the input–output diagram. The model that will be used will consist of differential equations and additional equations and the assumptions under which the model is valid. A model can be formulated top down by its structure, the functions and the parameters. The following design activities can be distinguished:

- determine the assumptions
- determine the model structure
- determine the model equations or laws (i.e. strategies or scenarios)
- determine the model parameters
- model verification
- model validation

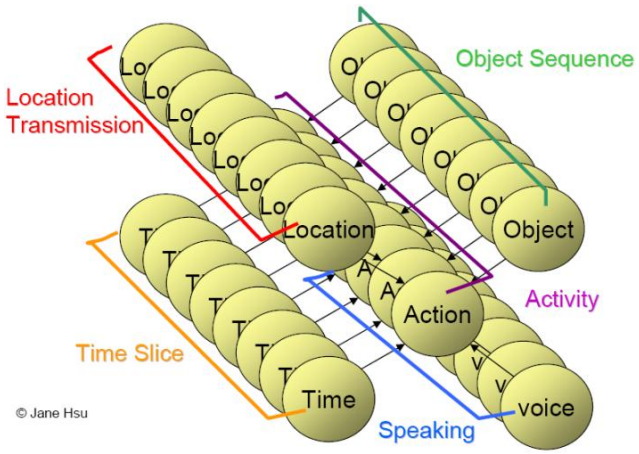


Figure 3.7 – Sequential Activity Model from (Hsu, 2008)

Jane Hsu from the National Taiwan University designed a context Aware activity recognition (Lin and Hsu, 2006) integrated in a mobile PDA to support people in their daily tasks (eating, sleeping, bathing, dressing, phoning, shopping, cooking, housekeeping and managing medication). The objective is to monitor and recognize the activities of people within the environment in order to provide timely support for safety, comfort, and convenience. Activity and Behaviour Modelling parts of her system are:

- *Event*: Sensor Signals (Device-dependent sensor model at low level)
- *Scenario of actions*: Temporal Event Sequence

  - HMM: Hidden Markov Model
  - GMM: Gaussian mixture model
  - DBN: Dynamic Bayesian network
- *Activity*: Evidential Reasoning Model
  - Bayesian networks
  - Case Base Reasoning

The sequential activity model links events to desired actions like shown in Figure 3.7. She used a SOUPA ontology built with OWL is also used to manage categories: Agent, Space, Event, Action, Time, Devices, Schedules, and so on. Voice in the figure is just one of the output modalities we can manage. This architecture is very interesting to realize a *Fusion* function in our own model even if we don't want in this thesis work on low levels of perception and action. Our design should also rely on input services able to extract sensors information, output services able to control actuators and software services. This architecture is limited to some activities, a house and the monitoring of users. Components are not distributed. For example, no concept of agent is used. The representation of the environment is quite simple limited to objects and locations. The meaning of situation is manually analyzed. Components are rigid and the architecture can't evolve. Modalities are GPS, Voice and the PDA screen. A generalization of this work should be taken into account in a more flexible architecture.

3.4 Environment Knowledge Representation Language

3.4.1 EKRL Grammar & Syntax

Environment Knowledge Representation Language (EKRL) allows the represention of what is happening in the environment during execution, in terms of meaning or interpretation. The objective of EKRL is to communicate, store and reason on perceived events by respecting the previous definitions of given in the previous sections. EKRL is a connector for our components. EKRL is a semantic formal language L that can describe events in a narrative way very close to a natural language.

EKRL is solely used to build event messages (to inform or to query agents) and store facts directly in their class of models in the the memory of agents. The formal system is composed of the formal language based on variable arity relations in predicate logic (event frames). It enables us to perform semantic inference in order to extract meaning from the situation. Ontologies are useful and powerful structures for storing events and extracting meaning. Inference systems may use models to match the ontology instances. In EKRL, frames are predicates with slots that represent pieces of information. A slot is represented by a role associated with an argument.

A predicate P is a semantic *n-ary* relationship between Roles and Arguments and represents a simple event e or a composite event e'; it is denoted by the following formula:

$$P((R_1 A_1)\dots (R_n A_n)) \qquad \textit{(i.e. Formula 1)}$$

where R_i is a predicate role and A_j is a list of arguments. Roles R_n are the possible roles (dedicated variables storing arguments) in the event model and A_n

are the possible combination of values or instances of concepts in a stored model or fact.

Figure 3.8 shows how knowledge (conceptual and behavioural information) should be represented with relationships between concepts, events, references (facts), contexts and dictionary definitions. Concepts and events (verbs) are extracted from a dictionary to form facts of references directly stored in an array of roles and arguments. Each model of event or reference is linked to a query because in a question, we often use the verbs and concepts corresponding to existing facts previously memorized.

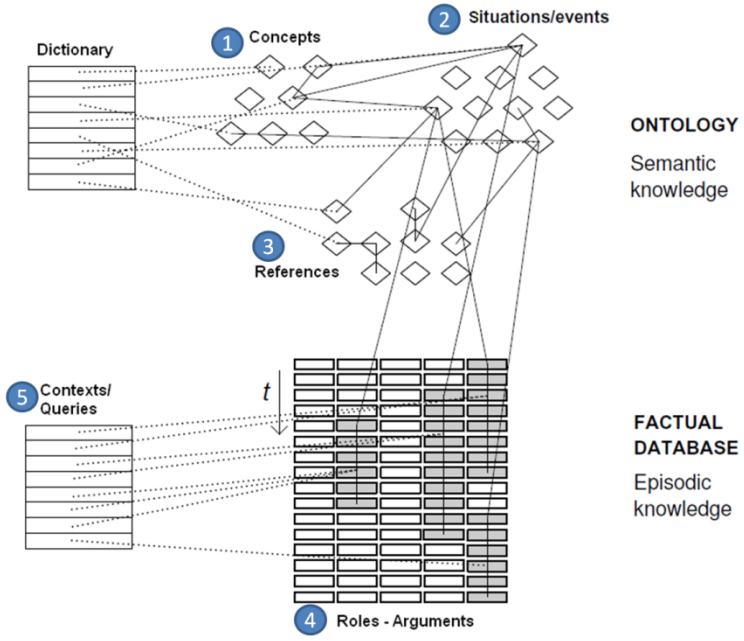


Figure 3.8 – Knowledge base representation

<RootPredicate>:<PredicateName> (*name of the event model*)
Natural language: '<Predicate Description>' (*optional natural language description*)
<Role1><Arguments1>
<Role2><Arguments2>
<Role3><Arguments3>

Figure 3.9 – Event Model Description

Figure 3.9 shows a sample model written with the EKRL syntax. The list of all roles is part of the Meta ontology of the the memory of agents. The term *Role* can be, for example, OBJECTIVE, SOURCE, BENEFICIARY, MODALITY, TOPIC, CONTEXT, MODIFIER, DATE, and so on. Event models are models of predicates and instances of predicates specific to a situation.

Exist: Available Service
    OBJECT: composition
    SENDER: services
    DATE1: date time
    LOCATION: location

Figure 3.10 – "Exist:Available Service" event model

Figure 3.10 shows a predicate model of an "available service" event. "Exist" is one of the root predicates of the model's ontology tree (along with MOVE, RECEIVE, BEHAVE, OWN). "Exist" is a general event model of the event model

ontology expressing a creation or discovery of anything. OBJECT (objective), SENDER, DATE1, DATE2 and LOCATION are the roles of this predicate. This event model will let agents know new services are connected and available for composition. This event will be sent by services to inform about their availability. In the instances of this model, agents will be able to know the service name, start and end date of existence, and location of the services.

event: rootpredicate ":" predicate frame

frame: frame role args

role: *Meta-concept*

args: NIL | args arg

arg: *Concept* | string | number | operator "(" args ")"

operator: *Meta-concept*

Figure 3.11 – EKRL Grammar of an event model

A formal grammar is required to achieve this task. *Meta-concept* and *Concept* are parts of a Meta ontology and Ontology of concepts (Figure 3.11).

The notion of operators exists in Description Language to present relationships between several arguments or properties linked to the same role for a concept or an individual, normally to refine the meaning of the event. We will use the operators defined in Table 3.1.

	Operator	Logical Operation
A	ALTERN (alternative)	Disjunction (OR logic)
E	ENUM (enumeration)	Distribution (Generalization)
C	COORD (coordination)	Conjunction (AND logic)
S	SPECIF (specification)	Attribution (Specialization)
CA	CAUSE (strict causality)	Necessary and sufficient
RE	REFER (weak causality)	Necessary but not sufficient
GO	GOAL (strict intention)	First argument necessary to realize second argument
MO	MOTIV (motivation, weak intention)	First argument not necessary but second is sufficient to explain the first
CO	COND (condition)	First argument will be realized if second can be realized

Table 3.1 – Binding Arguments Operators

ALTERN and COORD are standard OR and AND logical operators.

ENUM is a list of items. In a formal semantics, we can note:

$$(ENUM\ e_1\ e_2) = (e_1 \wedge e_2 \wedge \neg\ (COORD\ e_1\ e_2)) \qquad (Formula\ 3)$$

$$(SPECIF\ e_i\ a\ b) = (SPECIF\ ei\ b\ a) \qquad (Formula\ 4)$$

Formula 3 shows a relationship between possible concepts or instances (e_i) in an argument of a predicate. Concepts or instances are required to satisfy this relationship between predicate and structured arguments, they can be used separately. Formula 4 is a specification link. Others operators (greys) are useful to link several arguments as cause, intention, condition to realize an event.

Name: Move:TransferOfServiceToSomeone	
Natural language description: 'Transfer or Supply a Service to Entity'	
MOVE SUBJET	ALTERN(*Actuator29,Actuator30)*
OBJECT	*Service:Location*
SOURCE	*Server:Location*
BENEFICIARY	*Agent:Location*
MODALITY	COORD(*Composition,Execution)*
TOPIC	*UDDI*
CONTEXT	COORD(*Home Network,Emergency)*
MODIFIER	*forced*
DATE-1	*10th September 2009 / 10:00*
DATE-2	*10th September 2009 / 11:00*

Figure 3.12 –Move:TransferOfServiceToSomeone event model

Figure 3.12 shows the "MOVE:TransferOfServiceToSomeone" predicate of the description of an event model to supply a particular service or component (to control actuator29 or actuator30) to an agent. UDDI server will allow the webservices composition at the defined time period.

Our interaction models have the 4 main levels presented table 3.2.

This information contains multiple classifications considering the different possible roles: in time, by actors, by concepts domain, by objects, and so on.

A meta model is an intrinsic n-ary predicate P where n is the number of roles R that permit to add, modify or remove any concepts, any models or any sets linking concepts (sub tree) or models together (scenarios or schemes). This type of predicates is used only by the ontology manager; they can be associated to management events like a demand of storage of a concept or an instance in case of refinement.

Composed models or schemes are special predicates where arguments can be predicates. Thus these models link several predicates and after instances of each predicate are stored, we can query or evaluate a scenario. Rules of decision or reasoning will be also represented by one or several predicates.

Rules of Description Logic with the use of linguistic variables and values in our scales will be very close to fuzzy description logic. Since many concepts that are needed for *Intelligent Systems* lack well defined boundaries, or precisely defined criteria of membership, we need fuzzy logic to deal with notions of vagueness

and imprecision. This offers a motivation for a generalization of description logics towards dealing with imprecise and vague concepts. What people should also think about for intelligent systems is multiple viewpoints of the data. This will lead to subjective (as opposed to objective) intelligent systems. And thus, our reasoning and querying systems will be able to adapt scales to find the best approximate solution to a current problem without changing the logic.

Level	Description	Formalism
Meta	Management and execution scenarios, evaluation and adaption of the agent like choosing the program	$M: A, E, C, CS \rightarrow A', E', C', CS'$
High	Complex scenario (reasoning, dialog, learning, awareness, planning) like a software program.	CS: list of C in a graph or scheme order
Middle	Composed events and services, simple scenarios, situational and operational facts like functions of a software program.	C: list of linked predicates in a graph or scheme order
Low	Fixed simple events and actions realized by a service	$E: x \rightarrow x'$ past $A: s \rightarrow s'$ future where x is a environment state, s an agent state, *E an event function* and A are predicates.

Table 3.2 – Model Levels

The implicit nature of knowledge applies not only to common sense knowledge, but also to a wide variety of expertise and skills we possess. Such domain-specific knowledge is often represented as procedures, rather than facts and rules. In addition, areas which rely on procedural or implicit knowledge such as sensory/motor processes are much more difficult to handle within the Symbolic AI framework. That's why we introduce this work a brand new approach permitting symbolic AI to perform complex scenarios and actions. We will see in the chapter 5 how we manage events to control the composition of services, reactive architecture and operational tasks like awareness and focus.

3.4.2 Modelling Entities

Environment is a set of entities like human, agents, robots or objects. They are linked to different attributes. For instance, *cars* is a concept class with *subClassOf(terrestrial transport)* relationship like *trains, buses* or *trucks* classes. Instances of concept *trains* can be *Orient Express*. Metaconcepts are used to build all ontologies (nodes properties and relationships). The root of the Concept ontology is named *Thing*. Each node of ontologies have a unique reference (RDF *about* term), a comment (RDF *comment* term) and relationships like *owl:complementOf, owl:oneof, owl:unionOf, rdfs:subClassOf, rdfs:EquivalentOf or owl:intersectionOf*. These give us the possibility to directly insert OWL files into our ontology of concepts. Meta and Concept ontologies are

not bounded to external entities in the environment but they also contain internal parts of the designed system interacting with the environment.

One key idea must be highlighted here. Meta and Concepts ontologies are built with OWL v2 formalism but we have limited them to a tree structure and not a complex graph to ensure a fast search browsing of concepts and instances and especially ensure graph closure. Direct relationships between a class and a subclass are only allowed. So now questions are: - Where are others more complex OWL relationships? And - Where are the logical rules to infer the properties of these concepts? The answer is in Section 3.4.3, they are parts of the facts description and they are simply EKRL event models.

To improve the system and manage fuzziness later, we have added four values to each ontology node. More precisely, scaled values attached to concepts of some nodes: *access control list* to manage access rights, *rank* to order concepts in list, *strength* for % of truth for the inference system and *status* to active/deactivate it.

3.4.3 Modelling Behaviours

Ontology of concepts of the previous section is presented as a tree structure. The same relationships can exist between models. That's why we design our ontology of event models exactly like the ontology of concepts. The main difference is that the node structure of models respects the EKRL grammar. So models (our predicates) are much complex and each role into the model contain a formula on concepts (case of an event instance) or other models (case of a rule model). They are frames with slots where slots are our roles.

Behaviours are modelled by a list of events. All event models and past events (model instances called "facts") are stored in the ontology of models. EKRL root-predicates correspond to all possible events happening to components. Under these root predicates, subclasses of predicates will stand to describe more refined event models respecting the subsumption relation of the ontology of models.

Name	Description
Exist	New component appears or exist
Receive	Sense or receive
Own	Membership relations between entities or objects
Move	All simple moves, exchanges or transfers (even virtual like bank transfer)
Produce	Transformation or build process
Behave	All complex moves, adaptation &interaction schemes
Experience	Use or experiment, evaluate & measure

Table 3.3 – Root predicates

In Table 3.3, we give a description of the root-predicates. *Exist*, *Own* and *Receive* are root-predicates corresponding to action level (figure 3.4). For example, "*Exist:EntityPosition"* is an event model indicating a change in the position of an entity. Move and Produce are root-predicates corresponding to scenario level. "*Move:RobotWalk"* is another event model, it is a composition of several

memorized past facts under "*Exist:EntityPosition*" event model. *Behave* and *Experience* are root-predicates at modelling complex or dynamic behaviours level. Under these root-predicates are all specialised predicates; they are formed from the root-predicate and at different levels of the tree. Sub-predicates are specialised from their parent predicates. So parent predicates subsume children classes, and children predicates specialise parent predicates. The subsumption relation in the structure of our ontology realizes a part of the meaning function. *Behave* corresponds to a specific behaviour and can be applied to any entities like agent or process. *Behave* can also be used for architectural changes by replacing or disabling components of the architectural level. Architectural level also corresponds to the terms in the Meta ontology and permits to refine the ontologies. Any other root predicate may be employed to any kind of events while the integrity of all behavioural events in the ontology of models is respected. We have only presented in the figure the rootpredicates and predicates that have been used in our agents.

Our EKRL matches well behavioural levels since the ontology of models is partitioned in sub trees of root predicates for a fast querying and reasoning system (Figure 3.13). EKRL Formal grammar is used to generate and describe components and events. EKRL Formal language is used to query facts (instances) and recognize scenario using meta operators and concepts in a role of an event model. Roles are like properties in OWL. Roles in our EKRL describe an event happening on these entities or objects in all possible contexts and modal logics: temporal, spatial, acoustic, visual, danger, medical, emergency and so on. Examples of roles are SUBJECT (who or what is concerned), OBJECT (objective, goal), CONTEXT, CONTENT (values or string), LOCATION (space), DATE (date and time), and so on.

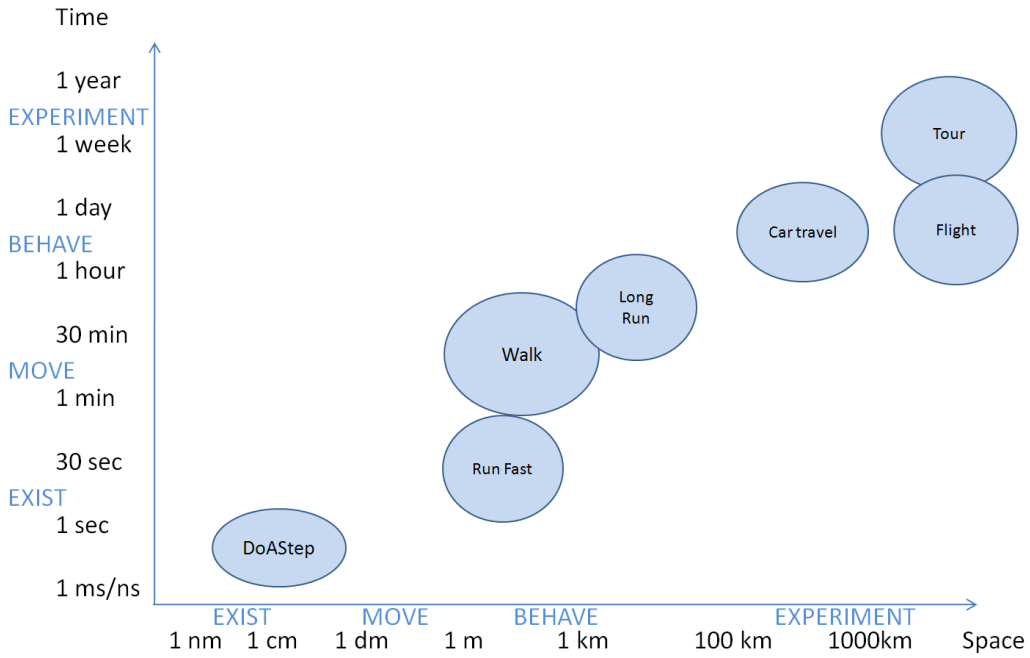


Figure 3.13 Space-Time Predicates

3.4.4 Modelling Connectors

Interaction Architecture can be defined with agent components and event messages, we define now the layers of exchanged and stored information (Table 3.5). Descriptions of these levels of information are:

1. Networking packets and SOAP web services protocol for interoperability. This layer implies a web or TCP-IP network and is useful for ambient and pervasive architecture but it is more related to physical transport of information. These messages are identical to electronic mails with FROM, TO, SUBJECT and CONTENT fields.
2. EKRL Events are used in communications between components. They are textual messages in the CONTENT field of the first layer. The content in a message is built under the form of the EKRL grammar.
3. KRL concepts and models in our two ontologies permits event storage like facts (past events) and models of events, models of query and models of scenarios.
4. Meta concepts are stored in a Meta ontology used to build Concepts and Models ontologies.

Level	Information Layer
1	Networking
2	Events Communication
3	Events Storage
4	Meta Information

Table 3.4 – Information layers. Lower abstraction level is 1, higher abstraction level is 4.

All agents must be conceived to integrate the 4 layers of Table 3.4.

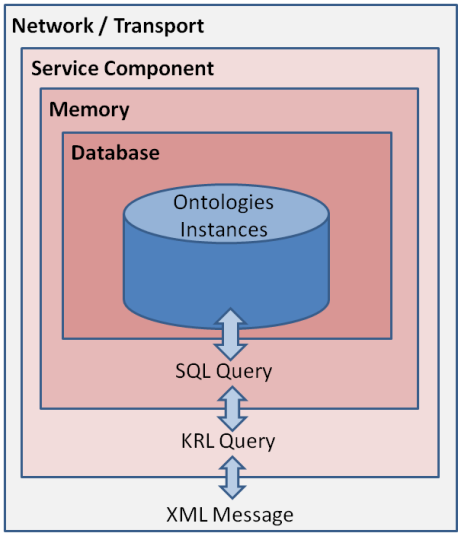


Figure 3.14 – Communication Protocols

Figure 3.14 shows our design of the memory access (store and query all information) of our webservice components. It is composed of 4 layers. The Network/Transport layer manages all XML exchanges that are performed at the Network level using SOAP or any standard webservice protocols. KRL Query in the description language is extracted and given to the Service component layer that will interpret the demand, realize operations. Memory Layer is the KRL

inference system which manages information stored in the ontologies. Ontologies are stored in the database. Memory creates and sends SQL queries to the database. Database layer is just a functional component integrated to store and query information in a fastest way. Memory role is necessary to reason and query the ontology at the different abstraction levels seen at Table 3.4 and to use the predicates and bring the meaning for the Service component to be able to act physically, logically or manage the network interaction.

3.4.5 Scales

Scales are special relationships used to implicitly link linguistics variables (set of concepts) to one or several ranges of numeric values depending on the context.

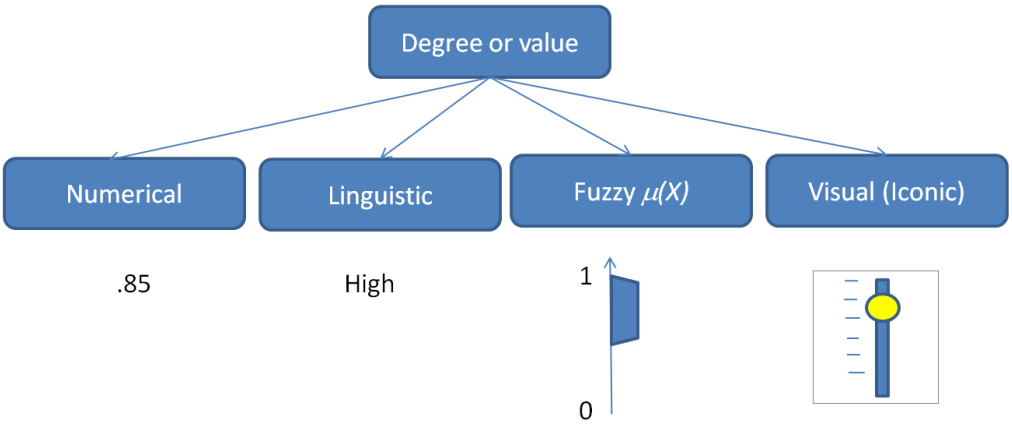


Figure 3.15 – Types of scale

They are very interesting to convert numerical values to linguistic variable and vice versa. As presented Figure 3.15, any numbers, linguistic symbols, distributions of probability or media with different attributes can have the same meaning in a specific context. As we defined a unique reference to any concept in the ontology of concepts, it is easy to attach any supplementary information to a concept. The notion of scale deals with predefined ordered set of concepts attached to values to convert a NL sentence in values and a value in a word. When used in conjunction with our inference engine, it appears very close to the paradigm of Computing With Words (CWW) recently introduced by Zadeh (Mendel et al., 2010).

Type	Description	Sample
Numerical to numerical	Unit or contextual conversion	°C → °K
Numerical to symbolic	Conceptual meaning	0.5 → middle
Symbolic to numerical	Numerical meaning	close → 1 cm
Symbolic to symbolic	Abstract level or contextual meaning	Move → Fly

Table 3.5 – Types of scale

Scales must be ordered, have a type (Table 3.5) and have a unit. And scales may possibly be attached to a unit conversion function available in the *Library* box of our agent.

$$\mu_c = scale_x(v) \qquad \textit{(Formula 5)}$$

for a digital to symbolic conversion where v is a numerical value, c is a linguistic symbol, x is a situational context. We obtain a probability distribution for c.

$$v = scale_x(c) \qquad \textit{(Formula 6)}$$

for a symbolic to digital conversion where v is a numerical value, c is a conceptual symbol, x is a situational context. The value v (attached to its unit) is directly extracted from the scale for the concept c in the context x.

For example, let us take the scale of distance: {"CLOSE", "NORMAL", "FAR"}. If our context is about a *boxing combat*, close is equivalent to 10 cm, normal to 1 m and far to 2 m. If the context is about a *trip in France*, our scale is equivalent { 200 km, 400 km, 800 km } or { 2 hours, 4 hours, 8 hours }. In a robotics context where *the robot manipulates an object*, our scale will be { 1 mm, 10 cm, 1 m }.

3.5 Modelling Architecture

3.5.1 Interaction Architecture

In control theory, the system is often decomposed in a controller, a plant to be controlled and a closed loop given a return from environment. Most automatic systems are designed following this scheme. Situation awareness model is built with multiple inputs and multiple outputs (Figure 3.16). In the interaction case, we consider all parts of the systems integrated in the environment. It is same for Multi agent systems (MAS), inputs received by sensors and outputs realized by actuators, are in interaction with the environment. But interaction can be extended to all components of the systems and the environment (Figure 3.17).

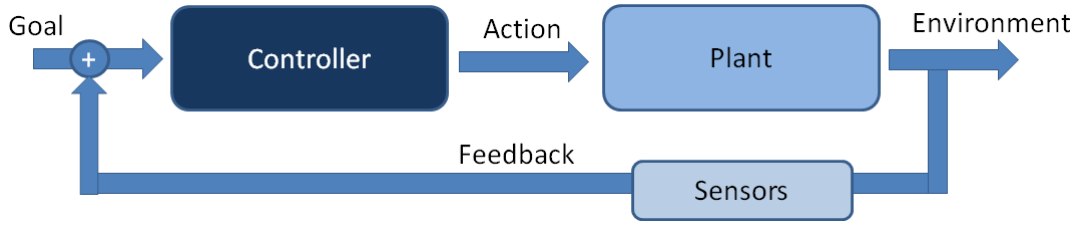


Figure 3.16 – Control Theory

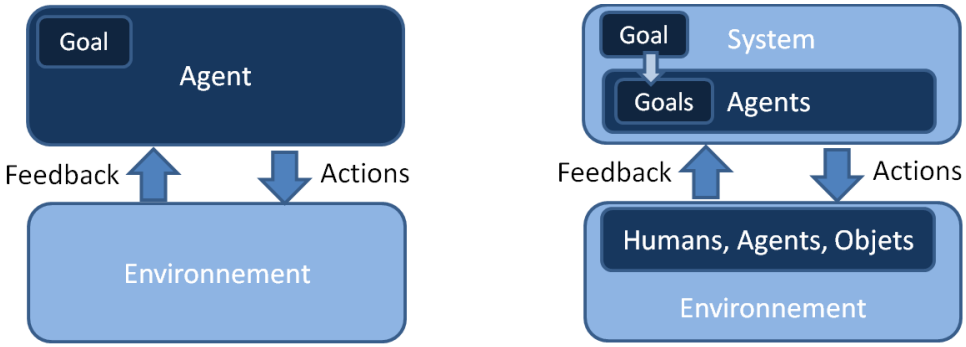


Figure 3.17 – Agents Interaction

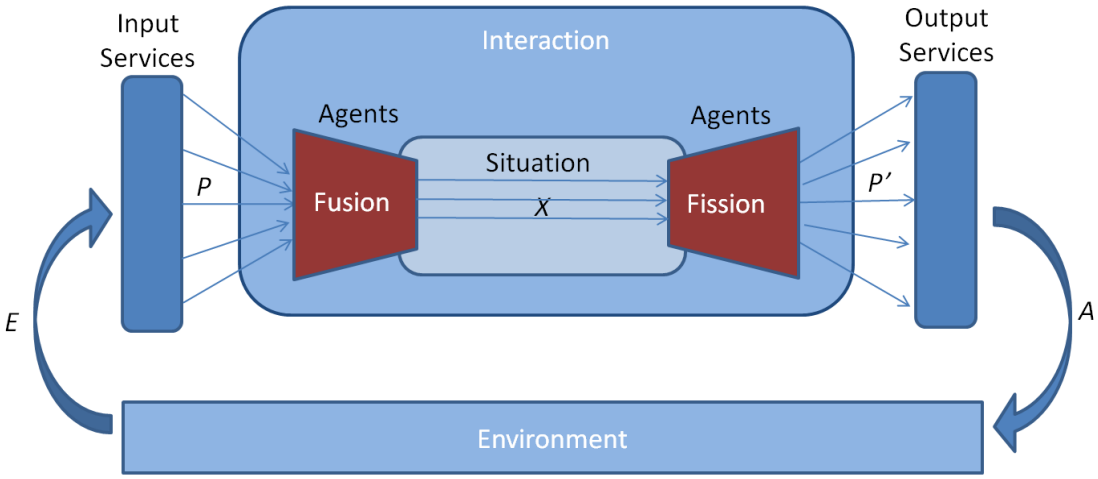


Figure 3.18 – Interaction Architecture

These agents are parts of a robot and interact between them and between environment items using *events* messages. Then we understand that complexity increases with the quantity of exchanges. Our architecture to be compliant with specifications must be modular and extendable. Architecture will provide all possible internal and external concurrent components. Figure 3.18 presents the composition of input and output services in the network and the agents managing the interaction. This architecture is modelled by the composition of the two Fusion and Fission main processes previously defined and required by the interaction. Fusion plays 2 main roles: First is selection of modalities. Second is meaning extraction (Formula 2) at different level of abstraction of information (numeric, symbolic, semantic, and cognitive). Rule models used to achieve this goal are predicates in memory. Fission plays also 2 mains roles: First is decision extraction. Second is output resources management. Fusion and Fission are realized by multiple semantic components called *Fusion Agents* and *Fission Agents*. Services are represented outside the environment in Figure 3.18 but in fact all the architecture (agents and services) is also parts of the environment. We also note that only one way is possible for the information to go through the architecture in the aim of efficiency and simplicity. It also means that the architecture is reactive but not only and agents work as state machines where their program is designed to anticipate and manage incoming events. Some effects on agents will be internal and personal to agents but other effects will be sent in the environment and observed by all other parts as it is the case in human environment for human.

To formally describe Figure 3.18, we denote:

$X=FusionAgents(P)$ and $P'=FissionAgents(X)$ *(Formula 7)*

$<=> P'=FissionAgents(FusionAgents(P))$ *(Formula 8)*

$And\ P=input\_services(E),\ A=output\_services(P')$ *(Formula 9)*

Where E is a vector of data coming from the environment, P are events encoded at a higher level of abstraction of the features extracted from data, X is the state or the situation including the situational and user contexts, P' are events sent to output services, A is a set of actions controlled by output services following the orders sent by fission agents. If the services are semantic or have a wrapper to

encode and decode events using an environment knowledge representation language (EKRL), we may also write:

$$P=EKRL(E) \qquad \textit{(Formula 10)}$$

where EKRL event are generated by input services

$$A= EKRL^{-1}(P') \qquad \textit{(Formula 11)}$$

where EKRL orders are sent to output services which will control actuators (hardware) or call network functions (software) to execute these orders.

3.5.2 Architecture Components

Formally, Figure 3.19 presents the composition of same agent component, and then we can denote:

$$X=FusionAgents(P) \text{ and } P'=FissionAgents(X)$$

$$\Leftrightarrow X= \sum_l \sum_i Agent_{l,i}(P) \text{ and } P'= \sum_l \sum_i Agent_{l,i}(X) \qquad \textit{(Formula 12)}$$

where P is the percept events vector, X is the complete situation vector (past and now), l is the layer and i the agent number, P is the action events vector.

The complete situation X is stored in the last layer of fusion agents and sent to the first layer of fission agents that can also store or forget these events after processing it.

If these agents are knowledge based, the data processing will be realized by an inference engine and then we can now write:

$$P'=FissionAgents(FusionAgents(P))$$

$$\Leftrightarrow P'= \sum_l \sum_i Agent_{l,i}(P)$$

$$\Leftrightarrow P'= \sum_l \sum_i Inference_{l,i}(P) \qquad \textit{(Formula 13)}$$

Inference function is realized by the inference engine of the agent i in the layer l. The architecture is a network or a grid of *inference* components.

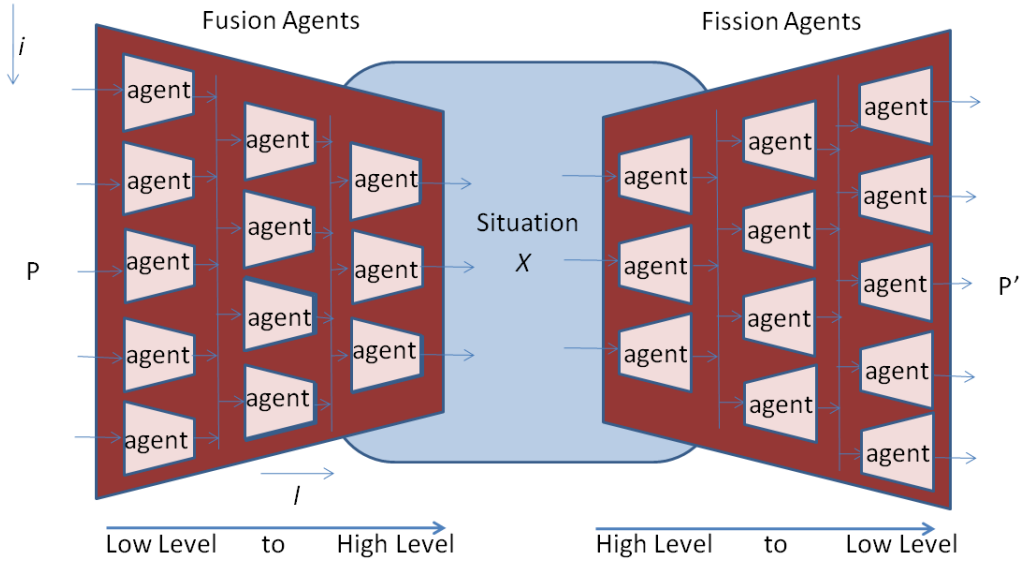


Figure 3.19 – Fusion and Fission

Conceptually, a policy is a mean offered to agents to define and modify the organization and the behaviour of a system. It generally contains rules, deriving

from strategy description and specifying to the system the actions to undertake to answer to a given situation. Policies are rules governing the choices in behaviour of a system. Or formally, a policy can be defined from two perspectives:

- A strategy, goals and objectives to guide and determine present and future actions to be performed within the environment or the system.
- A set of rules to administer, monitor and control access to system resources.

These definitions highlight the different levels of abstraction introduced by the specification of policies, according to the user. The job level abstractions simplify the administrators in defining their needs as a global goal as the rules go into details of the implementation of the strategy to use to achieve these goals.

Meanings or actions are extracted by rules of agents (in the knowledge base) by the inference engine from situation, context and objectives inputs.

This makes the system more flexible and able to reconfigure during their execution if necessary, to comply with the predefined rules. The rules that comprise policies are generally described by variants of the paradigm Event Condition Action (ECA). Rules used by the inference engine can be written in this form:

preconditions => postconditions *(Formula 14)*

Antecedent (preconditions) and consequent (postconditions) contain transition models. Informally, the rule asserts: "If the transitions specified by the antecedents are valid, then the transition specified by the consequent is valid."

For such rules, the preconditions are evaluated when a new event is inserted in the knowledge base and the postconditions will be executed if the preconditions are matching a rule model.

The transition relation, =>, for our KRL semantic is specified by a set of *rule models*. A rule model is a predicate p where some roles R_i contain relationships on other event models. In case of fusion process, a rule model produces one new fact from several input event, this fact will have the name of the consequent model. In case of fission process, it is exactly the same but there will be the same number of rule models that there are envisaged output facts or orders. We will present later an example of fusion and fission composite events for the « PutThatHere » scenario. The name of the rule is the name of the predicate used as a rule model to evaluate the preconditions. Inference engine will compare facts with this model (an in particular, compare all roles that are not empty) to check if the fact match the model. If it is the case, the output event is filled with the compound content of roles of concerned facts matching models, becomes an instance of the model and is sent to other agents (postcondition aggregation).

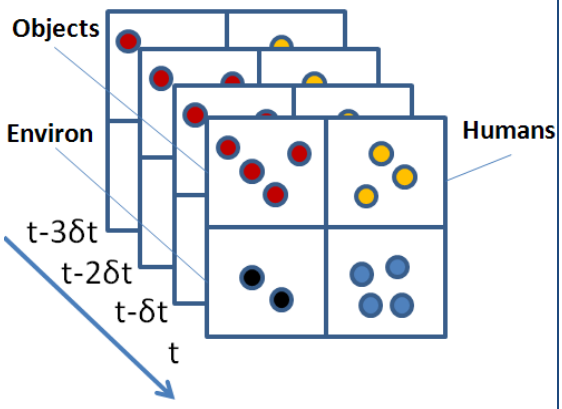
This architecture allows: *Multimodal Fusion* of events coming from the environment choosing input modalities and events to compose. *Query the system* as any incoming events will produce a response of the agents at the condition that they understand and don't ignore these events. It means that agents must obviously be programmed to manage these events. *Maintaining the situation* of any systems of the environment if sensors are able to observe all these systems. *Multimodal Fission* of events to select output modalities and

realize *Decision, Planning* or *Execution* by sending order to hardware controller services or software services. Learning is also possible by adding an external system to train the agents of this architecture or by adding internal agents to manage and adapt the events and the program of the other agents.

We may add that all agents can be autonomous and execution can be made by concurrent or parallel tasks. All components (agents, service and the environment) are connected by the mean of a common network. Implementation can be fully compliant with Microsoft Robotics Studio, URBI, JADE platforms.

3.5.3 Interaction Features

Little different from the OODA model presented in (Steinberg, 2008), we will now define our OODFA model. It is more precise and complete because the added *Fusion* process stores all situations *in KRL* in the memory of our agent component. This operation will permit the decision process to directly find decision using the fully stored meaning.

Operation	Information	Output stored in memory
Observe All event messages coming from the environment		All event messages come from network services (sensors, actuators, others)
Orient **focus composition**		Categorized events in an ontology
Fusion **time composition** Each process acts concurrently on several data sets (parallel schemas or scenarios) so δ parameter is different based on sample rate of the different stored scenarios.		Change and relationships are stored in memory to extract meaning. Blue events are stored but can be ignored or not depending of relationships with other information.
Decide Use meaning related to goals to choose actions to do	- Concerned Entities - Recognized Scenarios - Constraints & Possible Operations	Fission will realize interpretation of incoming events (situational

in future taking account of time of network transfer, time of planning and time of execution	- Predict using scenario	meaning), evaluate options, split the task to execute and plan actions to achieve
Act all planned action messages are sent to actuators or services or sensor controllers.	- Internal Actions (focus, process, memorize) - External Actions (send orders to actuators, composition of services)	Actions/Orders messages

Figure 3.20 – OOFDA Model

Figure 3.20 shows how components act as filters to bring features.

OOFDA is an interaction model, Letters stand for these operations:

- *Observe* receives cleaned acquired data sent from intelligent sensors in a XML message format.
- *Orient* interpret and classifies events to be stored belongs to several categories.
- *Fusion* extracts all meaningful information related to possible scenarios in time. Scenarios of environments and Meta-Scenarios of the Architecture execution. It is useful to update the world state and obtain a complete perception of the context in which all tasks will be deliberated
- *Decide* uses meaning related to mission goals to continually reason on opportunities and situations, and choose actions to do in future taking account of temporal aspects such as time of network transfer, time of planning and time of execution.
- *Act* manages the composition of service, planning to act in time and executing. All planned action messages sent to actuators or services or sensor controllers.

There are differences between our architecture and OOFDA model. Planning and decision events are managed by the *Decide* operation. Execution events are controlled in the *Act* operation. If an execution problem occurs, the event will do the loop and *Decide* operation will adapt the planning. Memory of agents is used to store all events and queries are used to retrieve all information and needed meaning, they are used by all operations. We will see how they can be integrated in one or several basic component structure in the next sections.

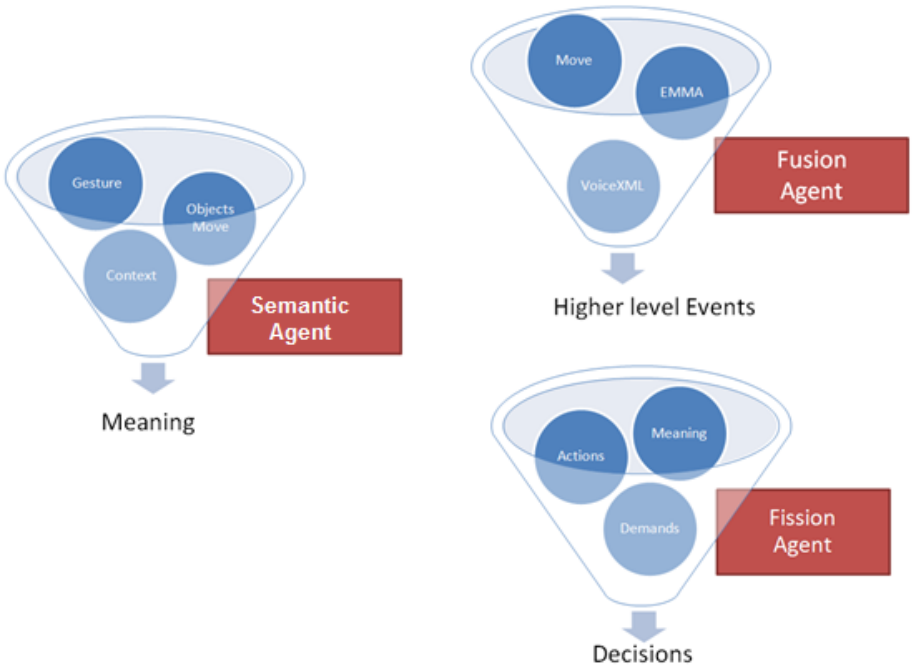


Figure 3.21 – Final Features

In Figure 3.21, we present the *semantic agent* able to extract any meanings from events (situational or contextual for fusion or operational for fission). From this semantic agent, we obtain the *fusion agent* and *fission agent*. They will be designed to make *Orient*, *Fusion* and *Decide* operations of OOFDA model to store, retrieve and filter information to be exploited. In our case, *Observe* and *Act* operations will be managed by services and agents, and *Orient*, *Fusion* and *Decide* only by agents. We decompose the interaction problem in two main processes: - *Fusion process* which is the first half part of interaction integrating *Observe*, *Orient* and *Fusion* operations of the OOFDA model; and – *Fission process* which is the second half part of interaction integrating *Decide* and *Act* operations.

3.6 Modelling Components of the Architecture

3.6.1 Component Classes

In our interaction architecture, a component can be an agent, an input service or an output service. Figure 3.22 shows the UML classes of our components. All our components are webservices. Webservice is a generic component with additional WS public methods. Then, intrinsically, each component can be discovered and can provide a set of personal functions to other components. Connector is the same for all components and is a standard network, often including TCP-IP with SOAP or REST protocols. Communication is done between agents, services, sensors and human interfaces.

Service and *Agent,* our two main components used in our interaction architecture inherit of the webservice component. Service can manage software and hardware and agent can only store and reason about information received. They can be several with being assigned with a specific task and a limited set of modalities. It permits to solve problems with several concurrent goals. Of course, the same type of components may have different names for the

composition to be very open for the designer of any interaction architecture. These services will be integrated in a network of computers or embedded systems as software services but they can also be integrated in hardware parts like into a mobile robot. Our model is platform independent.

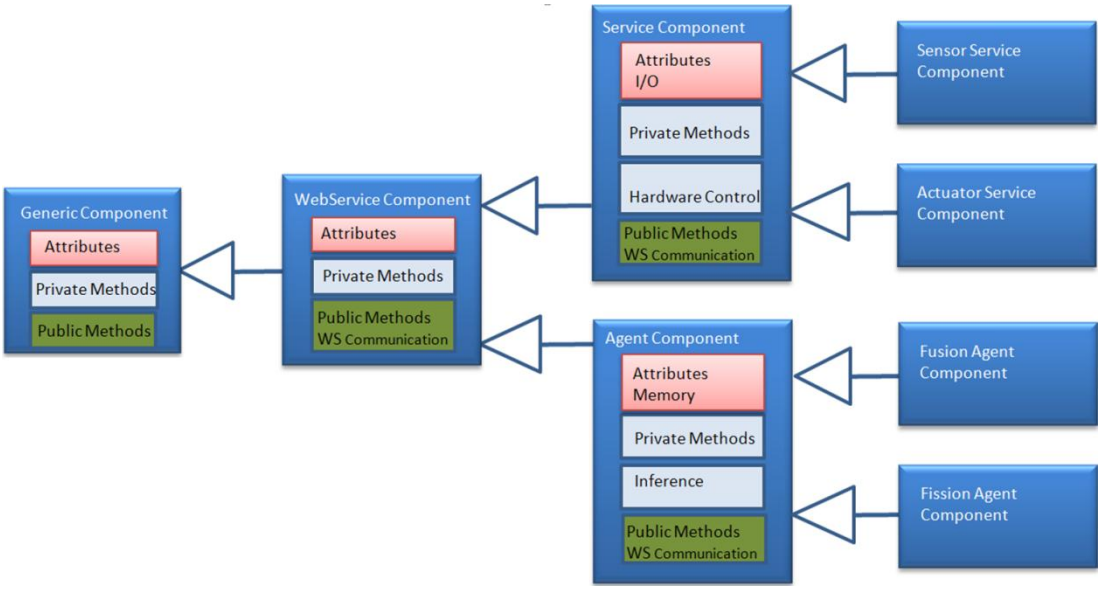


Figure 3.22 – UML Classes

3.6.2 Generic and Webservice Components

The generic component is functioning on the hardware component. In Figure 3.23, the content of generic and webservice components is shown. Memory can contain information of the data types: raw, database, ontology or any required structures. Code is the program or the inference engine to operate. Library is a set of functions specific to the job of the component; they can be software (agent and software service) or hardware (driver service).

Private methods are:

- *Code* executed by the processor reading the execution schemes in memory following received messages. Memory management is realized by the Code box, reading or writing data in the memory. A bootstrap method or component constructor initializes the memory and starts the component execution, generally run at hardware start or following a reset.
- *Library (software or hardware management)* composed of services or devices drivers and any control schemes.

Public methods are:

- *Receive and Queuing Messages* which performs network reception of messages that are buffered and then read and removed by the *Code* box.
- *Queuing and Send Messages*, these messages should be in the protocol format.

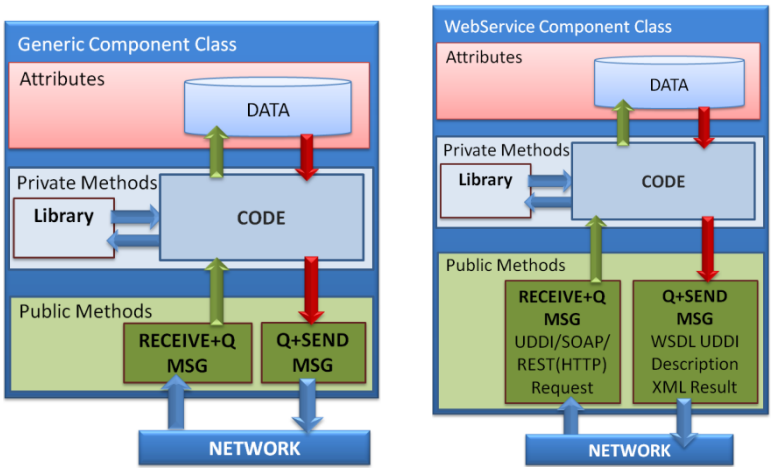


Figure 3.23 – Generic and Webservice Components

For the webservice component, methods are now augmented:

- Code of the component *constructor* performs the webservice registration in the network and starts the web service listening function (WsInit).
- *Receive and Queuing Messages* which gets XML events messages or UDDI request for discovery, integration or execution (SOAP, REST, OWL-S) from the network. They are buffered and sent to the *Code* box.
- *Queuing and Send Messages* coming from by the *Code* box. It can be also used for webservice composition sending WS specific events: WSDL description file or UDDI messages to others.

A webservice is located at a fixed address; it has a list of functions that are running on demand by clients. It may automatically trigger regular events. These components are pure standard Web services using SOAP for inter-operability. They interact with the environment via the wireless or wired network, also part of the environment. OWL-S repositories and UDDI directory servers for discovery are also input services to our architecture. We will present later their importance in the composition of services and agencies. Using external standard Web services requires an XML to EKRL wrapper (input services), since they do not use EKRL semantics. The result is that our architecture is operating-system independent and built on standards to remain compliant. To achieve composition, agent memories will keep the service profile, service grounding and service model in memory. Agent and Services have an IP address and one or more TCP ports. Mobile agents must change IP address by moving from one network connection to another. Additional security schemes may be added to manage the privacy of information, services, or the network when the mobile agent acquires a new address or a Kerberos ticket is granted to access a service.

3.6.3 Service Component

The *Input/Output service* is a webservice component structure that will permit to design, compose and build any low levels operations of our architecture. A service can be a sensor input controller, an actuator output controller, a functional web service (software operation, storage, searching, sorting, and so on) or a non-functional web service (QoS, Performance, and so on). Moreover, it will permit to reuse existing components and adapt a configuration of a dynamic architecture in real time.

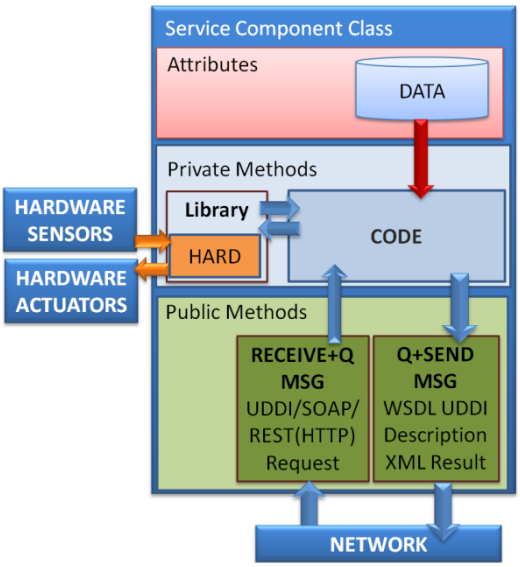


Figure 3.24 – Input/Output Service Component class

In Figure 3.24, Memory Header comprises the service name, service id, service list of available public methods, IP address, and IP port. This header will be useful for webservices discovery, composition and to store on a UDDI server as a WSDL file.

There are two differences with the webservice component in the private methods:

- Library contains a set of software functions and a set of hardware drivers to control attached hardware parts.
- Code can fill a very short set of EKRL messages (flat event models and concerned concepts without ontology) to send extracted features from sensors data to agents and code can read EKRL messages from agents to control hardware or run software functions. Code will also realize the operations and to convert information from the service to others services using any standards XML protocols depending on the modality or method of the target webservice which delivers the service on Internet.

We assume incoming EKRL message contain models recognized by agents. If a known predicate is not included in the EKRL message, then the fact will be ignored. Same for service which receives a message of which it has no model. Our web services are special web services, they can communicate using semantics knowledge in messages (the events) but they are not semantic. We will describe semantic knowledge representation language and show later how to use these web services in our architecture to develop powerful multimodal interaction applications in an ambient or pervasive architectures with the use of semantic agents. For example, different vocal sensors are located everywhere in a house and when one of them receive a vocal message from a user, it will interpret the message, translate the sound in the message in natural language or in a VoiceXML protocol comprising message, location, date, time and sent it to the fusion agent component. Maybe this vocal sensor could control the volume of sound or any sharp by using the *Hardware Output* block.

3.6.4 Agent Component

Our agent can be seen as the Match-Update-Prediction cyclic process of Crowley and Demazeau (1993). Starting from the Principles of fusion and fission, we define:

Principle 1) Primitives in the world model should be expressed as a set of roles. A model primitive expresses an association of properties which describe the state of some part of the world. This association is typically based on spatial position. For numerical quantities, each role can be listed as an estimate accompanied by a precision. For symbolic entities, the role slot can be filled with a list of possible concepts, from a finite vocabulary. This association of properties is known as the "state vector" in estimation theory. Schema provides just such a representation. The roles may be symbolic labels or numerical measures.

Principle 2) Observation, Model and Action should be expressed in a common coordinate system. This principle is a set of relationships between concepts and event models. This association may be on the basis of spatial or temporal coordinates, or it may be on the basis of a relationship between roles. Spatial location role is a powerful logic for associating information. Temporal start/end roles are also a powerful logic for associating information. Coordination depends of the context or the current situation too. A context is a collection of symbols and relations which are used to describe a situation.

Principle 3) Observation, Model and Action should be expressed in a common vocabulary. Common vocabulary is a set of concepts and event models. All parts of the architecture must use this common language to understand and act in the environment. This permits description to proceed by a process of prediction and verification.

Principle 4) Roles should include an explicit representation of uncertainty. As with numerical roles or concepts, symbolic roles or concepts have two kinds of uncertainties: precision and confidence. The classic AI method for representing precision is to provide a list of possible values. Such lists are used both for symbolic roles and for relations. Constraints are applied by intersecting lists of possible values. We defined a specific type to attach numerical values (fuzzy sets) to symbolic concepts (linguistics variables) called scales depending on the context.

Principle 5) Primitives should be accompanied by a confidence factor. This characterizes the quality of incoming information.

The *Agent* is a webservice component structure that will permit to perform any reasoning or cognitive function by processing high levels operations of our architecture. An agent is a web service that acts and communicates autonomously (Figure 3.25).

The memory is a *knowledge base* of classes, concepts, events and instances. Some events in memory correspond to the agent program. This program is a list of query models read in memory and applied to models and facts in memory. To run this program, the blue box code has been split in two parts:

- the code to start the agent and the *inference engine* and, to initialize the knowledge base and the network messaging
- the inference engine to store new recognized facts coming from the network and placed in the incoming queue, to match query models with facts

producing new events at a higher level of abstraction (case of fusion agents) or lower level of abstraction (case of fission agents).

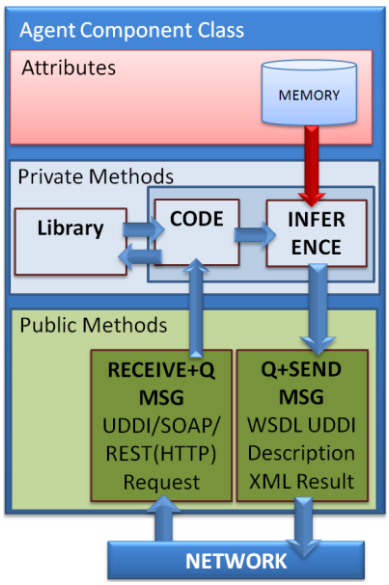


Figure 3.25 – Agent Component class

As for the generic component class, the agent component can use its own library of functions. It allows inference engine to execute basic functions (mathematical operations, temporal differences, distances & units conversion and so on) without waiting the answer from the network. It is important for a gain of performance. In the aim of interaction, agent will be dedicated to Fusion or Fission processes. Because of this ability to extract meaning, we call him *Semantic Agent*. A semantic agent can directly use natural language or EKRL to communicate. If it manages quality attributes like Quality of service, Performance, it will be called a *non-functional agent*. If it acts socially regrouping agents to do a task, it will be called *manager agent*. A *meta agent* can be: a *meta* functional agent capable to adapt the architecture, or a *meta* semantic agent capable to update the knowledge base (memory) of other agents.

3.7 Designing Semantic Agents

3.7.1 Definition

Semantic agents inherit web services components of our architecture model. They are different and a little more complex because they are cognitive. They possess cognitive abilities and programs to achieve their tasks and goals. Semantic agents contain their own embedded inference system able to process a matching operation. They can store EKRL events in a knowledge base (memory), answer a query and produce new events by extracting meaning of the situation. Scenarios or execution schemes are stored in their knowledge base. The only difference between two agents is the knowledge in memory, all agents have the same inference engine in the Code box. In particular, rule models stored in the agent's memory define the agent program, i.e., the role of an agent in the organization. They receive, filter, and attach facts to their own event models. Along the same lines, they will send event orders or new composed events according to their internal query models for the function they are designed for.

The developer programs agents by reusing default concepts and models and by adding specific query models in the agent's memory via an editor.

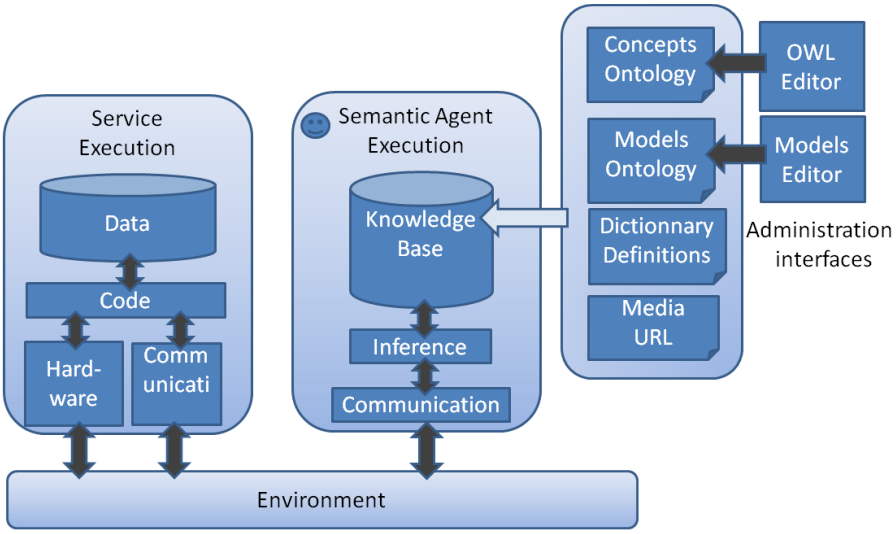


Figure 3.26 – Service and Agent Design

Figure 3.26 shows the interaction between an agent and a service. An agent comprises its knowledge base (also called memory of the agent), inference engine, and communication module. A service has only code and standard memory (properties and methods in the generic programming object model), a communication module, and a hardware controller module. The hardware controller enables the service to receive information from a sensor or drive an actuator. The communication module contains the network card and its semantic functionalities to write and send or to receive events in EKRL.

3.7.2 Memory of Agent

Agent's memory is the most important piece of software for our agent to be cognitive: ability to store and retrieve events, to understand the situational meaning, and to create new events to be sent to other semantic agents and services. Meta concepts, concepts, event models, query models and instances are stored in ontologies. Instances are facts, scenarios that have happened, and context knowledge.

Agent's memory is a knowledge base storing all events coming from the network and used for cognitive operations by recalling any past facts or for reasoning and acting according to stored event models. This memory is made of a domain ontology called *Ontology of concepts* and a second ontology called *Ontology of models.* Event models are fully linked with concepts in frames to give the agent its cognitive abilities.

Figure 3.27 is detailed. It represents the knowledge base and inference modules of the agent. *StoringEngine* and *QueryEngine* are functional parts of the ontological inference engine. The cylinders are components of the knowledge base (Figure 3.28) and are the ontologies of (meta, concepts, models) classes and instance bases respectively. All inserted facts coming from the network are fully and rationally linked to concepts and models in this the memory of the agent.

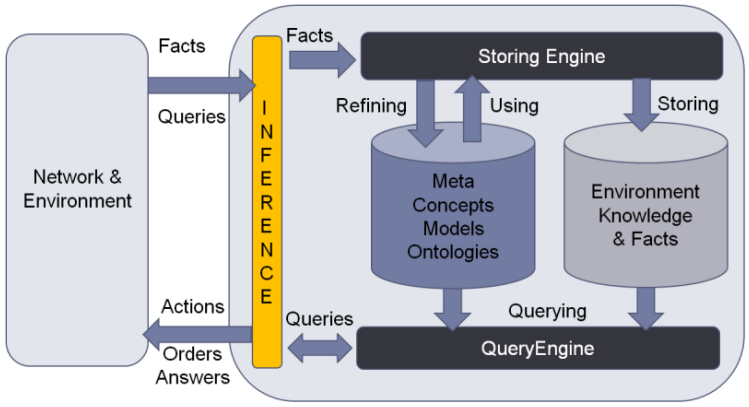


Figure 3.27 – Storage and Querying the memory of the agent

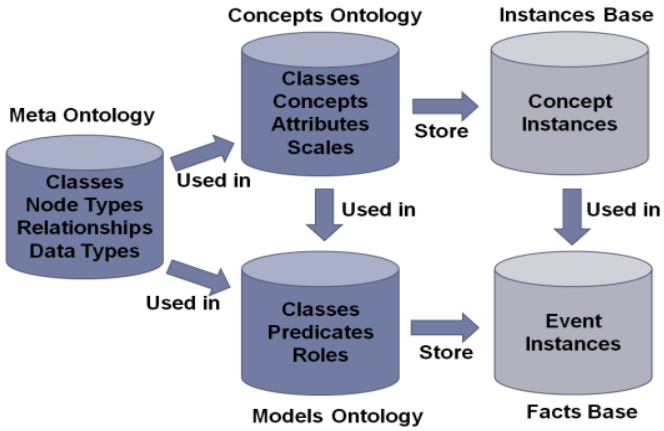


Figure 3.28 – Knowledge base content

Building an ontology is deciding on a set of primitives of representation of knowledge (entities, relations, functions, axioms and authorities). With these primitives, an operational semantics is given by clarifying the relations which bind these primitives between them compared to certain rules (rules of constructions and deductions) and the conditions of activating these rules. This requires the use of a formal language of representation of the ontology which can use the same formalism or two different formalisms of the system containing knowledge in which the primitives will be used like OWL and EKRL.

Meta Ontology is a domain ontology which has all types of nodes and relationships, roles and modifiers to build the concepts and models ontologies.

The Meta ontology (Figure 3.29) is fully compliant with OWL2 XML language terms and relations: subsumption, memberships, equivalence, synonymy, discrepancy, sentence modifiers (multimodal logic of the natural language). Ontology of concepts (Figure 3.30) is a common domain ontology containing hierarchically sorted concepts with instances of concepts fully compatible with the OWL2 standard. Ontology of models is a knowledge base (Figure 3.8) containing hierarchically sorted predicates with model instances called facts. It embeds event templates (formal T-BOX of models) under the form of predicates and instances of events (formal A-BOX). Gravier et al. (2011) studied A-Box revisions in Semantic Context-Aware Systems. Models are filled with concepts and instances of concepts to build the facts. (Figure 3.31)

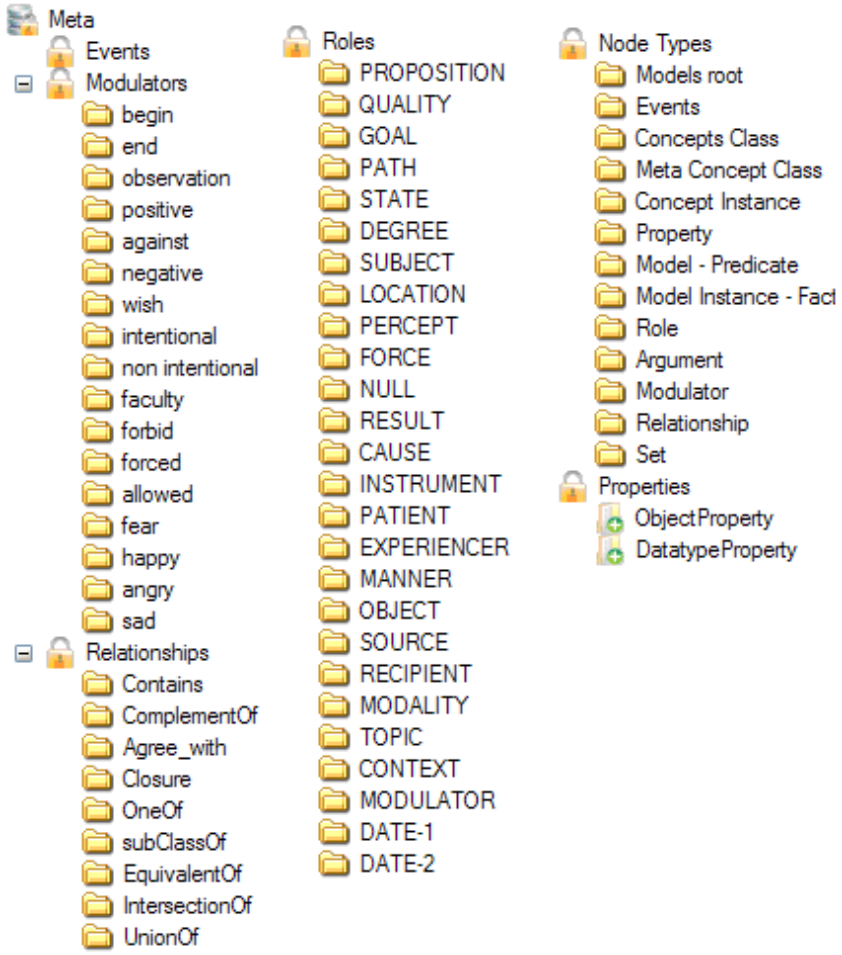


Figure 3.29 – Meta Ontology

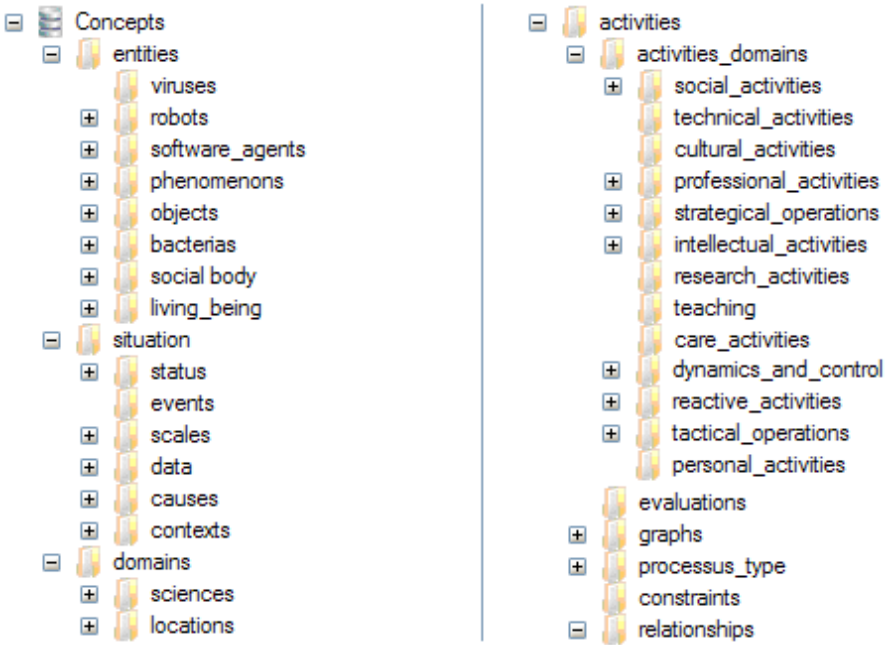


Figure 3.30 – Ontology of concepts

Figure 3.31 – Ontology of models

3.7.3 Inference Engine Algorithms

We propose the use of EKRL for the agent's communication, storage, and events ontology format. In all other platforms, agent behaviours are programmed in Java, C, or C++. Developers of our agents just need to insert EKRL events in order to program them. Generally speaking, ontology is a shared blackboard used for transmitting data between agents in ACL messages. Besides the programming work, converters/wrappers need to be developed, and standards and protocols controlled, resulting in an expert software engineering task that is difficult to implement and maintain. For defining and controlling applications, current standards rely on higher level protocols, requiring a lot of programming for executing tasks and interpreting the situation, rather than a more natural language. This was a motivating factor behind our contribution.

Data transmitted over the network are used by the other agents' code but are not stored in a memory using an ontology of models and not interpreted. We might say that these agents are rational but not cognitive. Event calculus increases these abilities (Kowalski and Sergot 1986, Reiter 2001, Shvaiko 2008).

One more point is that our ontologies are in no way restricted to communication only; they store facts and compose new higher level facts in the case of fusion and lower level facts in the case of fission. We include some event models in the

memory that correspond to actions or scenarios in order to make this memory recognize facts of a similar nature and send higher level meaning. It is more a modelling and agent programming memory than a self-adapting memory. We wanted a well organized symbolic memory able to fulfil system interaction requirements. Models are used so that:

- ✓ Agent can store the various events;
- ✓ Agent can read its program. This program is in the form of query models stored in memory, because the work of the fusion or fission agent is limited to producing events from those already present in its memory;
- ✓ Developer/Designer can question the memory to check instances of concepts, events and recorded facts;
- ✓ Developer/Designer can program the agent, adding or modifying concepts, event models, request models;
- ✓ Facts and actions can be stored directly under the corresponding event models. Thus EKRL is also used as a programming language besides as a language for communication and storage.

The Agent inference engine processes memory information and is used to:

1. Store events using event models already in memory
2. Query the memory (using query models) to find direct answers (direct matching) or to find indirect answers (matching requires the execution of operations) using operations on concepts as arguments of a role and operation on events (other predicates) as arguments of a role, or query rule models (program of the agent).
3. Execute scenarios (send/broadcast several events to one or more service agents).

```
Input: QueryModels
Do
  Facts←krl_listen()
  If Facts Then NewFact←StoreFact(Facts)
  [Result, [Queries]]← QueryEvents(QueryModels)
  If Result or NewFact Then
     For each Query of Queries
       [Result, [MatchEvents]]← QueryEvents(Query)
       If Result Then krl_send(Aggregation(MatchEvents))
     Next
   End If
Loop
```

Algorithm 1 – Main Code

Algorithm 1 is the main algorithm executed by the agent's code. The main loop stores new facts and checks for query models (equivalent to the agent's program). If new facts are added or the queries match query instances, the main algorithm will check for new matching facts and send them to other agents able to accept them (i.e., if they have the same models in memory). If an event model doesn't exist, the new fact will simply be ignored. The agent then won't manage this kind of event meaning and will concentrate on the tasks it is programmed for. The *[vectorname]* syntax denotes a vector named *vectorname.*

StoreFact Function

This function stores a fact in memory under its event model. If Result (Boolean) is true, the fact is stored under its model. Else the event is ignored. If no corresponding event model is found, it means the agent is not designed to process this type of event, reducing its memory and workloads. The function prototype is: *[Result]← StoreFact(Fact).*

```
[ParseError,RootPredicate, Predicate, [Roles], [Arguments]]←Split(Fact)
If ParseError Then Return False
EventPredicateID ←Matching(RootPredicate, Predicate)
If EventPredicateID>0 Then
   StoreRA(EventPredicateID, [Roles], [Arguments])
   Return True
Else
   Return False
End If
```

Algorithm 2 – StoreFact

QueryEvents Function

This function queries memory for event models and facts. If Result is true, events found are sent to all other agents. Else no event is sent. Prototype of the function is: *[Result, [MatchEvents]]← QueryEvents(QueryModel or QueryModelID).*

```
If (QueryModelID) Then QueryModel←Get(QueryModelID)
[ParseError, RootPredicate, Predicate, Roles, Arguments]←Split(QueryModel)
If ParseError Then Return [False, EmptySet]
[EventsPredicateID]←Matching(RootPredicate, Predicate, [Roles],[Arguments])
If count(EventsPredicateID)>0 Then
   [MatchEvents]←GetTextEvents([EventsPredicateID])
   Return [True, [MatchEvents]]
Else
   Return [False, EmptySet]
End If
```

Algorithm 3 – QueryEvents

Matching Function

This function performs matching operations between predicates and roles. *StoreRA()* and *ReadRA()* are SQL Insert and Select operations respectively in the *Role-Arguments* table filtered by ID arguments given to these two functions. Modifier is a role for modifying the sense of an event. Depending on modifiers, the matching result can vary. If the sense of the fact is negative, the event's sense is inverted so matching must take account of this too. The function prototype is *[EventsPredicateID]←Matching(RootPredicate, Predicate, [Roles], [Arguments]).*

```
RootPredicateID← GetNodeID(RootPredicate)
PredicateID←GetNodeID (Predicate)
[RolesID] ← GetNodeID ([Roles])
[ArgsID] ← GetNodeID ([Arguments])
[EventsID]←ReadRA(RootPredicateID, PredicateID,[RolesID])
For Each EventID in [EventsPredicatesID]
        [[EventRolesID],[EventArgsID]] ←ReadRA(EventID)
        For Each ArgID in [ArgsID]
                EventArgID←EventsArgsID[rank(ArgID)]
                If not MatchArguments(ArgID,EventArgID) Then Next EventID
        Next ArgID
        If ApplyModifier(EventRolesID) != ApplyModifier(RoleID) Then Next EventID
        [EventsPredicateID]← [EventsPredicateIDEventID]
Next EventID
Return [EventsPredicateID]
```

Algorithm 4 – Events Matching

MatchArguments Function

This function performs matching operations between two arguments of a role. ReadConcept(QueryArgID) is an SQL Select operation in the *nodes* table, where the node type is concept classes or instances and where these nodes are under the given node using the subsumption relationship of the *links* table. The SQL request gives all the subtree nodes sorted. The “date”, ”location”, “context”, ”content” and “value” role arguments will be compared using specific meta operators like “>”, “<”, “<=”, “>=”,”AND”,”OR” to check the given event models. This explains why the C*ompareEvents*() function returns a boolean. In fact, any event calculi are made using this function. The *MODIFIER* is a role of predicate to change the meaning of the sentence (fact) and thus the applied logic (temporal, spatial, modal with necessity and obligation modalities, and so on). This function compares all arguments and applies a modifier to the final boolean result. The function prototype is *[MatchingResult] ← MatchArguments(QueryArgID, EventArgID).*

```
If QueryArgID=EventArgID Then Return True
[ArgsID]=ReadConcept(QueryArgID)
For all ArgId in ArgsID
   If ArgId=EventArgID then Return True
   Else Return CompareEvents(ArgId, EventArgID)
Next
Return False
```

Algorithm 5 – Argument Matching

Aggregation Function

The main code can be called at each time new facts are stored or at different time (interval specific to the agent triggered by a timer set by the designer). The given arguments are the rule models stored in the agent’s memory. If rule models match past and incoming facts (the precondition) then new events are produced (the postcondition). These events are obtained from the class of rule models. The arguments in the roles of all corresponding facts are aggregated to fill the roles of the rule models. Each role corresponds to a modality (section 3.4).

The precondition is a specific role (of the rule model) containing a formula on roles and arguments of different event models.

Behave:PutThatHere

Object:*precondition*

User: *f1.User*

Source: *f3.content*

Recipient: *f6.content*

Date1: *inf(f_k.date1)*, for *k*=1 to 7

Date2: *sup(f_k.date2)*, for *k*=1 to 7

Location: *f1.location*

Figure 3.32 – "Behave:PutThatHere" Rule model

In our example of scenario "PutThatHere", there is a rule model "*Behave:PutThatHere*" (Figure 3.32) which is composed of 7 facts (Table 3.6). Usually, the event names are used but, we used f_k for readability. The precondition is in the *Object* role of the model (Figure 3.33). We could also add and *f3.source=OneOf(Objects) and f6.Source= OneOf(Objects)* to match objects of the concept ontology. If the precondition is true, i.e. the arguments of facts match the arguments of the *Object* role, then the inference engine will produce the "*Behave:PutThatHere*" fact which is the aggregation of the arguments of the facts *f1* to *f7* in the other roles of the model.

fact	Event names	Roles	Arguments
f1	ArmShowsPosition	User, Content, date1, date2	"James", "Point" .38 s, 1.95 s
f2	SpeechOrder	User, Content, date1, date2	"James", "Put" .42 s, 1.45 s
f3	ArmShowsObject	User, Content, location, date1, date2	"James", "Glass5" "room1","1.34 s, 2.05 s
f4	SpeechOrder	User, Content, date1, date2	"James", "That" 1.7 s, 2.3 s
f5	ArmShowsPosition	User,Content, date1, date2	"James", "Point" 3.1 s, 4.2 s
f6	ArmShowsObject	User, Content, date1, date2	"James", "Table2" "room2", 3.5 s, 4.3 s
f7	SpeechOrder	User, Content, date1, date2	"James", "Here" 3.9 s, 4.45 s

Table 3.6 – Facts matching "Behave:PutThatHere" rule model

Depending on used roles in the precondition and postcondition, the matching will use temporal logic (working on dates), spatial logic (working on locations) and in general in multi modal logics corresponding to any roles of the model. We will present the temporal, spatial and multimodal validation of logic associated to this rule model with the formal Petri nets in Chapter 5.

precondition ←(m1.date1≥f_k.date1 and f_k.date2≤m1.date2) and same(fk.Source) for k=1 to 7, and f1.content="Point" and f2.content="Put" and f4.content="That" and f5.content="Point" and f7.content="Here"

Figure 3.33 – "Behave:PutThatHere Rule model Precondition

3.7.4 Memory Editor

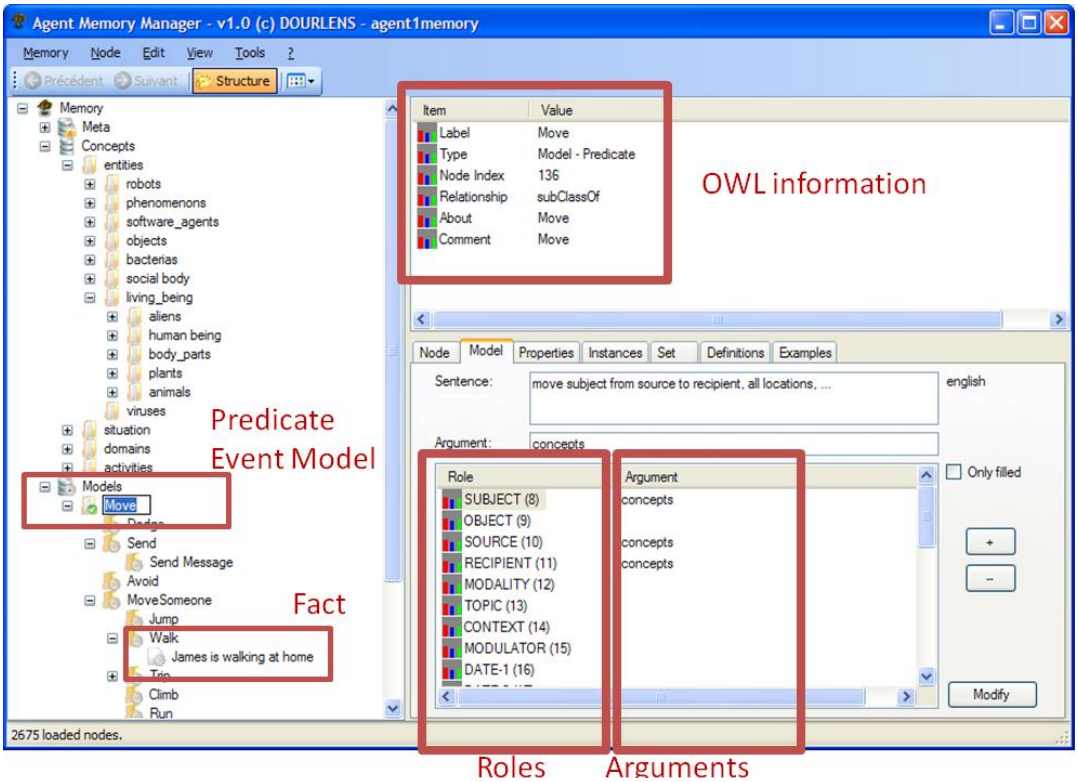
Figure 3.34 shows our editor of agent's memory. The application user or developer can build and import concepts, models, term definitions and unified media related links. No programming is necessary. Some concepts can be attached to multiple media URI/URL.

To build the concept ontology, other standardized OWL editors exist: Protégé, Swoop, or any other that is OWL v2 compliant. OWL files can be imported into the agent knowledge base with our editor as well as exported to OWL (Menu). French and English languages are integrated. We also developed dictionaries importated directly into the database. The interest of our own editor is to build and modify model frames for the ontology of models that comply with EKRL syntax. The memory of agent is composed of three ontologies. Memory of agent is developed in an SQL database within 8 tables (Figure 3.35). One frame query is equivalent to one SQL query sent to the database and the matching is direct and very fast because of the complexity obtained from storing the ontology in database tables at the time of creation of concepts and models.

On Figure 3.35, schema shows all tables with their fields, these tables are grouped by categories:

- Nodes of the ontology of concepts and the ontology of models are in *nodes, labels,* and *links* tables
- Roles of models (event classes) and facts (event instances) are in roles-arguments *ra* table
- Natural language sentences attached to a node in *nlsentences* table
- Dictionaries, media and samples can also be attached to a label.
- Medias and samples are attached to a definition of the dictionary table.

Relations between primary keys are also presented in Figure 3.36.



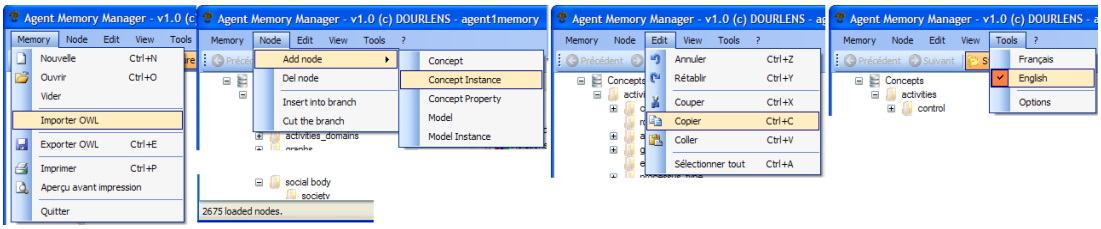


Figure 3.34 – Ontologies in the editor of the agent's memory

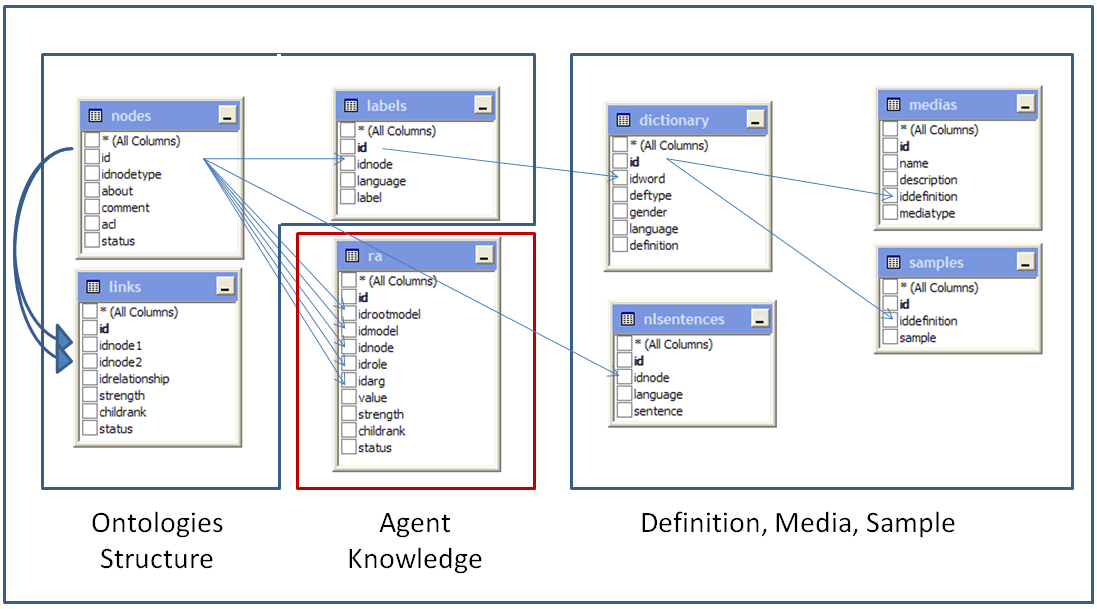


Figure 3.35 – Database Relational Model

Table	Description
Nodes	Nodes of the global ontology (ontology vertices) where a node can be a metaconcept, a concept, a concept instance, a model or a model instance
Labels	Unique reference name of a node
Links	Relationships between nodes (ontology edges)
Ra	Predicate model or instance (role-argument edges)
dictionary	Definition attached to a node
nlsentences	Natural language sentences describing a node
medias	Media attached to a node (picture, video, URL, …)
samples	Samples attached to a definition

Table 3.7 – Ontologies Database Tables

To store all ontologies, we will use database tables on the following Table 3.7. Roles and associated Operators and modifiers can be variable depending on runtime. But it is not really required to add meta-predicates to manage them at runtime because we have defined enough default roles, operators and modifiers.

These few close relations between concepts, models and questions (in the *ra* table) are the spine of our database system bringing speed, power and extracted meaning. Additionally, we have done an important improvement to make it faster, we assume arguments are sub-concepts of a concept because arguments are also words and concepts, so ontology designer must distinct the names of concepts of the names of attributes using a personal syntax.

3.7.5 Fusion Agent

A *fusion agent* is a semantic agent which has the role of extracting a specific meaning from several events. The matching operation in the inference engine

will use event models to extract the required meaning at a higher level of abstraction.

Once information is extracted from the knowledge base, the fusion agent will be able to create a composed event that will also be stored in the knowledge base and possibly shared with other agents by sending it to other agents in the network. Each fusion agent is specialised for a task or a domain to compose new events (of a higher abstraction level used in scenarios) or to act in the environment using linked semantic services. This specialization is performed by a composition agent. All fusion agents have a program as defined in Algorithm 1 in the previous section. The fusion agent will read its own code in its memory in the form of a scenario of actions to perform according to the affected role at that time.

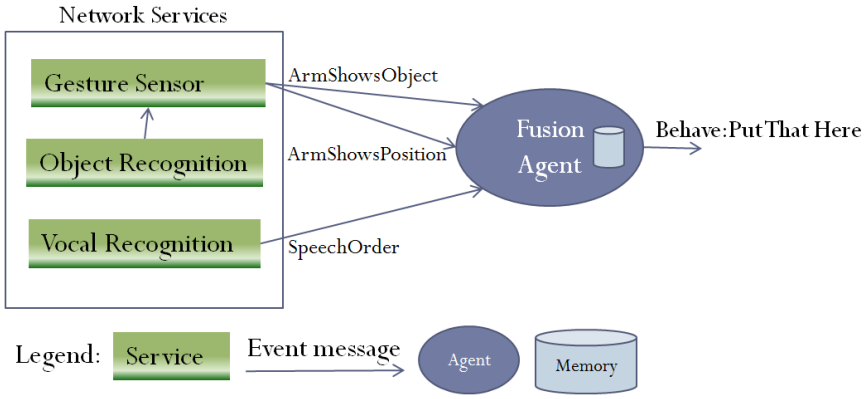


Figure 3.36 – "Behave:PutThatHere" events production

```
Behave:PutThatHere
    PRECONDITION: COORD(ArmShowsObject, ArmShowsPosition, SpeechOrder)
    SOURCE: COORD(GestureSensors, VocalSensors)
    DATE: date time
    LOCATION: location
```

Figure 3.37 – "Behave:PutThatHere" event model (fusion)

```
Behave:PutThatHere
    PRECONDITION: COORD(ArmShowsObject, ArmShowsPosition,
                        SpeechOrder)
    SOURCE: COORD(GestureSensor1, GestureSensor1, VocalRecognition1)
    DATE: 09/04/2010 10:04
    LOCATION: room 5
```

Figure 3.38 – "Behave:PutThatHere" fact

We return to our initial "PutThatHere" scenario. Figure 3.37 shows the fusion of events coming from two input services of the type "Gesture Sensor" and "Vocal Recognition". It's a composed event that represents "a human giving an order and pointing at an object and a location". One instance of our fusion agents is in charge of merging events that happen in the same period of time and are sent by two services embedded in the robot or parts of the house. Thus at a scheduled interval of time and after new events take place in the memory of agent , it can compose an instance of the *Behave:PutThatHere* model. Fusion Agent uses an event model that awaits three events: *"Behave:ArmShowsObject", "Behave:ArmShowsPosition"*, and *"Behave:SpeechOrder"*. Figure 3.37 is the predicate of the event model in EKRL, and Figure 3.38 is an instance of the event model (simplified version of Figures 3.32 and 3.33). Event instance (fact)

appears at 10:04, sent by the two service instances: The first service "GestureSensor1" is in charge of the detection of gestures produced by a human, it also uses the "Object Recognition" service to name the pointed object. The second service, "VocalRecognition1", uses one or more microphones to recognize a speech sentence. Note that GestureSensor1 is defined twice in the model because three events are needed even if only one service, "GestureSensor1", sent two different events. The interval of time for receiving events can be managed by changing DATE role by DATE1 and DATE 2 roles.

Services are also able to send their basic event in parallel with any other agents in one of two modes: directly addressed (BENEFICIARY role is specified), or to all agents (no BENEFICIARY role specified for broadcasting). The interpretation of events happening in an environment is very simple and fast. The matching operation that follows a query will give a true description of the event.

3.7.6 Fission Agent

A *fission agent* is a semantic agent which has the role of managing actuator services of the Nao robot. The fission agent acts exactly like the fusion agent except that it also produces orders sent to services to execute present or future jobs. Fission agents control hardware services or execute software services at a specific time. The agent model presented in section 3.6.5 is the same for the fission agent, but the meaning is of a different type because events will be orders or plans called "execution events" and sent to actuator services (to act physically) or software services (to execute a program).

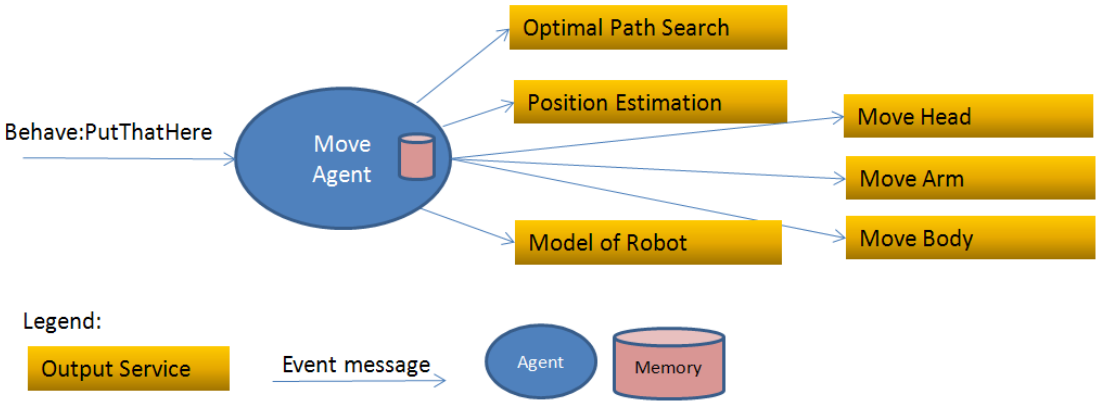


Figure 3.39 Fission Agent

Figure 3.39 presents an example of fission agent that manages different services to move the robot Nao in our "PutThatHere" scenario. This agent with the "*Move:MoveArm*" event can decide to move a part of the robot (head, arm, body) in order to take or move an object.

If the "*Behave:PutThatHere*" arrives and because it is part of the preconditions set of the "*Move:MoveArm*" event model (Figure 3.40), the inference engine of this agent will produce the order "*Move:MoveArm*" (Figure 3.41), store in its memory and sent it to services via the network connection. As we force the output modality via the role MODALITY in the event model, we specialise this fission agent to control the Nao robot. And then the "Nao1Proxy1ArmLeft" service receiving this order will choose a hardware arm (left or right) to take or move the object.

Move:MoveArm
SUBJECT: ALTERN(Behave:PutThatHere,Behave:DoMoveArm)
OBJECT: entities
TARGET: entities
MODALITY: NaoProxyArms
DATE: date time
LOCATION: location

Figure 3.40 – "Behave:PutThatHere" event model (fission)

Move:MoveArm
SUBJECT: Behave:PutThatHere
OBJECT: Book1
TARGET: Table2
MODALITY: Nao1Proxy1ArmLeft
DATE: 09/04/2010 10:04
LOCATION: room 5

Figure 3.41 – "Move:MoveArm" order

If we don't force this modality, meaning we don't specialise this fission agent, any other actuator of our architecture can accept the order and move the object for the reason that it is job-free or closer to the object. The elicitation mechanism is based on W3C standard web service composition.

3.8 Conclusion

In the previous Chapter 2, we have presented different scientific domains which can be useful for interaction with the environment. To complete this step of our work, we have listed here the necessary requirements applied to our architecture. We have given our own knowledge definitions and specifications before modelling all the components of the interaction architecture. These components are defined to perform an assembly at all levels: environment representation, information storage and retrieval, connection to hardware and software services, and communication. To ensure interaction, we derived the fusion and fission processes of the OOFDA operational model. We made the choices that we expected to give the best results to fulfil the requirements. Since agents and services are basic components based on web services, our approach is distributed, can store any information using predefined models of concept, event and question in agents. Incoming events or queries, and outgoing actions or answers, can be distributed in a concurrent functioning. Information can be exchanged with other components and web services respecting W3C standards.

The components of the designed interaction architecture are organized in a closed loop integrating the environment as introduced Figure 3.44. We have paid more attention on the knowledge representation language for the information managed. Concerning the EKRL choice, we justify its use by the expressiveness regarding to the natural language and the no limitations of the multimodal logic associated, in particular for the management of events which is a limitation of any existing inference engine like Pellet in standard tools like Stanford Protégé. We defined the items of the memory component used in our agent and service components. Memory of agents is composed of concepts, instances of concepts, models of events and facts stored in a database. Webservices are the basis of our components. SOAP/REST XML messages directly contain EKRL queries used in the inference/reasoning engine of our semantic agents. In the aim to develop interaction architecture, and in particular, to realize fusion and fission process

of all information coming through the architecture, we specialised semantic agents in fusion agents fission agents to extract meaning from input information (perception) to a global context and from this global context to output information (orders to act). Agents and knowledge are entangled at different level of abstraction in order to model any compositional semantic architecture.

In the next chapters, all actions or scenarios like composition and adaptation will be presented using EKRL models and instances. EKRL is the language to represent the environment and, the behaviour of our architecture and its components. In chapter 4, we will adapt our components to fulfil the problem of interaction and the inherent mechanisms like composition, adaptation and learning in a pervasive context. Lots of optimizations have been realized here in the design of agents. Inference system has been detailed to bring powerful search or query. We will check the measured performances in chapter 5. Next chapters will present our use cases scenarios of basic theories described in chapters 4 and 5. Application samples will show the interest of all these principles.

CHAPTER 4

PERVASIVE COMPUTING AND INTERACTION

« We have to live not in a new world of which it would be possible at least to make description, but in a mobile world, i.e. the concept of adaptation must be generalized to remain applicable to our society in acceleration » Gaston Berger

4.1 Introduction

In the previous chapter, we presented agent and service components and the knowledge representation language to describe the environment and the exchanged events to make components communicate. Perception and execution tasks are realized by input and output services. This chapter describes the mechanisms of interaction such as discovery, composition, orchestration, modality management, awareness management, integration and adaptation. In that perspective, we specialise the components of the architecture to manage the interaction. First, we will come back to the interaction architecture principles and present an example of a fully connected architecture for interaction. Second, we present the specialised components for composition and adaptation to make the architecture pervasive. Third, we will show how the architecture can manage the multimodal interaction. Interaction is realized with several modalities on simple examples knowing that we rest mostly on the abilities of each input service to synthesize a part of information (events) required in their area of expertise. We also study several types of awareness management like perception awareness based on several contexts of input modalities, and action awareness based on tasks and actions to achieve in planning. Fourth part is related to learning at different semantic levels like knowledge, agent and interaction in the agency.

4.2 Interaction Architecture

4.2.1 Introduction

The interaction architecture is built with services and agents previously defined in chapter 3. Agent is a semantic component manipulating only information (EKRL events), it is composed of a communication module, an inference engine and a knowledge base. The two possible roles of the agent is to extract meaning in a fusion process of incoming events from services or from other agents, and a deciding role in a fission process or decomposition of actions and orders to execute where orders and actions are output events. Agent may act at different levels of abstraction to manage knowledge at different level of granularity. Service is not semantic but can send EKRL messages by filling some models with perceived data and receive EKRL messages to execute tasks, hardware to act in the environment or software to run a function or a program. Next section presents the building of interaction architecture.

4.2.2 Interaction Architecture

Interaction architecture is a composition of agents and services (Lécué and Léger, 2006). An agency is a composition of agents. A semantic agency is a composition of semantic agents in order to produce higher level of abstraction events or lower level of abstraction actions. The composition is virtual because agents remain autonomous and free to collaborate or not between themselves according to their program (rule events for agents). Figure 4.1 present a network with agent and service components.

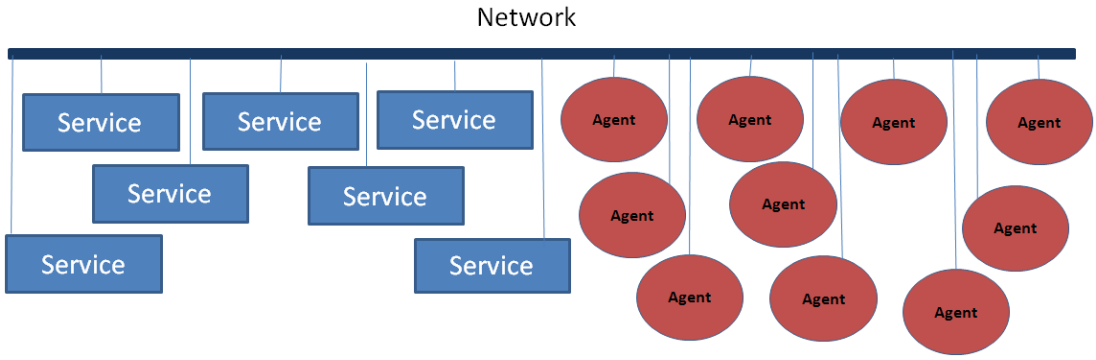


Figure 4.1 Network of agents and services

Below we give an example of an interaction architecture (Figure 4.2). This architecture contains an agency (a set of agents working together) to understand and reason about the environment and services to sense and execute in the environment. Agency is not uniformly composed of fusion and fission agents. We split agents of the network into two parts (the fission part and the fusion part). This possible agency can be embedded in a robot or any parts of the environment connected by a network. Here, the composition largely depends on the designer's or developer's choices and the implementation of the agents' program (rule models in memory).

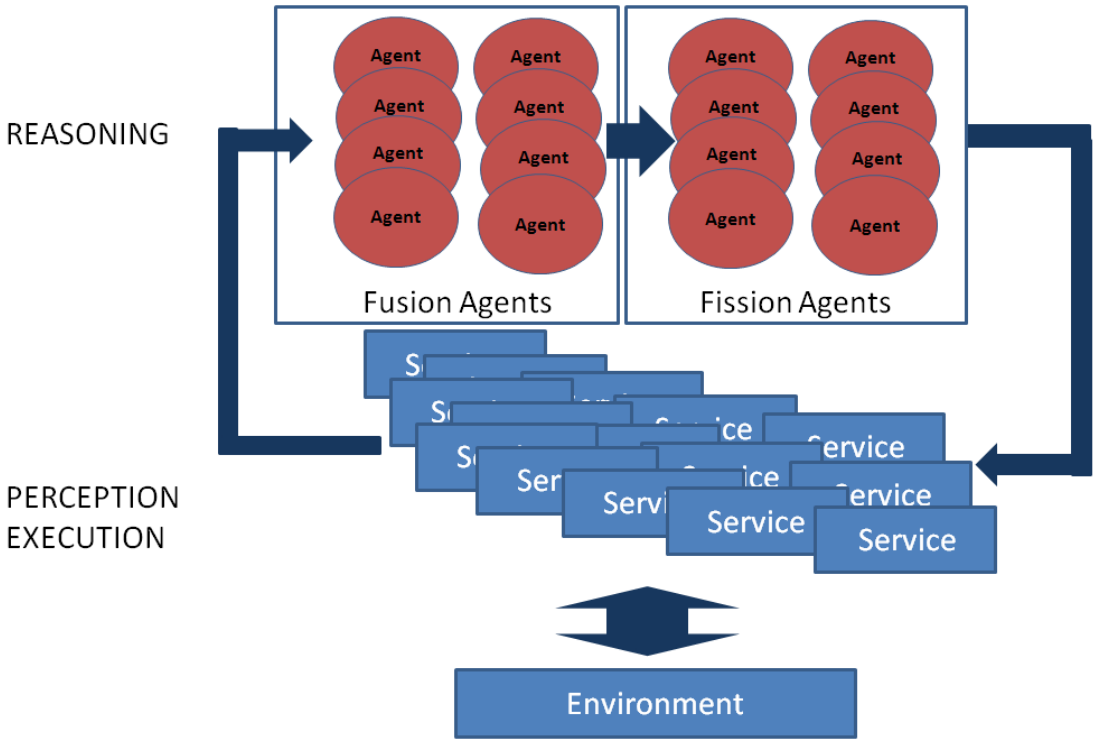


Figure 4.2 – Agents and Services for Interaction

Figure 4.3 is the same representation of figure 4.2 but we explicitly separate input services from output services. The interest is the emergence of the four important function of the interaction: sense, understand, decide and act.

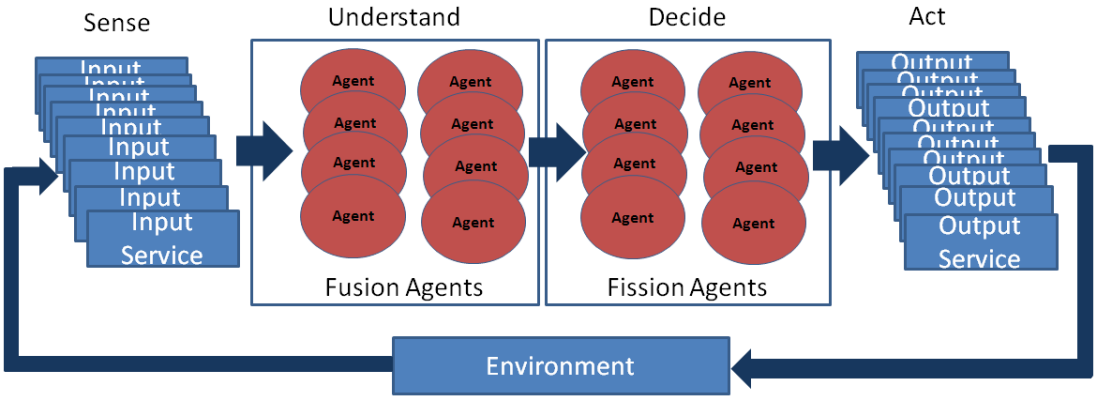


Figure 4.3 – Decomposition of the interaction architecture

With an aim of clarification, we separate the fusion part of the fission part in the next figures but the architecture stays similar to Figures 4.1 and 4.2.

4.2.3 Composition Example

Figure 4.4 and 4.5 present an example of interaction architecture to drive a robot of assistance to human for safety or caring. Sensor services allow detecting object information about location and objects in the room. Output services allow controlling room's temperature drive a robot, call for emergency help and make the robot or the house to speak.

Figure 4.4 shows a composition of input services and fusion agents. It is a more concrete example and the architecture is a composition of several fusion agents to build composed events at different layers of abstraction that will be sent over the network and stored by the memory of all other agents in order to be used for reasoning, acting, or adapting the architecture itself. As a designer, we give names to agents that are specific to the desired role. In this example, the *Context FA* (fusion agent) is in charge of reading all single events from the environment context: time, sound level, location building temperature, etc. The state fusion agent receives a focused object context from the environment by reading the object's temperature, activity status, name of the object recognized by the object recognition sensor. The *Shift FA* agent perceives basic events in the behaviour of the focused object: speed and direction. Other fusion agents (layers *entities level* and *behavioural level* in the figure) will receive extracted meaning for the previous layer under the form of event messages, to produce in their turn an event of higher-level abstraction. In this example, we have deliberately chosen to give the names of last-layer agents according to root predicates of the ontology. Each agent can then be dedicated to part of the whole ontology. This ontology is spread over several agents.

Figure 4.5 shows the second part of the interaction architecture, it is an example of composition of fission agents and the output services performed by three fission agents: *Settings* (fission agent), *Communication* (fission agent), and *Move* (fission agent). Agents are represented as disks and services as boxes, with arrows representing the communication of events to the agents. Past facts or previous events sent over the network are available in the agent memories. We notice two types of service: software services with useable methods and hardware services that drive actuators. The choice of software and hardware services also depends on the designed robotics application or composition.

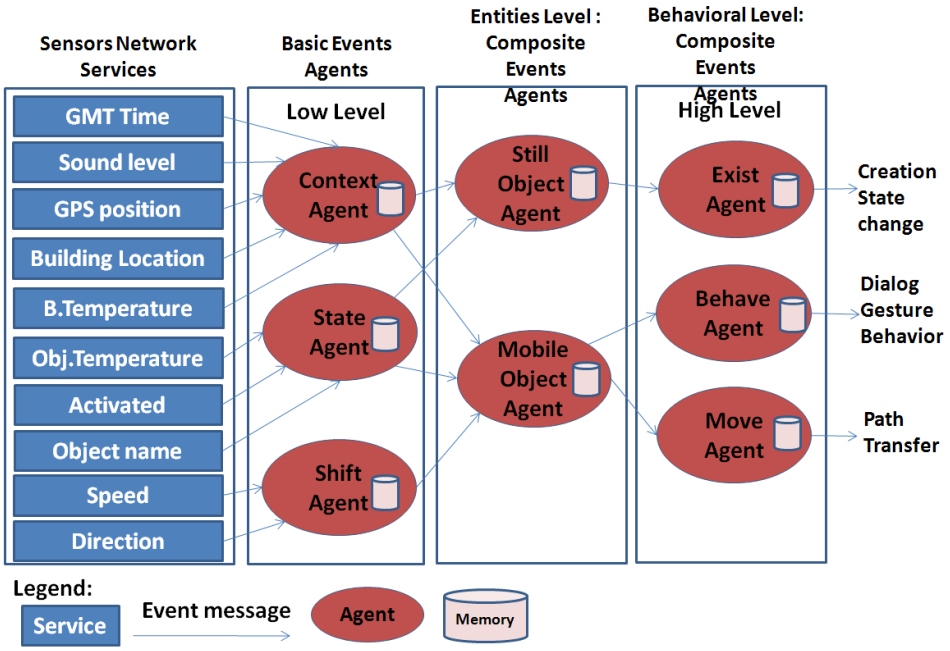


Figure 4.4 – Input services and fusion agents (first part)

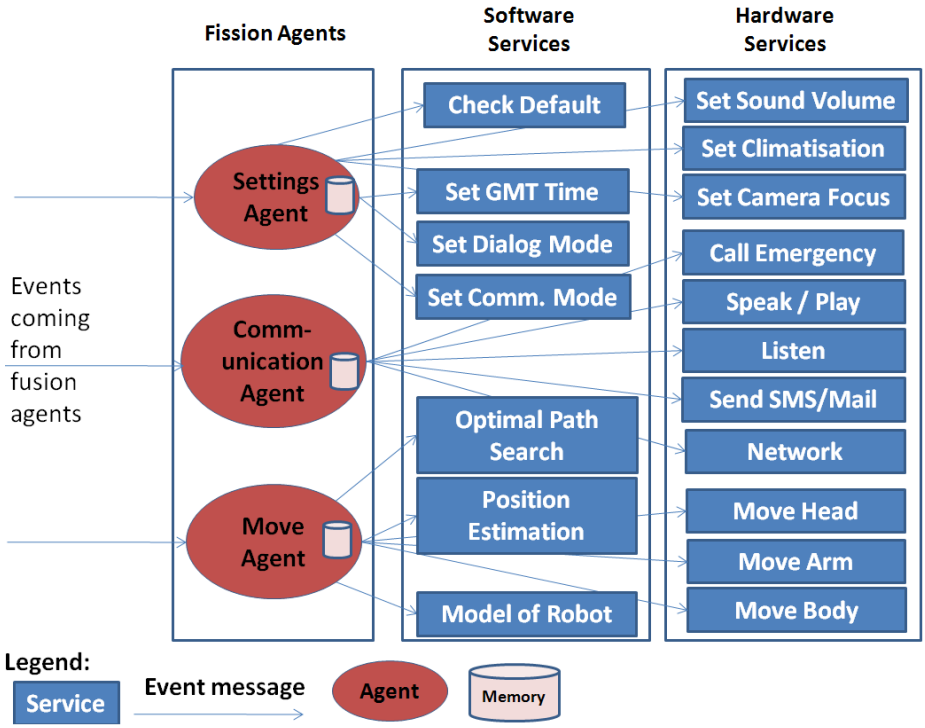


Figure 4.5 – Fission agents and output services (second part)

This short section was necessary to explain how to manually model concrete interaction architecture from the previously defined components available in the network. We introduced different notions like agency, abstractions levels of knowledge and roles of agents. Next sections will be more precise on the mechanisms of discovery, selection, composition and execution of services in a pervasive context.

4.3 Architecture for Pervasive Computing

4.3.1 Definition

Pervasive architecture extends the concept of interaction architecture to encompass worldwide web services. To make the architecture pervasive, we have to add discovery and composition mechanisms. Composition requires two important processes: discovery and selection. We have introduced the mechanisms of web services composition in Chapter 2. As our components are webservices, we may exploit these mechanisms but there is a major difference. It is the ability for some agents to take in charge of the composition. A second key point is the pervasive property to integrate any agents and services to the interaction architecture because, precisely, they are already web services as defined in Chapter 3. If any of the components on the network, whatever their location is, are declared in a UDDI server or WSDL files, they are available to make part of the architecture. The agency and the services can then be extended to a wide network. Another idea is that all agents can benefit from the expertise and resources of others without knowing how they will achieve their sub-goals. This section explains how pervasive architecture and ambient intelligence may be managed and will emerge in the architecture. Complex social organization emerges from self-organization according to previously determined or chosen objectives. In this section, we assume an agency is able to execute one coordinated or cooperative scenario at a time.

4.3.2 Architecture Composition

The architecture composition is an important mechanism for pervasive computing and pervasive environment to explore more space and actions in order to solve the tasks required by the user of the application. The composition (Bertoli et al., 2007) requires five steps as presented in the workflow of Figure 4.6:

1. *Discovery*: It consists in receiving any information about available services on the network for a future work. This operation is realized by a specific service (UDDI or WSDL files server), called *discovery service*, to collect services information and registration in a directory. To correctly start the application, this service should be known or should declare itself to the agents. To regularly update the memory of agents, a *composition agent* sends a discovery message to this service. This agent is a fission agent containing the "*Exist:Available Services*" and "*Exist:Available Agents*" query models of event. If the discovery service is able to send these events periodically without receiving a request, then the composition agent becomes superfluous. It only depends of the design of this service.
2. *Filtering*: Discovery service will send the list of registered services and their skills (EKRL models built in services). Answers are received by fusion agents. They will send the recognized actions that a service can do to the discovery agent. Retained models of services have to be among the possible models of actions in memory of these agents. Agents will extract meaning from incoming events to be sent as facts of the "*Exist:Available Services*" and "*Exist:Available Agents*" models of event.
3. *Recording*: The discovery agent is a fusion agent keeping a list of services and skills. The facts, coming from the discovery service, will be stored as

instances of the "*Exist:Available Services*" and "*Exist:Available Agents*" models of event. Discovery agent will send possible actions, which can be executed by services, to fission agents for selection to solve a task. These messages are EKRL models of actions to inform fission agents about the offered possibilities. Even without answer from discovery service, the discovery agent will received the "*Exist:Available Services*" and "*Exist:Available Agents*" events sent from the composition agent in the environment interaction loop and then will also send the required information. This agent internally plays the role of discovery service.

4. *Selection*: A fission agent also called *selection agent* must select output modalities or services regarding the tasks to execute currently. The selection agent, reasoning about goals and actions, will decide to send a query or an order in the network to target services. These tasks can be actions to execute in the environment by output services or queries sent to input services.
5. Responses: Regardless of the type of services (input/output or hardware/software), they will send answers to fusion agents respecting the interaction loop. The interaction will continue indefinitely. In case of the services send no answer, fusion will use other perception modalities to inform fission agents of confirmation or rejection of the orders, a meaning will be created about the situation and fission agents will again decide what to do to finally achieve the goal or inform the user that the goal can't be reached and what continuation should be considered.

Blue arrows are network messages, red circles are agents and blue boxes are services and environment.

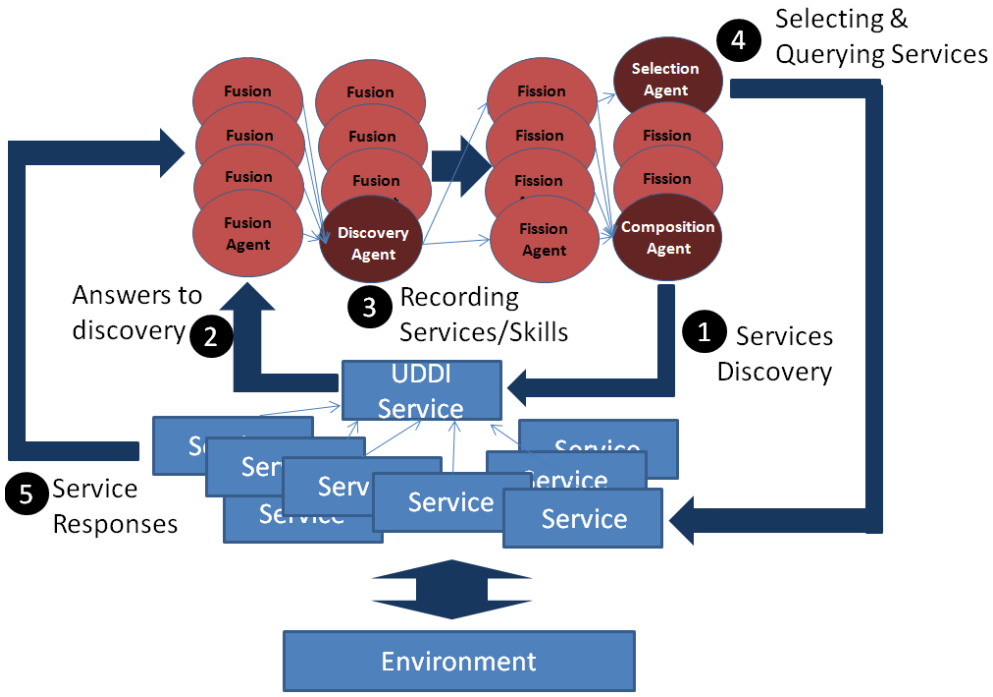


Figure 4.6 – Composition of services

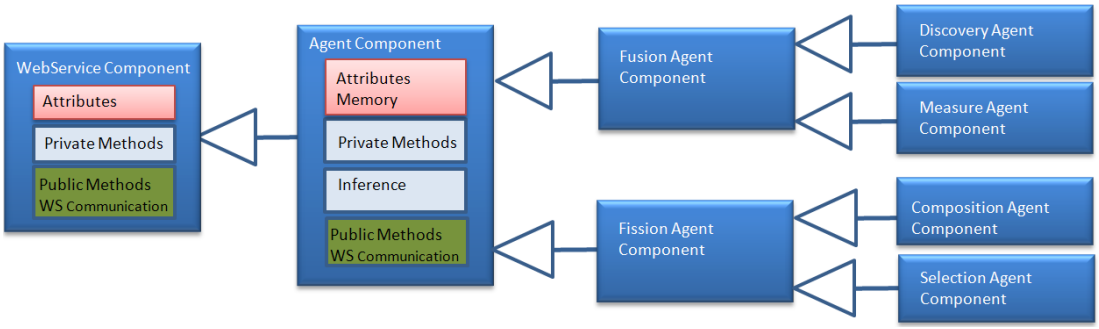


Figure 4.7 – UML Classes of agents

Composition affects all web services due to the respect of W3C standards. It means all components are concerned: agents and services. All components are pieces of software in the network and can be integrated into the architecture, they have to be registered in the UDDI server or share a WSDL file. Figure 4.6 shows the composition of services using services when there is a need. According to the content of their knowledge base, and more accurately the event models into the knowledge base to reason on environment, agents will be fusion agent or fission agent, discovery agent, selection agent or composition agent, and then will extract and send their meaning of the situation or the orders to execute to other agents or to services, independently of their positions and discovery in the network. If they are made to take into account an input event, a fact or a query, they will automatically store the event, reason and eventually produce a new event of higher abstraction level. This mechanism can also be used to decompose or unselect services. It is also possible to simply ignore events from some input services and never send orders to output services depending on the context or current situation. Figure 4.7 shows the new agent classes with inheritance.

Discovery Agent

A *discovery agent* is a fusion agent whose role is to find web services that can be exploited by fission agents for their requirements in terms of task execution and available services. It is a "meta" agent programmed to select and compose the architecture at a scheduled period of time in order to achieve goals or needs. It can act like an interaction manager and a modality selector.

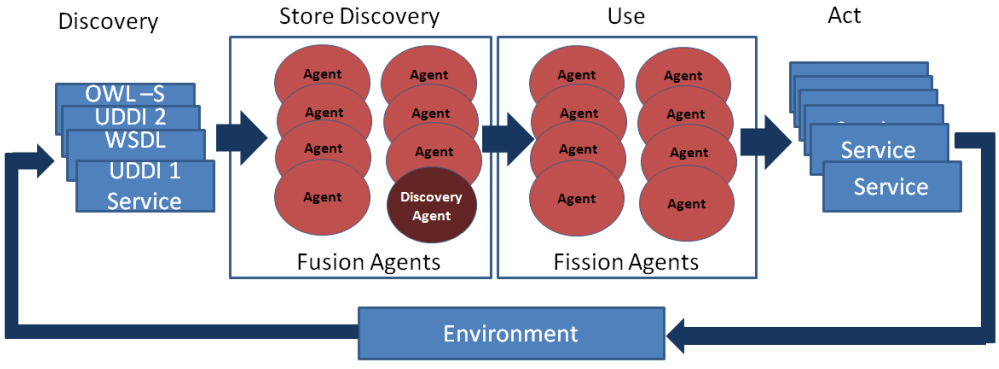


Figure 4.8 – Multimodal Services Discovery

Figure 4.8 shows a part of the architecture of agents and services with a discovery agent. Some input services have been designed with SOAP protocol to access UDDI server or WSDL files on the Internet (WSDL, UDDI1, UDDI2/OWL-S in Figure 4.8). They periodically access servers to refresh their

web services directory with all the web service properties and methods they provide. From these data, the services create events about available services and agents, their location, their possible roles, and their performance and send them to the fusion agents and thus to the discovery agent. The two event models to check available services and agents are in Figures 4.9 and 4.10. The first model checks existing services and availability of these resources, depending on the emergency and any other criteria or qualities.

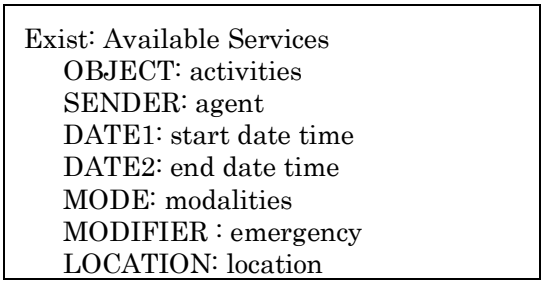


Figure 4.9 – "Exist:Available Services" event model

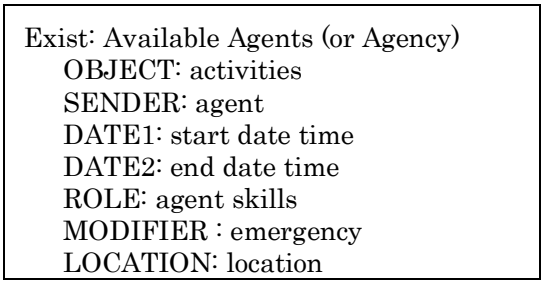


Figure 4.10 – "Exist:Available Agents" event model

The second model is only necessary to check activated or deactivated agents because the agent composition is automatic once agents are connected to the network. For instance, the unavailability of some agents may occur if the computer is temporarily inaccessible. It also can be useful for security checking if existing agents are well authorized to receive or produce events. We will see in the chapter 6 about applications how we use these models.

Depending on its program or the current context, discovery agent will inform fission agents about changes by sending them new events. Fission agents will be able to use these services. In an adaptation scenario, the discovery agent will be able to replace services with a better and more efficient one. This mechanism will be explained in the next sections.

Selection Agent

Selection agent is a fission agent who has the role is to select services of modalities, manage the orchestration and run the execution of tasks. Tasks are executed by services. Most often the selection agent is dedicated to a modality or a task. When an agent has to start a previously scheduled job or to plane a new job, due to the reception of events from other agents, this agent chooses the service and the action to achieve. The selection agent is directly connected with discovery agent and works in relationship with composition agent. It helps to the composition of the services in the architecture if it can't achieve its goal

alone. The job of the selection agent can be summarized in the following planning process:

1. List all possible scenarios or actions at the current state of the scenario (not started, started, paused, cancelled, or finished). This operation is a result of the work of all other fission agents concerning the task.
2. Get available service resources sorted by their skills and possibly by their quotation. Check if redundancy requirements can be used here. This operation is often given by the discovery agent.
3. Schedule a plan to achieve the goal. Each agent will receive an event from its modified schedule in terms of actions to complete or sub-goals to achieve. This operation corresponds to a set of models (the program) of the selection agent to achieve the task or to control a modality.
4. Monitor execution. Steps of the scenario are followed by fusion agents and when precondition are reached to go to the next step, stop a scenario or start a new task, the selection agent receives specific decision to apply from other fission agents.
5. If one or more agents are late or show failures, go to step 1

In pervasive composition, all fusion agents in our architecture could be able to select resources to help them achieve their goal. A list of possible scenarios or tasks that others can do may be found with a single EKRL discovery query sent to the network. The same process is used to check the schedule of required resources in the memory of agents .

To illustrate the selection mechanism, we can take the "PutThatHere" example. We want our architecture (fusion agents) to be able to detect "*Behave:PutThatHere*" facts. The model of event "*Behave:PutThatHere*" requires 2 modalities (visual and vocal) and at least 3 sensors (Human gesture recognition, Human voice recognition and Object recognition) managed by at least 3 services (Figures 3.36 to 3.38). In this example, the role of the selection agent is to select the two services "GestureSensor1" and "VocalRecognition1" where "GestureSensor1" can send 2 types of event (one for object recognition and one for pointing) and "Vocal Recognition1" sends one event with the recognized word. After the discovery, selection agent sends an order of connection or activation of these services. Events from these services are now continuously sent to fusion agents. If some events like *"Behave:ArmShowsObject", "Behave:ArmShowsPosition",* and *"Behave:SpeechOrder""* are received by fusion agents, then they confirm the composition. This was an example of composition of inputs services with fusion agents but it is simpler to select output services by fission agents. In figure 3.39, the "Nao1Proxy1ArmLeft" service is available and allows the architecture to control the left arm of the robot Nao1. All services of type "NaoProxyArms" are working to achieve the "*Move:MoveArm*" action in a more complex scenario. In figure 3.39, the choice criteria could be more stringent if we add more conditions in the role "Modality" of the "*Move:MoveArm*" model. For instance, the condition could be *MODALITY: Coord(NaoProxyArms, NaoProxyArms.MaxSpeed>.7, NaoProxyArms.PositionErrorRate<.2)* where *MaxSpeed* is the maximum speed of the arm move and *PositionErrorRate* is a measure of error between the desired position and the real position. This

condition is feasible because this information is directly available and accessible in the nao proxy.

4.3.3 Architecture Reconfiguration & Adaptation

Discovery agent, selection agent and composition agent allow the modification of the architecture by discovery, expansion and redundancy. This is a form of quick reconfiguration adapted to needs and to be flexible internally and externally (extending the scope of the network and looking for other agencies) respecting current standards recommended by W3C consortium. Many parts of the architecture can be adapted to solve one of these required tasks:

- *Exploring*, by searching resources like new possible web service components (by heritage, our services or agents)
- *Operation*, by sorting all discovered or used services by performance (measured by a specific service) with minimum time and resources
- *Coordination or Orchestration*, by sharing a scenario, managing centralized or distributed services by choosing the service with the best skills and sending to it the planning of the desired task execution
- *Monitoring* and evaluation of services and scenarios past or in progress

Web services composition provides an open, standards-based approach for connecting webservices together to create higher level scenario and expanding the architecture. Standards are designed to reduce complexity to compose web services, hence reducing time and costs, and increasing overall efficiency. In a point of view of requirements, it allows to invoke web services in an asynchronous way to achieve reliability, scalability and adaptability. It provides a dynamic, flexible and adaptable framework with a modular approach separating control logic of cooperation and coordination and the web service components.

Figure 4.11 presents a little more complex mechanism including the same principle (Figure 4.6) but with additional input services to manage qualities, user needs or preferences and contexts, and agents to take into account these new observations. In the figure, we have three types of incoming events: Qualities measured by the *Evaluation service*, Users goals and preferences given by the *User(s) service*, and situational or contextual events coming from any other services including the *UDDI service*. These 3 services can be specific to the developed system or can be discovery and then external. For example, the evaluation service can manage SGDS and ISO standard qualities measures. The evaluation service may use OWL-S for example. OWL-S is a OWL-based Web service ontology, which supplies Web service providers with a core set of mark-up language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-interpretable form. OWL-S mark-up of Web services will facilitate the automation of Web service tasks, including automated Web service discovery, execution, composition and interoperation. But the details of the evaluation service are beyond the scope of this thesis.

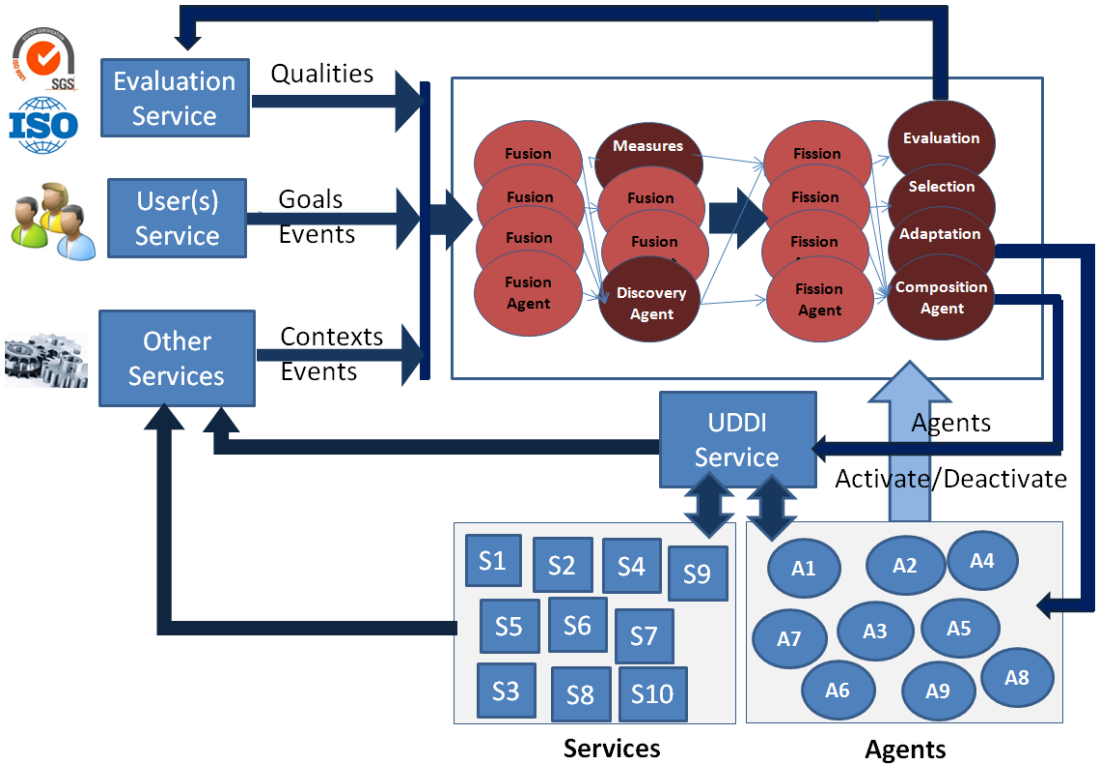


Figure 4.11 Architecture reconfiguration

We specify three new agents appear in the figure:

- *Measures agent* which is a fusion agent that captures a trace of all relevant measures extracted and compiled by other fusion agents.
- *Evaluation agent* which is a fission agent that uses the measures given by the Measures agent and the list of discovered services and their personal information sent by the discovery agent, to send it to the Evaluation service.
- *Adaptation agent* which is a fission agent that uses the measures given by the Measures agent and the list of discovered services to adapt the architecture by activating, deactivating or replacing a component. An agent can be deactivate the extraction of meaning is not enough relevant, a service can be replaced by another service more efficient or in case of failure.

On services, principles of composition are similar to the previous section but the selection mechanism is enhanced by the evaluation of quality of service. Agents are also managed by the UDDI service because they are also web services. In practice, the adaptation of agents of the architecture is simply activation and deactivation of agents, it means the conceptor created a set of agents available before the execution of the system and, over the time, the adaptation agent may augment the reasoning capacities. No change is made into the memory of agents in terms of models and concepts adaptation. There is no notion of learning for the moment.

4.3.4 Scenario Evaluation

Exactly in the same principle of the previous section (Figure 4.11), we will now be capable of evaluating a scenario. After a task is executed, a fusion agent in

charge to evaluate a part or the complete scenario (a measures agent) can send the results to the evaluation agent to store the qualities in the evaluation services. These qualities and quotations about the service and its performance to realize or not the desired action can be stored for future selection and use. So, according to the scenario to start or in progress, fission agent will be able to choose the best service candidate.

The interaction is realized in EKRL. To start an evaluation, which will be realized by the evaluation service, selection agent or adaptation agent may use the "*Produce:Evaluation Situation*" and "*Behave:Monitor*" event models in Figures 4.12 and 4.13. The following events show how the selection agent can interact with other agents in a precise location, a given context, a specific activity, and a period of time in order to compose an agency to execute a scenario. We assume a list of scenarios has been previously entered in the memory of the agent. The action or the scenario is an event model in this memory. A single action could be a "*Behave:Speak*" event sent to the service in charge of speech synthesis actuator (Figure 4.14).

```
Produce: Evaluate Situation
   OBJECT: contexts
   SENDER: COORD(Agents, Services)
   DATE1: start date time
   DATE2: end date time
   LOCATION: location
```

Figure 4.12 – "Exist:Evaluate Situation" event model

```
Behave: Monitor
   SUBJECT: COORD(Monitoring,Task Achieved, Failure)
   SENDER: COORD(Agents, Sensors Services, Actuators Services)
   RECIPIENT: agent
   DATE: date time
   LOCATION: location
```

Figure 4.13 – "Behave: Monitor" event model

```
Behave: Speak
   OBJECT: activities
   CONTENT: "Be careful of the object in front of you"
   RECIPIENT: SpeechActuator1
   DATE: 25/04/2010 14:05:34
   LOCATION: location
```

Figure 4.14 – "Behave: Speak" action

```
Behave:AskInternet
   OBJECT: coord(GoogleSearch,coord(WaitResult,Speak),Display,Speak)
   CONTENT: coord(query,coord("Waiting","Search in progress"),"Over")
   RECIPIENT: coordGoogleService1,coord(Timer1,SpeechActuator1),Monitor1,
                  SpeechActuator1)
   DATE1: 25/04/2010 14:05:34                    // start date
   DURATION: coord(0,coord(5s,3s), 2s,2s)        // from start date
   CONTEXT: User Query
   LOCATION: Office
```

Figure 4.15 – "Behave:AskInternet" scenario

A scenario is a rule model which is a composition of actions. Figure 4.15 is a scenario to ask a question to an internet search engine. Selection agent sends this event to all concerned services (in the *Recipient* role), the user query is embedded and sent to the search engine service. After waiting the answer, a presentation of the results is done and the user is informed by vocal message during the search and at the end. The first role of the event (*Object*) contains a coordination of actions (event models). In the figure, they are *GoogleSearch* for searching information in Internet, *WaitResult* to wait for some time, *Speak* to say a sentence and *Display* to show found webpages. Each event in the composition is executed one after another (comma separator) or concurrently (& separator or *coord* keyword). Once the scenario is established, the plan (the "*Behave:Askinternet*" scenario) is directly sent to all services which will act in the scenario with the corresponding date and time.

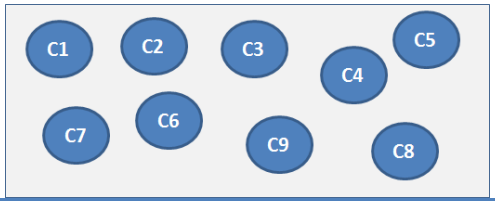
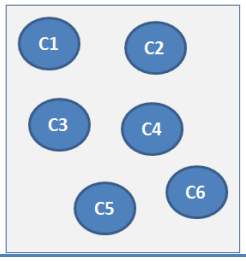
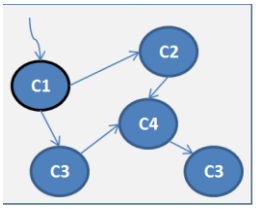
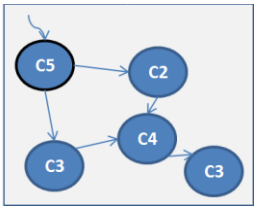
#	Step	Information
1	Discovery existing components	
2	Qualified components for the scenario	 Components c1=GoogleService1 c2=Timer1 c3=SpeechActuator1 c4=Monitor1 c5=YahooService1 c6=BingService1
3	Possible scenarios (in memory of agent) with c1 or c5 starting from c1	  Coord(c1, Coord(c2,c3), c4, c3) (Scenario 1) Coord(c5, Coord(c2,c3), c4, c3) (Scenario 2)
4	Execution d'un scenario	c1,c2&c3,c4,c5 c1,c2&c3,c4,c6
5	Evaluation du scenario	Q1=75%, Q2=84% ⇒ c6 > c5

Table 4.1 – Scenario and component Evaluation

Table 4.1 is a simple example of steps to evaluate a scenario or an action and even the components. The first step of the evaluation is now well known; it is the discovery of all existing components to compose a scenario. The second step is the filtering by selection agent of the existing components regarding the scenario to execute. The third step is the evaluation of the scenario using different components if we have enough information about the qualities of these scenarios and the components in the scenario. Else the selection agent will choose to

execute the scenario with the first component (c1 in the figure) and, the next times, it will choose the second component (c5 in the figure). Table 4.1 scenarios at step 3 are similar to Figure 4.15. The step 4 is the execution of the scenario until the display of information on the monitor. At the execution step, all information coming from the GoogleService1 is sent to the agency (via a "*Exist:SearchResults*" event), and then the content will be transferred to the monitor or rerouted to another modality if required by another multimodal context. For example, if the user is blind, the context will replace the monitor service by a speech service. This knowledge management can take place because of the use of the CONTEXT role in the *"Behave:AskInternet"* event. All events sharing this context can use content of it. The *User Query* context is a variable of the global context or situation *X* of the architecture. The figure shows the 2 executed scenarios after two execution times. The step 5 is the evaluation of the 2 global qualities Q1 for scenario 1 and Q2 for scenario 2 are measured. A quality value is computed from values of attribute such as functionality, performance, security, availability and possibly cost. In this example, the results of the evaluation are that scenario 2 is better than scenario 1 and we can also deduce component c5 (for example GoogleService1) is more efficient than component c1 (for example YahooService1).

In addition, accuracy and number of events depends on the period of time asked for and the current scenario. For example, the long term granularity of events will be larger, depending on the abstraction level. This means that if, for example, a human walks for 2 hours, we don't consider each step but only the followed path, or the start and end points. Execution in an agency is exactly the same as for a stand-alone agent but with the distribution of tasks to other agents involving more communication and monitoring. Along the same lines, an agency can get help from other agencies in the form of a recursive deployment of executable tasks to the appropriate agency that has the right skills, time, and energy to execute the desired goal. Social intelligence is a result of cooperation and increasing knowledge. It is very important to understand that any evaluation of the architecture is executed and evaluated by one or several services and sent to the agents which will store services qualities for further requests.

Once all components are chosen and interconnected (integrated), the orchestration defines the behaviours of the system: dataflow between components, execution priorities, and reactions of systems to events. In our approach, the programming and the distribution of behaviours is flexible, directly related to components; scenarios and communication are using EKRL.

4.3.5 Worldwide Agency Nodes

Services are located anywhere on internet. The incorporation of UDDI servers can help to reduce or expand the architecture by a composition of services. Until now, we have considered a set of agents (an agency) on the same computer, a same local network or a same location (room, house, building, and street). We also may give the name of agency node. As agents are semantic components, we may also call them a semantic agency. In fact, each agency node is part of a network working on a specific domain of knowledge with some different means to solve a problem. They can connect the other agencies in this network if they are open to other networks in different locations or made part of a more pervasive agency (Figure 4.16). It can be very advantageous to share this

knowledge and abilities (concepts, models and instances from the memory of agents) for many reasons like backup, security, replication, improvement, learning and so on.

There are two ways for an agency to exploit the knowledge of others:

- *Importation of knowledge* from other agencies is interesting for an agent, which can assimilate their experience and also compose and perform a new job with agents that are not too busy. This knowledge (concepts, models and relevant instances) is fully or partially transferred, or the two or more agencies accept to cooperate together. The first case is interesting if the knowledge is not changing too often and is not dedicated to the location. Each agency has fission agents with specialised event models customized to exchange information about their accomplished jobs and knowledge, we call them *information agent*. If we take the example of the Post office, the agency of the Post has to keep track of recorded deliveries and then inform people when they are arrived by email or by postman. They have to work for everybody on every delivery letter and package. It is not possible to give this work to a local agency at home for example. But it is possible to do the same work in a big company where some agents are dedicated to deliver mails to the different offices.
- *Exploiting*: The second case solicits a composition mechanism. For instance, Post office is a service we need sometimes: inform about a delivery package and timetable, or deliver mails. Our agency does not need their knowledge. The composition between several agencies uses the same mechanisms we have already introduced. The types of EKRL message may possibly change and be at higher level of abstraction than limited to services more basic. Manage and deliver mails is a complex task.

The way events are propagated, the expansion of the tasks in the scenario, and the use of agents in the whole network are potential problems. But these can be limited by application modelling (i.e., the human user) or automatically through the location of the required hardware sensors and actuators.

In addition, we assume that each agent, service, entity, and object in the environment has a unique reference name, which is an instance of a concept in the ontology of concepts of the agent's memory. Thus all event messages are signed. In this work, we shall not talk about the network security, which is important especially when controlling hardware with an open agency. But an access permission policy can be easily added replacing each reference name by a unique hash code or by using a cryptographic encryption according an inter-agencies agreement.

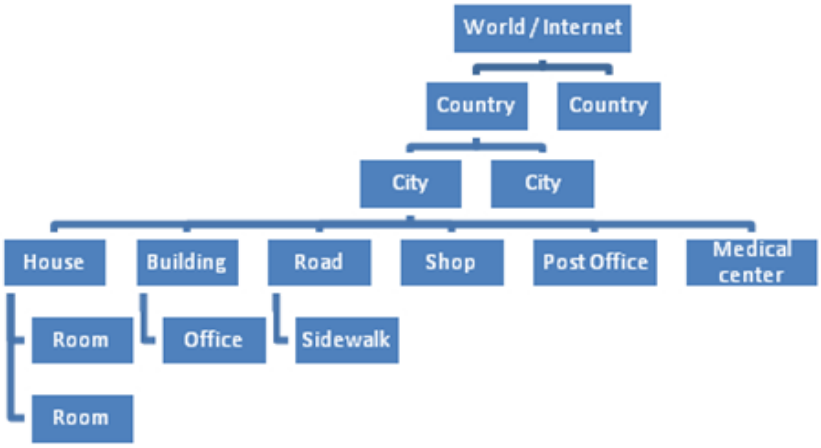


Figure 4.16 – Agency Nodes

Figure 4.16 shows a simple example of interconnected nodes in the world. Using the agencies of these nodes, a robot can help a human at home, provide navigation in a car or when walking to shops, company or the post office. Some agencies are accessible from home, like the medical centre which the robot will call in case of emergency. Some agencies associated with roads will be accessible for navigation from home to work, informing the robot about road traffic, and even giving the current colour of the traffic lights at the crossroads. The agency in the shop will give information about opening hours, the quantity of bread available, or what's new in fashion. The same goes for the post office, which could give information on packages delivered following an on-line purchase. At home, the network might inform the robot about the current health of the human, the daily living tasks in progress, or any abnormal situation.

4.4 Interaction Management with our architecture

4.4.1 Modality Selection

Modality selection is a mechanism allowing managing and choosing the best modalities in order to achieve a task, present information or dialog with a user in the human environment. We showed that our architecture has been designed to achieve a dynamic selection of services. A modality is a set of input and output services. Modality management just implies a higher level of abstraction. Our semantic agents are designed to manage several levels of abstraction depending on their event models. Thus agents are able to manage the modalities in real-time by taking into account multimodal contexts: user like health, disability; temporal like day or night, spatial like location, ambient like light, sound volume, dialog, cooperation, failure, and so on. They do this work inside the architecture by the exchange of EKRL messages. A modal context x is presented here as a part of the input vector P (read from sensors by input services) and after some refinements by some fusion agents, x will also be a part of the vector of the situation X which is useful for fission agents. Modality management allows the selection of modalities according to a current context (activate/deactivate a service), reroute or ask services a conversion of modalities to another more suitable, and control the active modalities. Modality selection is a part of the composition and adaptation mechanisms where modalities contain different input/output services. Modalities are directly managed by agents according to their rule models. These agents play 2 main roles: A role of

meaning extraction retrieving information about current multimodal context and a role of decision and selection of modalities by fission agents. Semantic rules used to achieve this goal are rule models in the memory of agents. For fusion agents, they work like the "PutThatHere" example (Figure 3.32 in section 3.7.3) and for fission agents, they work like the "AskInternet" example presented in Section 4.3.4.

To illustrate our discourse, Figure 4.17 shows an architecture with services and agents fully connected to manage hardware and software modalities. Agents (circles) in green receive events from input modalities, generate specific contexts related to this input modalities and then corresponding output modalities. To make this example clear, we suppose agents execute one specific job and level of abstraction. Fusion agents A_k build the context and Fission agents B_k select modalities according to the context (user context, ambient context, audio context, visual context). In the figure, two input modalities are sending information, *Audio* and *Vision*. For *Audio* input modality (i.e. audition), sound level is low and speech recognition works fine. That is why *Talk* service in the *Audio* output modality is enabled. We see in the figure that agents *A1*, *A3* and *A6* manage the audition context; agents *A5* and *A7* manage the gesture input context. Agent *B1* builds the message to deliver and agent *B4* sends the message in text format to two different modalities, Talk service has a text-to-speech functionality; Television Message service has the ability to control television and then display a computer screen or windows on the TV. In addition, as *Gesture recognition* is also working, agents propose to display a message or a menu on the television set and *Writing* modality is then on. *Human Detection* service and *Ambiance* modality are not available with a reliable confidence where confidence is directly provided by services. Many management scenarios are conceivable.

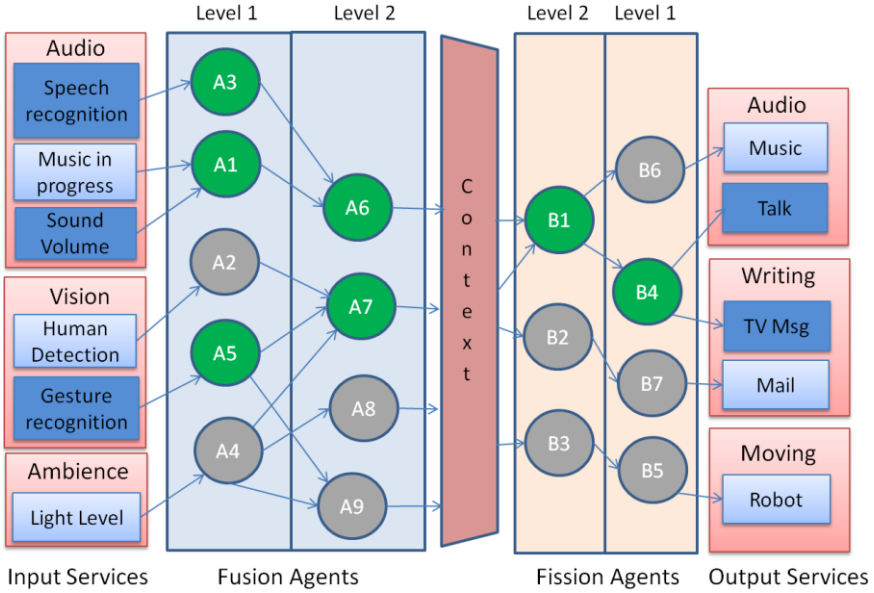


Figure 4.17 – Modalities Selection

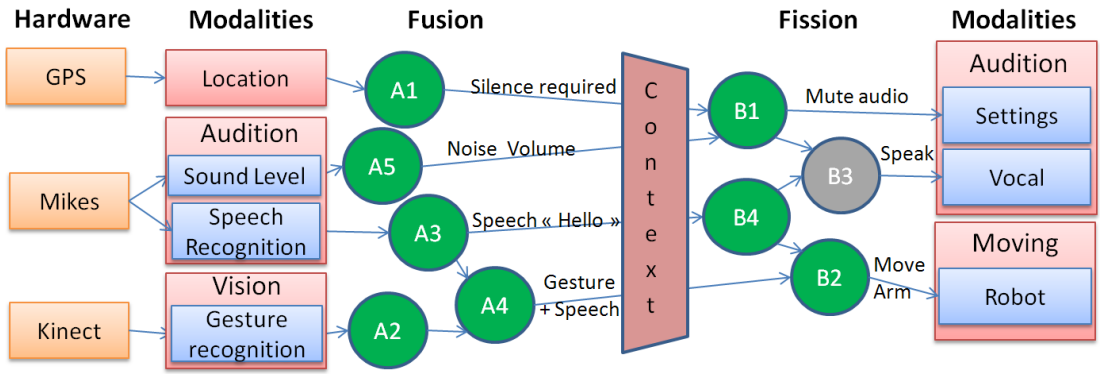


Figure 4.18 – Noise Level Management

Figure 4.18 is another illustration where noise context is used to manage output modalities. In the figure, the GPS sensor of *Location* modality sends an event to inform the system (more precisely agent A1) that silence is required in this location. Agent A5 interprets the current state of noise with the equivalent rule model of Table 4.2. At the same time, a user says "hello" and makes the "hello" gesture, the information is extracted by agents *A2* and *A3*. Agent *A4* couples Speech recognition and Gesture recognition events in order to generate the "*Behave:SayHello*" event. The noise context and user action are now available for fission agents. Fission agent *B1* understands that it must mute the sound of the audio modality. The direct effect is that the agent *B3* can't generate a vocal speech. Agent *B4* decides to return the greeting and then generates a "*Behave:SayHello"* order to agents *B3* (speech agent) and *B2* (robot move agent). Unfortunately, agent *B3* can't send an execution order. But fortunately, agent *B2* can move the robot arm to say hello. Table 4.3 is an abstract of the output modalities management according to the required level of noise at the current location.

Noise Level (dB)	Meaning
0-40	Quiet
41-50	Moderate
51-100	Noisy

Table 4.2 – Meaning of Noise Levels

GPS Location C1	Output Audio Mute B1	Output Vocal Speech B3	Output robot Moves B2
Quiet required	Yes	No	Yes
Quiet not required	No	Yes	Yes

Table 4.3 – Output Modalities

Furthermore, *Audition* modality connected to agents *A5* and *A3* is the same as the one connected to *B3.* But, in our case, the selection agents only control the output services of the *Audition* modality. It means that the systems will continue to "hear" what is happening in the environment and *Audition* input won't be muted. Output services are well managed by the extracted context and inputs are always being present to keep informing the system about the environment. It's a rule of good usage but, in some more complex cases, it is still allowed to mute the inputs. It is to the goodwill of the designers and developers.

4.4.2 Multimodal Interaction

Audio Modality

Audio interaction concerns recognized sounds and vocal messages in the environment where the scene takes place. Audition modality brought input and output audio services together. We want to show how our architecture can be suitable to audio modality management. We take the example where we have a security robot at home listening sound and moving its head to check if someone is there. Figure 4.19 illustrates the semantic event flow between services and agents of our architecture in recognition of barking dog integrated in the interaction loop between the system and the environment (read by sensors). GPS sensor of *Location* modality sends an event to inform the system, more precisely agent *A1*, that silence is required in this location. But a dog is barking in the street, agent *A3* interprets the current state of noise with the help of Table 4.2 and agent A5 detects that the type of sound is "Barking". A video camera is looking outside to possibly allow a user or another system to identify the sound's source. Distance of sound and angle of the video camera are managed by agent *A4*. This situation is composed of a noise context, a location context and audition context, and is managed by agent *A6*. It is now available for fission agents. Fission agent *B1* understands that it must mute the sound of the audition modality using the Settings service. Settings service manages audio channels selection, volume up and down, mute and so on. The direct consequence is that the agent *B3* can't generate a vocal speech. Agent *B4* can't inform the user with the vocal service but could use another modality. It is also in charge of video camera rotation so it sends a message to agent *B2* in order to move the camera where the louder sound level is. We design a kind of awareness loop.

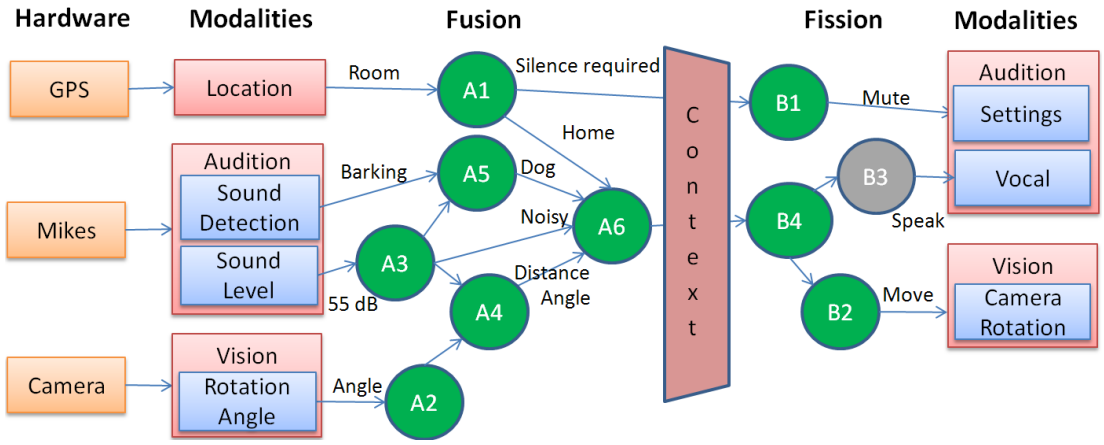


Figure 4.19 – Audio Management

The understanding process is always the same. After the capture of stimuli from sensors, using sound level and video angle to manage awareness, services compose EKRL messages and send them to our agents. Inference engine of agents will store information in the events ontology as a fact. Figure 4.21 explains the semantic relationships between the "*Behave:Bark*" event (Figure 4.20) and the knowledge base of agents, here, the agents *A4* and *B4*. The audio information is immediately connected to event models (Ontology of models), concepts (Ontology of concepts), dictionary and media in the database of an agent. One important key here is that the agent can store the model of barking sound as a reference to the media for a future use. Other services of audition modalities could use this reference compose and exchanges sound detection information, here, the dog barking sound.

```
Behave:Bark
   SUBJECT:  Brutus (Dog)
   CONTENT: Angle=320°, Distance=1500 m
   SOURCE:   Agent A4
   DATE1:    01/10/2011 20:45:31
   CONTEXT: Nuisance
   LOCATION:Home
```

Figure 4.20 – "Behave:Bark" Event

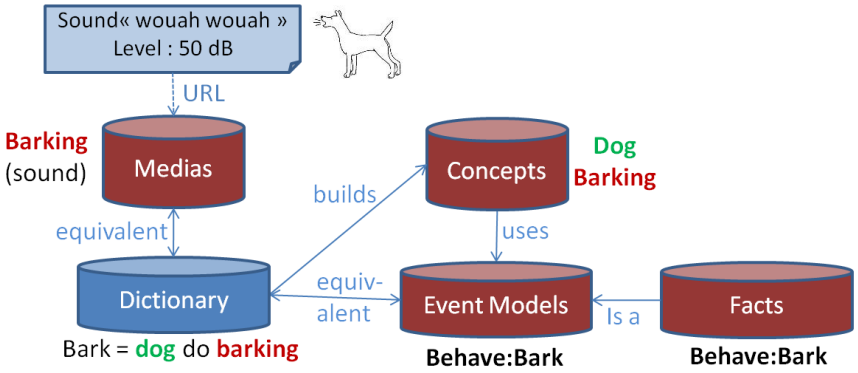


Figure 4.21 – Audio & Event Relationships

The architecture is compliant to manage and reinforce the audition interaction. As previously said, we introduced in the database a confidence factor which can increase with time and with this kind of reinforcement. We introduce the notion of awareness that will be explained more deeply in Section 4.3.3.

Vision Modality

We may apply the same principle to the Vision modality. Vision modality has different services at different levels of abstraction working on video pictures, colours and histograms (Low level image processing), optical flow (Middle level), gesture recognition system and shape identification (High level).

We set an example where entities (humans, horses, dogs) are moving, walking or running in a indoor stadium or a outdoor marked track with an unique armband to identify the entity. A camera with gesture recognition is able to send EKRL messages about entities movements. GPS service gives information about location (for instance, Olympic stadium, London). Date and time service gives temporal information. Video cameras recognize the gestures, identify the person's mark. The speed of the runner is taken at crossing points. After

reception of inputs from services, agent fusion sends a message "Jimmy is running" with subject, date, time, location and speed. (See Figure 4.22)

Move:Run	
SUBJECT:	Jimmy (human entity)
CONTENT:	Speed=15 km/h
SOURCE:	VideoGesture210
MEDIA:	\\dfs\storage7\215421524.avi
CONTEXT:	Olympic games
DATE1:	01/10/2011 10:14:31
LOCATION:	Olympic stadium, London

Figure 4.22 – "Move:Run" Event

Agents will store the fact in their knowledge. Figure 4.23 presents the semantic relationships between events and video media reference. The media can automatically be retrieved when querying the fact with a "*Move:Run*" query model. The reference of the video media about running is stored in the media table of the database. This media can also be transferred to any components.

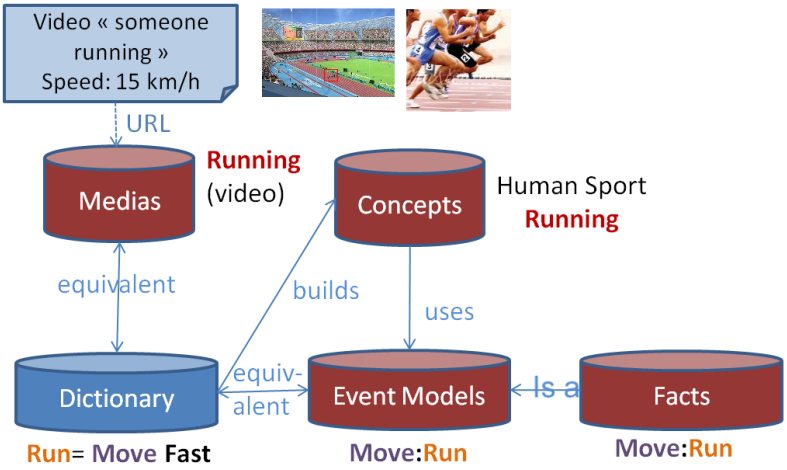


Figure 4.23 – Video & Event Relationships

Some services could use this media to compare with other videos to classify running samples. Shape of entities and speed are often enough to quickly deduce the entity and the running action (>10 km/h for human). But it depends on available modalities and sensors. Today many sensors use fast video processing.

Dialog Modality

The dialog is another case of interaction where system and human can talk together. Paying attention of classified speech acts (Searle 1975), we can summarize them into 3 kinds of dialog interactions between human and system: informative (answering, assertive or declarative), request (action, question, directive or commitment) and expressive (calling, courtesy and gestures). We may add practicing which is a form of learning with others or improving itself. We are more interested by existence of an assertion than by truth of this assertion. Inference engine will deal with corresponding modal logics.

```
Behave:Ask
   SOURCE:   Jimmy (human entity)
   CONTENT:  How are you?
   CONTEXT:  Kindness
   DATE1:    01/11/2011 10:01:05
   LOCATION: Versailles University
```

Figure 4.24 – "Behave:Ask" Event

```
Behave:Speak
   CONDITION: Behave:Ask
   TARGET:    Nao1 (robot)
   CONTENT:   I'm fine, and you?
   CONTEXT:   Kindness
   DATE1:     01/11/2011 10:01:08
   LOCATION:  Versailles University
```

Figure 4.25 – "Behave:Speak" Event

Dialog management in our loop architecture depends on the new input events and the past facts already in the memory of agents. Following a discussion consists to keep one or several dialog contexts by using a context name attached to the EKRL event. The dictionary can help to swap subjects. When a context is clearly identified in the event, inference engine will directly retrieve all events of the same context on a chosen interval of time.

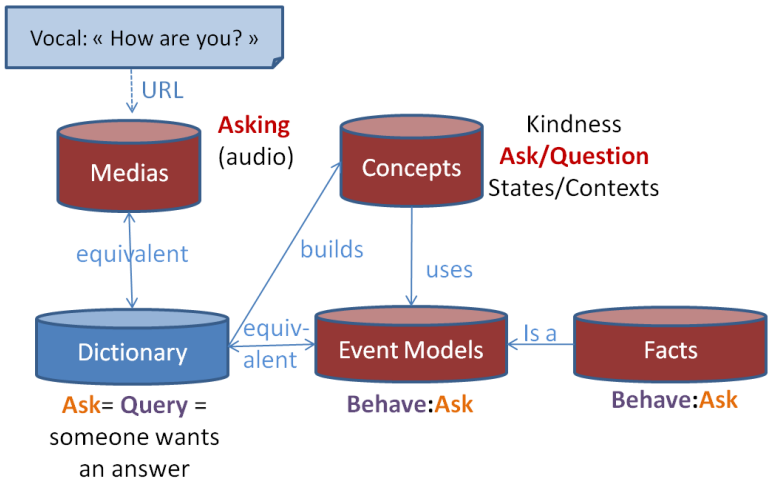


Figure 4.26 – Dialog Interaction

Figure 4.26 illustrates the semantic relationships between event and vocal input with speech recognition and output modality. It concerns an example where the robot listens to the human. Fusion agent send a "*Behave:Ask*" event to inform the system that Jimmy asks a question (Figure 4.24). The interpretation of the speech by agents is important to define the act of speech and then formulate an answer and or an action. The output modality also depends on the multimodal context. The answer can be vocal, gestural, written, and so on. Dialog and communication in general is not limited to vocal modality. For human, it is most often multimodal. System chooses to send a "*Behave:Speak*" event to answer Jimmy (Figure 4.25). A similar work of meaning extraction from dialog is presented in (Reckman et al., 2011).

Dialog Scenario

Let's assume we are in the "PutThatHere" scenario with 4 input services (corresponding to 4 different modalities) and 4 output services. Four agents must manage the dialog. Figure 4.27 presents the architecture. Arrows represent the dataflow of events between components. 2 services send gestures and vocal query of the user. 2 other services give more information about the user context (current health and room). In particular, the user profile contains user information stored in the global situation context *X* of the system like if the user is in a normal state, incapacitated or handicapped.

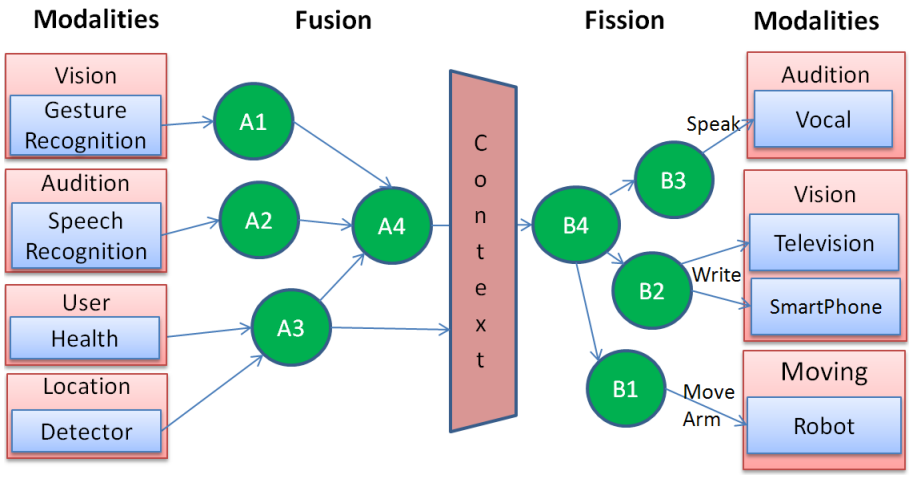


Figure 4.27 – Dialog Management

In Table 4.4, we wrote the rule models of each agent under the form "post condition events → triggered precondition events". It corresponds to the semantic reasoning of agents. Fusion agents will take several events and compose one. Fission agents will take one event and decompose into several events. The fusion agents *A1* and *A2* from the first layer receive and store the 2 events, "*Behave:Point*" and "*Behave:Speak*" coming from the *Gesture Recognition* and *Speech Recognition* sensor services. They send their events to the second layer where the fusion agent *A4* is standing. Agent *A3* manages the user health and location in the house coming from the *User Health* and *Location Detector* services. Agent *A4* receives the events from agents *A1*, *A2* and *A3.* Its inference engine finds a matching with the 3 events "*Behave:Show*", "*Exist:HearingLevel*" and "*Exist:AvailableModalities*" which are sent to the fission agent *B4*. . User query and user context are parts of the global context. *Coord* and *&* are the symbols used for the *and* operator, i.e. a temporal or spatial coordination for the inference engine of the agent. Agent *B4* can now decide what is the best modalities (here "Audio" and "Video") corresponding to the user context. It sends the "*Produce:Message*" to be displayed and/or said to agents *B2* and *B3* in charge of the video and audio modalities, and sends the "*Move:MoveArm*" event to the agent *B1* controlling the *Move* service (a Nao robot proxy).

Agent	Rule Models *Preconditions → Postconditions*
A1	"Point"(gesture) → Behave:Point(Object)
A2	"That" (vocal) → Behave:Speak("That")
A3	"Health=bad hearing" → Exist:HearingLevel(Value=Low) "Location=livingroom" → Exist:AvailableModalities(modality= Coord("Video","Audio"), location="livingroom")
A4	Behave:Point(Object) & Behave:Speak("That") → Behave:Show(Object)
B4	Exist: HearingLevel(Value=Low) & Exist:AvailableModalities(modality=Coord("Video","Audio"), location="livingroom") & Behave:Show(Object) → Produce: Message("I take the object",coord("Video","Audio")) and Move:MoveArm(Object)
B3 B2 B1	Produce:Message("I take the object",coord("Video","Audio")) → Behave:Speak("I take the object") Produce:Message("I take the object",coord("Video","Audio")) → Produce:WriteMessage("I take the object") Move:MoveArm(Object) → Move:MoveArm(Object, coord("Nao1", "ArmLeft"))

Table 4.4 – Rule Models (Semantic reasoning)

4.4.3 Situation Awareness

Definition

It is a higher level function of the system, often considered as a tactical objective after situation assessment, i.e. situational understanding realized by fusion agents. We define situation awareness as the priority and attention management. It means fission agents will decide where to pay attention from some contexts. This attention can be focused on tasks to achieve, perception modalities or knowledge in "mental" visualization.

In Robotics and Control systems in general, it is very important to always know the state of the hardware system we command. In particular, we can't let the robot body parts to move freely. It is the same when a robot or a human have to use a hardware device or a tool. These multiple control loops, submerged into the global interaction loop of our architecture, require attention (Posner, 1980; Corbetta and Shulman, 2002). It is necessary to keep these states in a part of the global context to take continually in charge these modalities extending the architecture. The main reason is the possible risks of injury. In the same point of view, we have to keep the control of software agencies until the task to do is definitively achieved.

Awareness is used to focus on contexts, modalities or knowledge. It permits to orient a part of the architecture (agents) on goals with higher priorities and thus

focus on reduced memory and activity spaces to improve the quality and efficiency of the context-driven processing. We have presented how multimodal meaning extraction forms a global context X from different modalities. Contexts $x_i \subseteq X$, like task contexts, require more attention, some others have less priority. Each context can have its own awareness needs. Thus we have several types of awareness. For example, *Visual awareness*, based on visual context, helps to follow moving objects with eyes (video camera). *Audition awareness*, based on audition context, allows to focus on activities around us like traffic flow, speaker in the conversation and abnormal noises in house (gas or water leak), and also evaluate distance of danger. Health awareness, based on user profile and human medical models, is a lot more complex but rests with user context. *Perception awareness* is calculated from services sensing speed, direction and current location of an entity or an object and sending it in the *Content* role of a *Move* event to fusion agents. Task context or goal context is about planning of tasks. Looking at the required execution date (in late, close to now, far in the future) and space where to realize a task (location or distance), the *Action awareness* is realized by fission agent and integrated in the composite event to be sent to other agents and output services. Mixed contexts awareness is another type where awareness can be based on multiple modality contexts and knowledge context like objective, actor, source, destination, energy, health, and more generally, all roles which can be found in the predicate or our event models.

We will present how to develop Perception awareness, Action awareness and mixed contexts awareness with well-known and standard fuzzy logic. The idea is to make our multimodal architecture to focus on the priority between several contexts at a time. We show the importance of awareness in the "PutThatHere" scenario.

Perception Awareness

We want to manage the perception awareness by affecting values to objects of the current scene. Each object has a speed and is located somewhere close or far from a visual system (here it can be a video camera). We choose to develop a Sugeno-type fuzzy inference system (Takagi and Sugeno, 1985) to give an awareness importance to each object (Figure 4.28). Our experiments are made on Matlab 2010 Fuzzy Toolbox.

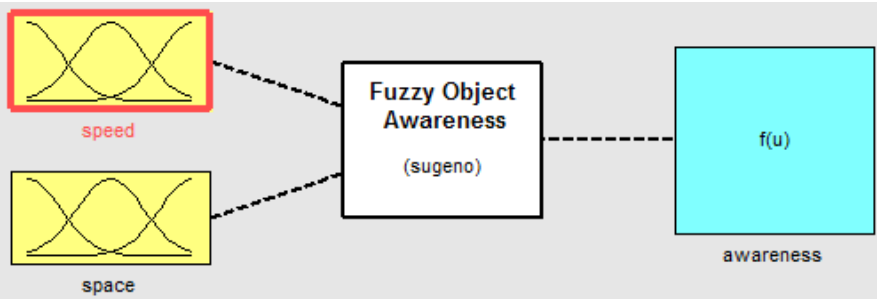


Figure 4.28 – Fuzzy Object Awareness

Speed/Space	Close	Middle	Far
Slow	High	Middle	Low
Middle	High	Middle	Middle
Fast	High	High	Middle

Table 4.5 – Awareness rules function to Speed and Space

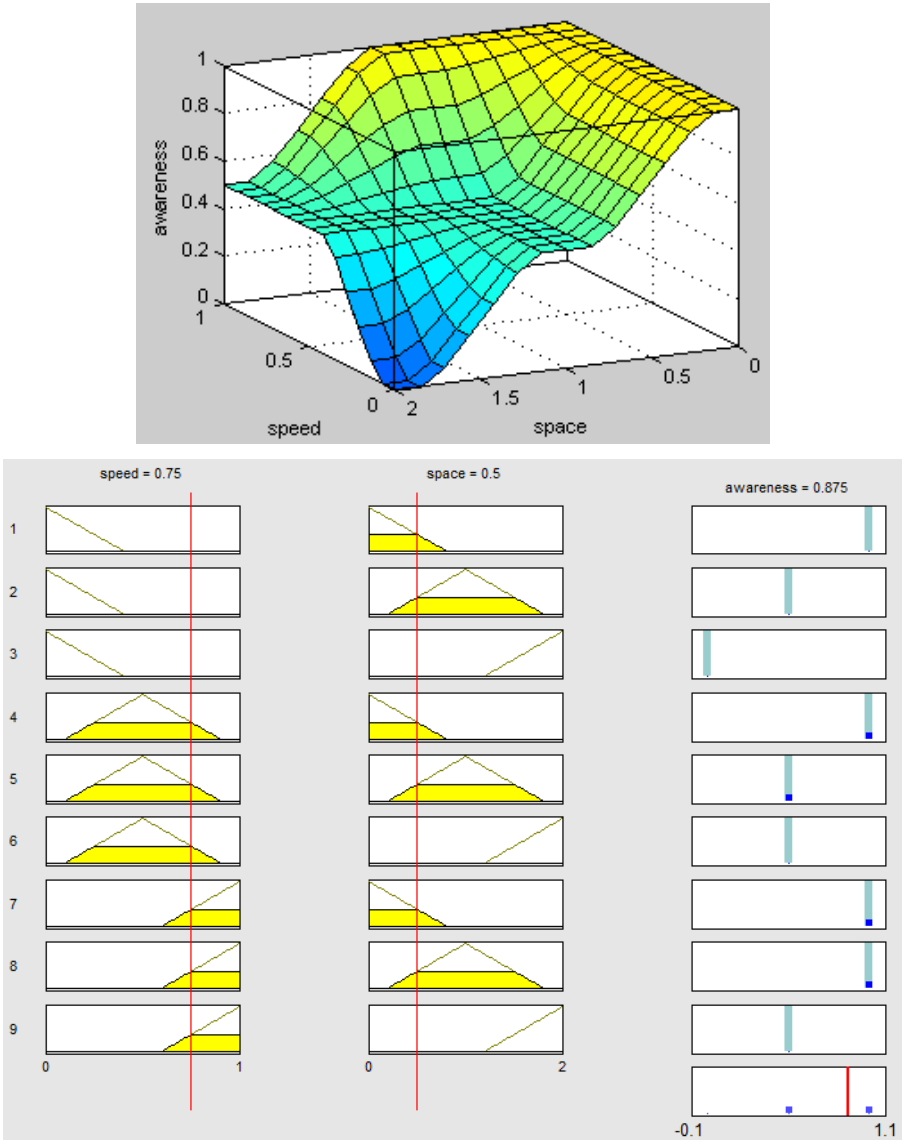


Figure 4.29 – Perception Awareness Curve & Rule Viewer

In this example, speed parameter is attached to a symbolic scale { "Slow", "Middle", "Fast" } and normalized values [0,1], and the space parameter (computed from location or distance) is attached to the scale {"Close", "Middle", "Far" } and normalized values [0,2]. We apply the rules in Table 4.5. We obtain the non-linear surface curve in Figure 4.29. Output is attached to the scale {"Low", "Middle", "High" } and is a normalized value in [0,1].

For instance, if an event, describing an object movement, indicates speed of this object is fast (.75) and current location of the object is at a short distance (.25) from the head of the robot in the room then awareness is high (.875) and priority of perception management increases. The trajectory of the object is checked to validate if the object can hurt someone or the robot, can be caught or not, and so on. If this operation is repeated in parallel for all objects of the scene, then we obtain a classification of perception priority. A second step of this operation is to reduce perception priority for all objects that have been used to seeing moving at a rapid rate such as a pet or a small robot vacuum cleaner (awareness/2).

Action Awareness

Here, we want to manage the action awareness by affecting values to actions to be started (planned), running or to be stopped in the current scene. Each task to achieve depends on the temporal context (date and duration, or starting date and end date) and spatial context (location where this task takes place). Action awareness is only necessary to plane and start a task. If the task requires to be stopped, another task will be planned to stop this previous task. Planning a task requires preparing a composition of agents and services before the task starts, the focus on a task in advance permits to integrate this anticipation time. As for the perception awareness, we developed a Sugeno-type fuzzy inference system to give an awareness importance to each object (Figure 4.30).

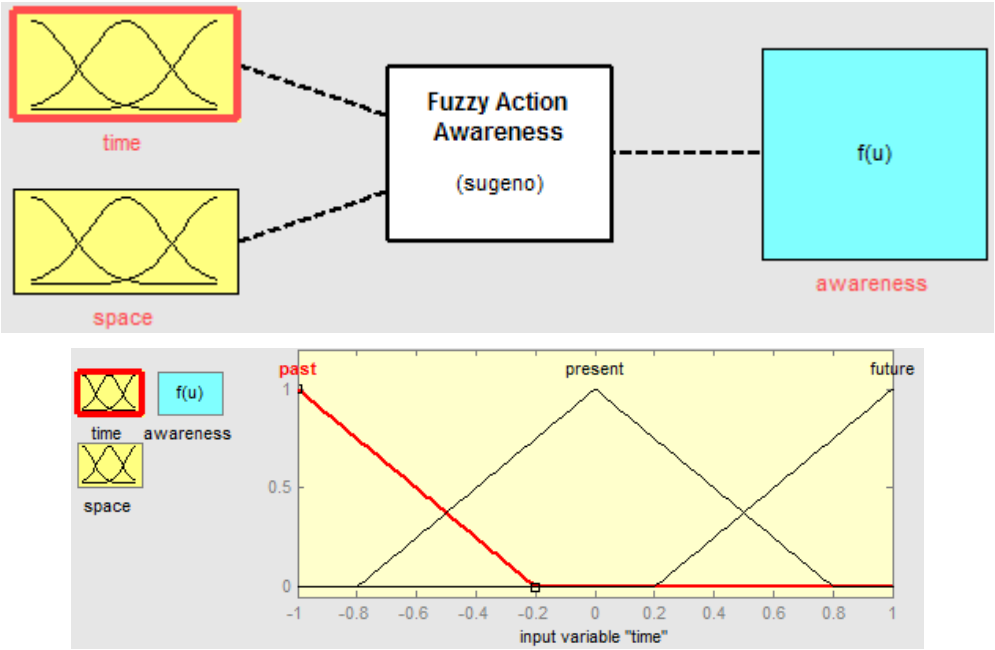


Figure 4.30 – Fuzzy Action Awareness

Time/Space	Near	Middle	Far
Past	Middle	Low	Middle
Present	High	Middle	Low
Future	Middle	Middle	Low

Table 4.6 – Awareness rules function to time and space

In this example, membership functions for the *time* input value are presented in Figure 4.30. Time parameter is attached to a symbolic scale { “Past”, “Present”, ”Future”} and normalized values [-1,1], and the space parameter (computed from location or distance) is attached to the scale {“Near”, “Middle”, “Far” } and normalized values [0,2]. We apply the rules in Table 4.6. We obtain the non-linear surface curve in Figure 4.31. Output is attached to the scale {“Low”, “Middle”, “High” } and is a normalized value in [0,1]. Rules can be modified by developer before running, or adapted by user preferences or a learning system at runtime. An awareness calculus is shown in Figure 4.31.

For instance, if an event, describing a near future action to execute, indicates time of this action is future (.25) and current space parameter (location or distance where action will be executed) is at a near distance (.25) from the head of the system or robot in the room then awareness is high (.729) and priority of action management increases.

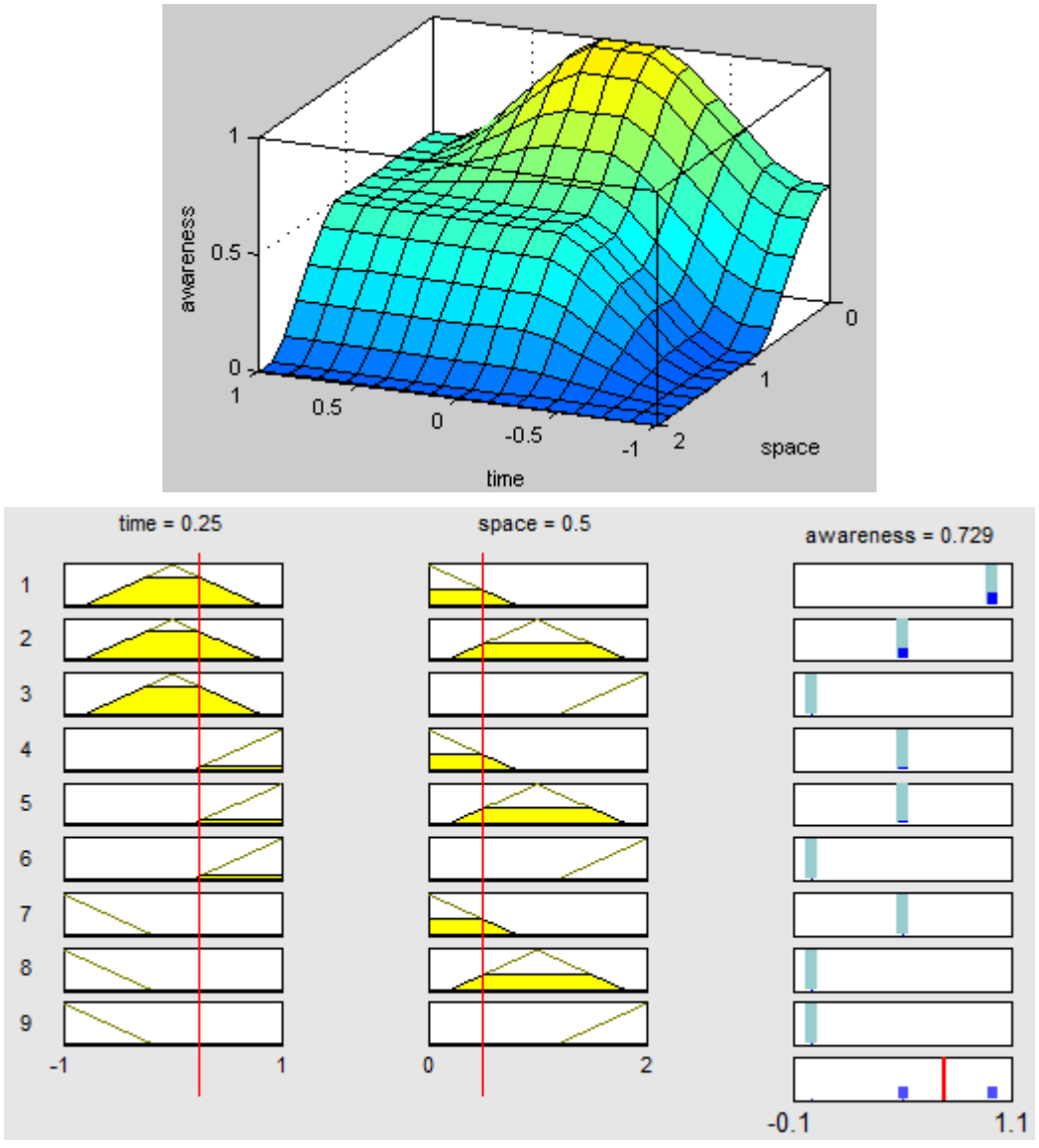


Figure 4.31 – Action Awareness Curve and Rule Viewer

Mixed Contexts Awareness

In most of cases, spatial and temporal contexts are not sufficient to calculate priorities and focus on perception or action. For example, in the previous case of action awareness, once a task is running, if this task requires more attention for any reasons (emergency, danger, required precision, and so on), the temporal and spatial contexts won't suffice, a task context is added to well manage the attention of the robot, the user or other systems. Multiple contexts must be taken into account. We propose to add normalized input values $0 \leq v_i \leq \alpha_i$ to our Sugeno box where v_i is a value of a context x_i and α_i is the importance factor of the modal context x_i. x_i is the i^{th} modal context belonging to the situational context X. for example, x_1 is spatial context, x_2 is temporal context, x_3 is user state context, x_4 is weather context, x_5 is noise context, all are included in X. Some contexts are more important than others, for example, the personal health care context. α_i is chosen by developer. Output awareness is a Sugeno-type inference. To generalize the method, we don't use our symbolic scales but we define the same n (n>1) membership functions $\{ F_1(),\ldots,F_n() \}$, equivalent to n linguistics variables, for each modal context x_i. Membership functions can be triangle, Gaussian or other. The total number of rules is $N=n^i$. For each rule k between 1 and N, the rule output is $z_k = c + \sum_j a_j w_j$ where a_j and c are constants.

In an instant context X_k, z_k is weighted by the following rule weights (firing strengths): $w_k = AndMethod(F_{k,1}(v_1),\ldots,F_{k,i}(v_i))$ for each modal context i, $AndMethod()$ is often the $min()$ function. And finally, the fuzzy box ouput is the awareness value:

$$A_t = \frac{\sum_k w_k z_k}{\sum_k w_k} \qquad \textit{(Formula 15)}$$

Figure 4.32 shows a simple example with the 5 previous contexts x_i with $\alpha_i=1$ (i from 1 to 5) measured at 2 different dates t_1 and t_2 (orange and blue). For each date, we obtain a surface, awareness will be the barycentre. We notice awareness A_1 is more oriented to space and sound, and awareness A_2 is more oriented to user health and temporal. The resulting architecture is focusing to different contexts and, more concretely, it will help fission agents to decide which action is a priority.

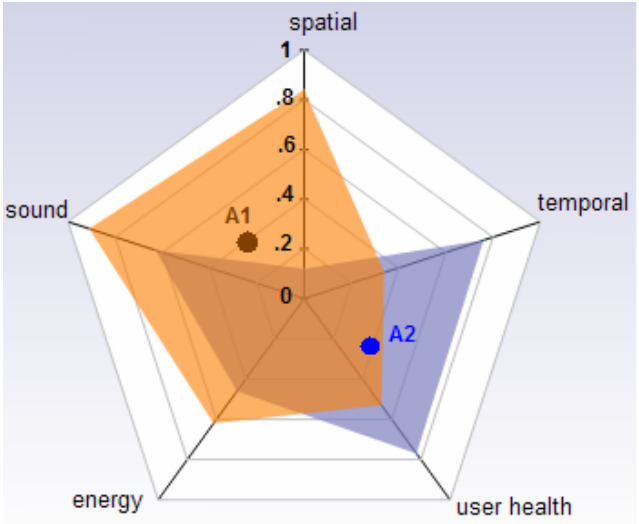


Figure 4.32 – Multi-contexts awareness

Priority of awareness and latency of adaptation should be manually set. We apply this operation to multi-contexts but it can be applied to multivariable perceptions, actions, goals, any qualities attributes or any decision processes to focus on. In conclusion, awareness can be applied to all information present in the global context contained by the higher level agents, the last layer of fusion agents and the first layer of fission agent.

Awareness in "PutThatHere" scenario

In the "PutThatHere" scenario, several problems of awareness appear. We have two perception contexts: visual and audition. In case it's a robot, we have to control the orientation of the head to make the video camera follow object to grasp (the ball) and object to put grasped object on (the table) and control the orientation of the head to make the microphones locate and listen the voice orders of human to hear "Put", "That" and "Here" (from the Speech Recognition service). That means awareness is swapping from vocal to visual context in time or is locked between the 2 contexts. We experiment the awareness on a Nao robot with a 2D moving head (Figure 4.33). It embeds 4 mikes (left, right, front, behind) and 2 video camera (up (0°),down (35°)). 2 motors allow changing pitch and yaw of the head.

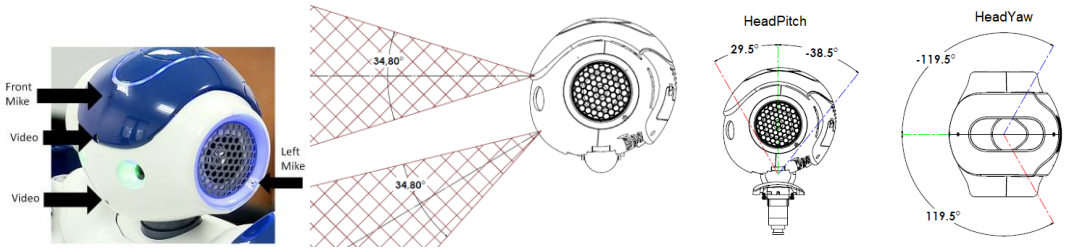


Figure 4.33 – Nao robot Head (Audio and Video sensors, Move Actuators)

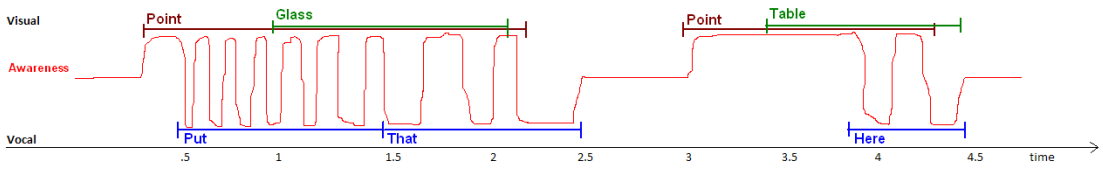


Figure 4.34 – Swapping Awareness

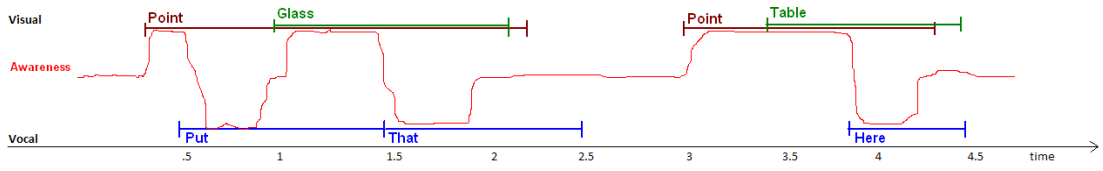


Figure 4.35 – Awareness based on event changes

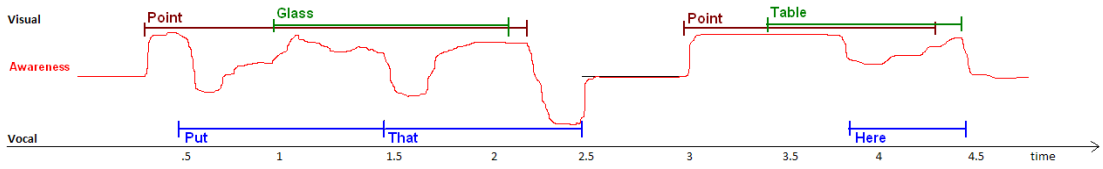


Figure 4.36 – Awareness vision-oriented

We measure values of awareness (red curve) varying between the visual and vocal contexts. Timeline is in seconds. Visual events are brown for gesture recognition and green for object recognition. Vocal events are in blue for recognized words.

Only at each new incoming event, the context is modified. But Figure 4.34, even if no new event appears, we observe a schizophrenic behaviour of the head. This is related to awareness instability oscillating between 2 contexts. After adapting the evolution of the variation δAwareness/δt, it is the case of Figure 4.35, we reduce the speed of the head move and fission agents act only after an event reception to keep it into a fixed stable area. The problem with these two previous methods of awareness management is that the detection of *glass* and *table* objects is not good, so we change vision/audition ratio to .75 (Figure 4.36). The quality of objects detection has been improved and audio is still quite good due to the orientation of 4 mikes and the environment is quiet. Awareness is necessary in this case, where 2 modalities are on the same physical support controlled by one output service (here, Nao proxy service), to orient the head on visual events. Other events than the "PutThatHere" one will require different awarenesses. It is also necessary to manage goals priority and knowledge priority. We try here to bring a semantic solution taking into account of multiple contexts but awareness also requires control theory and parameters adaptation to be fully integrated in the control loop. It means that a part of awareness should be cleverly distributed into the fission agents and the services. It could be a future work.

4.4.4 Semantic Learning

Introduction

Machine Learning (Mitchell, 1997) is of great interest in what it supposes little knowledge a priori, they enriched with experience related to the environment, in opposition to the traditional solutions, largely centred on knowledge a priori (or at least know-how). Learning methods are a kind of man machine interaction and learning often use communication acts with environment (Hoet and Sabouret, 2010). Learning and autonomy of a system is in fact the system's ability to evolve in a space of behaviours more or less defined by the engineer or scientist (Könick and Laird, 2006). These approaches help to develop creative solutions without the intervention of the designer but the knowledge of the search space. It is possible, based on current research in developmental system engineering to move towards systems capable of cognitive abilities like real learning, of dynamically constructing the meaning and objectives at several levels of abstraction, and so, to allow the system not to be *programmed* but to be *educated*. System can learn new tasks, new concepts and also new rules, as it is done today with a child. Because our systems have to exist in a human environment, social interaction becomes an important facet of research. Building social interaction into systems provides not only a natural means of human-machine interaction but also a mechanism for bootstrapping more complex behaviours. Humans serve both as models the system can emulate and instructors that help to shape the system's behaviour.

The objective of learning is the adaptation of the designed system. Adaptation of the architecture can be made at different levels: conceptual, behavioural,

functional and structural. Structurally, we have seen how to adapt the architecture with the discovery, composition and selection mechanisms based on specific agents. Services were more concerned by this adaptation of the architecture than agents. We have seen that agents have no need to be structured or fixed as they are autonomous, distributed and may communicate with each others. The link between agents in our architecture is made by the agent communication language and the rule models of the agent determining its position in term of level of abstraction. In reality, agents can work on any levels of abstraction. They can have a very large and complete knowledge base or a very simple one, according to the role and domain in which they operate. Two adaptations are possible for agents, the adaptation of their knowledge base only because inference engine and communication functions can't be modified, and the adaptation of the fusion and fission processes in their entirety. The first, *agent learning*, implies the modification of concepts, instance of concepts, and event models in the knowledge base. Most important operation is the modification of rule models, i.e. the behaviour of the agent. As for human, memorization is necessary; it is the support of learning. Stored facts are cases of reference. Models represented at different levels of abstraction can be used to quickly compare the base of facts to find analogies and transposed solutions. Conceptual learning consists to make concept ontology evolve. Behavioural and functional learning consists to make model ontology evolve. The second, *semantic learning*, will concern a set of agents which has to realize the process of fusion or fission. Semantic learning concerns the learning of meaning extraction realized by fusion and fission agents on events passing through our architecture and how agents are exploiting their semantic memory and inference engine to manage events.

Agent learning

As inference engine and communication function are fixed, agents can only learn concepts classes and instances, event models classes, behavioural rule models. But in most cases, for the sake of simplification and modularity, only rule models are adaptable. Concepts and models modification concern the issue of *ontology refinement* which it is not our concern in this thesis. In our work, we added the importation and exportation of concepts via OWL files until the root concept root of the tree of concepts classes to import has the same reference in an existing concept in our ontology. It is also possible to remove some erroneous facts by setting their confidence to 0 in order to validate theories. In rule models, conditions are in predicate role and parameters (numerical or symbolic) are based on scales. These key points are important and well defined in Section 3.4, 3.6 and 3.7.

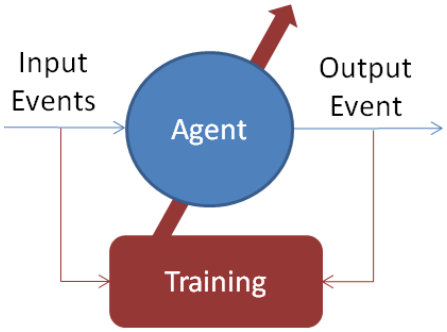


Figure 4.37 – Agent Learning

Most of the time, scales are corrects and fixed. Some missing facts can also explain an error in the production of a rule but, in this case, the mistake is not attributable to the agent. In conclusion, agent learning lies on rule models refinement, that is, agent programming by rule addition, rule removing or modification of arguments formulae. This operation must be applied when some expected events are not produced by the agent or when agent produce erroneous events. The training program must have a set of event examples and act as a supervised learning (Figure 4.37). As we have designed our component with a library block, we may add the learning function in the library block to store the part of *P, X* and *P'* information in the memory for future use. They are already memorized if the agent has the corresponding rule models. We can consider an agent:

- *Consistent* if there is no error of classification. Adding a concept or a model implies to know the class of concept or model and the name of this class which must already exist in the concept or model ontology. So the definition of our agents - knowledge base and inference engine - implies they are always consistent. This is guarantee by the fact that all events which have no model in the ontologies are rejected by the matching function of the inference engine.
- *Correct* if all examples are correctly classified. This is still the case as all facts are directly classified under its predicate model. If the model doesn't exist, no fact can be inserted and inferred. All roles and concepts must respectively exist in the Meta ontology and the concept ontology.
- *Complete* if all possible examples are represented. If the agents are specific to a task or if they are general but not able to manage all possible concepts and models of their environment, they are not complete. It is only the case in a perfectly known environment or if the agents always give the correct answer (produced event) for all the examples occurring in the scene. Our agents are rarely complete; it depends of the application and of the designer of the architecture.

Semantic learning

The ability to learn is a key aspect of intelligence and thus understanding. Let's assume fusion agents are producing a situation vector *X* from an input vector *P* and fission agents are producing a vector *P'* from an input vector *X*. Figure 4.38 shows how the learning box (light blue) could be connected to the interaction box (light blue), in an external system able to read *P*, *X* and *P'*.

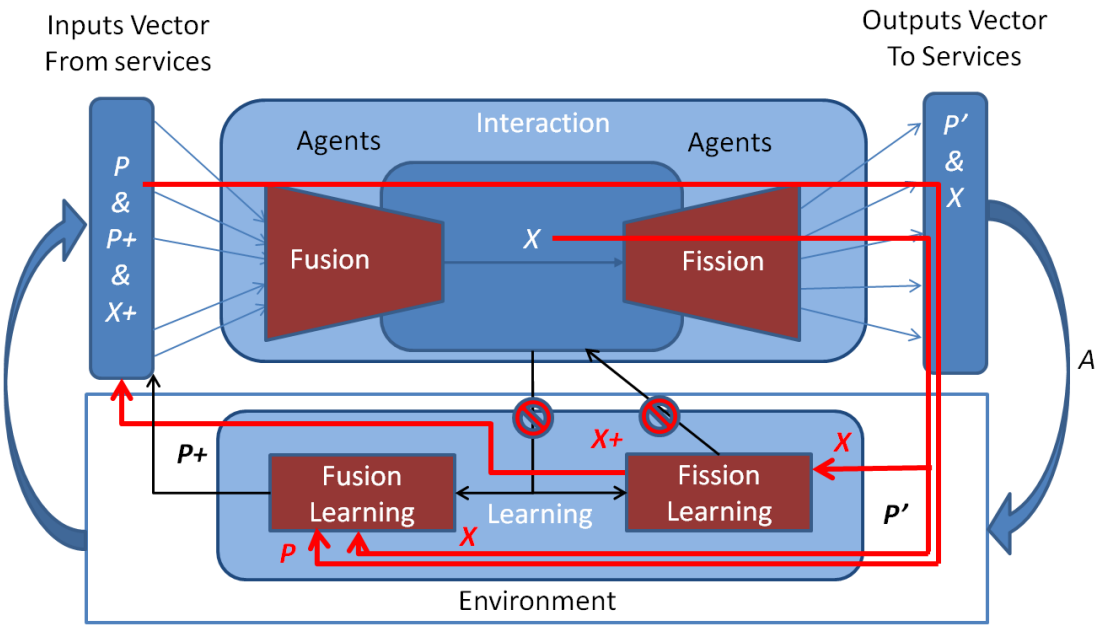


Figure 4.38 – External Fusion and Fission Learning

In Figure 4.38, at an instant *t*, *E* is the raw input events coming from the environment, *P* is the vector of predicates sent by services to fusion agents, *X* is the state or situation produced by the last layer of fusion agents and stored into the first layer of fission agents (see Figure 3.19), *P'* is the vector of predicates sent to service to act in the environment, *A* are the actions and plans controlled by services on systems or entities in the environment. The objective of the fusion and fission learning boxes (red) are to improve the quality of the fusion and fission boxes containing agents (red). Fusion learning has *P* and *X* as inputs and a fusion learning vector as output. Modification of fusion agents is done by adding motivation events into *P* in order to update the agents and then fusion learning can be denoted:

Fusion learning: $(P,X) \rightarrow P^+$ *(Formula 16)*

where P^+ is an augmented input vector to train fusion agents.

Fission learning have P' and X as inputs and a fission learning vector as output. Modification of fission agents is done by adding motivation events into *X* and then fission learning can be denoted:

Fission learning: $(X,P') \rightarrow X^+$ *(Formula 17)*

where X^+ is an augmented situation vector to train fission agents.

As *X* and *P'* are already produced before the learning time (loopback arrows in Figure 4.38), the update events X^+ and P^+ are actually applied at the next step of the global loop (system-environment interaction). Red bold arrows represent the obligatory paths for events in *X* and *P* in order to be in inputs of the learning boxes. It means learning boxes (in red) can find and modify the *X* information necessary to work (symbolized by the forbidden panels in the figure). In practice, the learning part of Figure 4.38 is not feasible as is to be an internal operation of the architecture, because it required agents that manage events in a reverse flow, and it cannot be as a part of the environment due to the necessity to get the situation *X* even if *X* could be added to *P'* and exploited by the environment. In real life, humans have all their own observations *O* of the environment. And more accurate is the perception, more it means that the difference between real

situation and observation is weak (X-O=ε where $\varepsilon \to 0$). This is still better when people are sharing the same language, culture, job, environment, location, situation and life. But in the other cases, ε can be *>>0,* i.e. lots of misunderstandings occur requiring further explanations to elucidated the true meanings. To train the whole system, it is also possible to add an external teaching component (human or not in the environment) which only requires a perfect outcome P' for all presented input P.

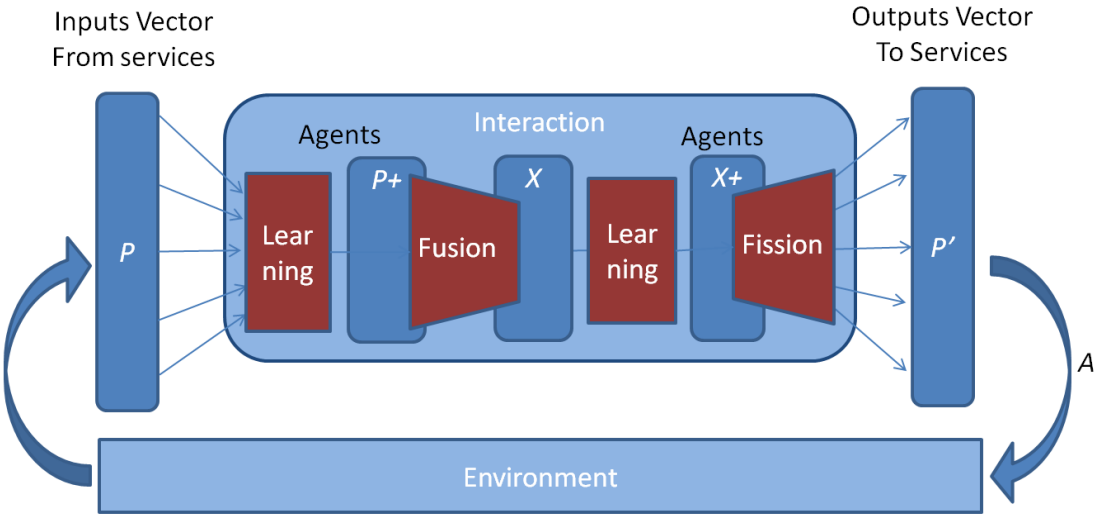


Figure 4.39 – Internal Fusion and Fission Learning

Otherwise, a solution is to integrate the learning boxes into the architecture (Figure 4.39) under the assumption that the information from situation X is well preserved by the environment or well transmitted to perception P for future interaction. It is often the case of learning or improving actions performed in the environment, the environment keeps the trace of the last facts.

In this case, at discrete step $k+1$, we can denote:

$$P^{+}_{k} \leftarrow Learning(P_{k+1}, [P_k,] X_k \subseteq P_{k+1}) \qquad \textit{(Formula 18)}$$

where P_k has been kept in the learning box at the step k.

and $$X^{+}_{k} \leftarrow Learning(X_{k+1}, [X_k,] P'_k \subseteq P_{k+1}) \qquad \textit{(Formula 19)}$$

where X_k has been kept in the learning box at the step k.

$P+$ and $X+$ are P and X augmented with the training events. Learning boxes contain agents able to interpret errors in produced events by the fusion and fission agents of the step k and possibly $k-1$, $k-2$, ...,$k-n$ according to needs. At step $k+1$, these training agents will send additional events to change internal rules of some fusion and fission agents.

Learning boxes use classical learning algorithms to measure the error and send the correction to be applied. Target agents must have a rule model to apply these corrections. The 3 correction events are "*Exist:AddRule*", "*Exist:DeactivateRule*", "*Exist:ModifyRuleCondition*".

It's not our assumption, but if we wanted to modify concepts and models, we could add these 4 correction events: "Exist:AddConcept", "Exist:ModifyConcept", "*Exist:AddModel*" and "*Exist:ModifyModel*". Note that we can't delete a concept or a model at runtime but only modify or adapt it.

Reinforcement learning applied to our architecture

Reinforcement Learning (RL) is a technique to train a system directly connected to the environment, i.e. the system is learning while interacting, which can give a reward for the actions previously done by agents (Sutton and Barto, 1998). It is learning by experience. The main advantages of RL over other methods in system learning and control are:

- There is no need to know a prior model of the environment. This is a crucial advantage because in most complex tasks a model of the environment is unknown or is too complex to fully describe.
- There is no need to know previously what actions for each situation must be presented to the learner.
- The learning process is online by directly interacting with the environment.
- It is capable of learning from scratch.

RL can be applied to manage tasks in a very simple way. The basic elements of the RL learning system, in our case of predicate frame, i.e. role-argument representation as a vector space consisting of a fixed number of independent dimensions, are:

- An agent or a group of agents that perceives the environment (typically called state *s*) and behaves on the environment producing actions *a*. $s \in S$ and $a \in A$ with *S* the set of states and *A* the set of actions. State *s* is a part or our entire input vector *P*, a set of incoming events which are under the form of predicates *p*. And action *a* is a part or our entire output vector *P'*, a set of output events which are under the form of predicates *p*. All predicates *p* are models in the model ontology.
- The environment in which the agents live, that could be a simulated world or the real world, including our services.
- A reward signal *r* that represents the evaluation of the state and it is used by the agents to evaluate its behavioural policies. The reward can be directly given by the environment via a service sending a clear reward event or by the learning boxes evaluating the situation of the environment and composing a reward or a correction event (in *P+* or *X+*).

The formulation of reinforcement learning must include three aspects: perception, action and goal. It can be expressed as follows:

- A hypothesis or fact space *H* describing policy functions $\pi: S \times A \rightarrow [0,1]$
- A reward function $R: S \times A \rightarrow \Re$ *(real)*
- *A* transition function $T: S \times A \rightarrow \pi(S) \Leftrightarrow T: S \times A \times S \rightarrow [0,1]$
- An optimization criterion *J* that enforces any policy π and a reward *r* to a function value such as $V_r^{\pi} : S \rightarrow R$

RL problem can be described formally as a Markov Decision Process (MDP) (Puterman, 1994; kaelbling, 1998), i.e. a 4-tuple <S,A,T,R>. $\pi(S)$ is a probability distribution applied to the set *S*. We denote *T(s,a,s')* the probability to get a transition from state *s* to next state *s'* using action *a*. *R* is a function that gives an expected reward function of current state and action of an agent or a set of agents. Markov model implies state transitions independent of previous environment states and actions. MDP ensure convergence of the learning algorithm.

The trade-off between exploration (determining the scope for action in time: memory for past, untested actions for the future),i.e. $a_t \neq a_t^*$, and exploitation (choice of action, interaction with the environment and reward obtaining) can't be neglected, i.e. $a_t = a_t^*$. Exploration and exploitation are dependent and must be executed alternately. Another important point of the reinforcement learning is its ability to treat the problem as a whole even and especially in an uncertain environment.

Value Function and Optimality

The value function V^π is the sum of reward values r obtained by iteration during time following a policy π. It is a mapping function between states and values. The objective is to obtain the maximum value in each state. In a considered infinite horizon, we may use the discount rate $\gamma \in [0;1]$ which can be also called the *forgetting factor*. In this case, MDP will be a 5-tuple <S,A,T,R, γ>. t is the step or time, and k is the number of previous rewards. E is the expectation operator.

$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \ldots$

$= r_{t+1} + \gamma (r_{t+2} + \gamma r_{t+3} + \gamma 2 r_{t+4} + \ldots$

$= r_{t+1} + \gamma R_{t+1}$

Bellman's equation gives the expression of a value function in terms of possible successors s' to the current state value s (Bellman, 1957). In the probabilistic case, we use transition probabilities and in the stochastic case, we use estimated values.

$$V^\pi(s) = E_\pi\{R_t \mid s_t = s\} = E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\} = E_\pi\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s\}$$

And without the expectation error:

$$V^\pi(s) = \sum_a^\infty \pi(s,a) \sum_{s'} T_{ss'}^a \left[R_{ss'}^a + \gamma V^\pi(s') \right] \qquad \textit{(Formula 20)}$$

Kopecky (2006) proposes a study of the value function iteration The Bellman's optimality equation is the maximum reward over chosen actions:

$$V^*(s) = \max_{a \in A} E\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\} \qquad \textit{(Formula 21)}$$

which is equivalent to $V^*(s) = \max_{a \in A} \sum T_{ss'}^a [R_{ss'}^a + \gamma V^*(s')]$ where $P_{ss'}^a = 1$ in equiprobability of transition. The discount factor γ can also be set to 1 depending on the value function. The goal of reinforcement learner is to learn a good estimate of $V(s)$ from its own experience and use this estimate to choose the best policy to achieve a task or a scenario on a time period. The optimal value function is $V^*(s) = \max_\pi (V^\pi(s))$ is the maximum possible value of $V^\pi(s)$, where π is a policy allowed to change. We determine an optimal policy $\pi^*(s) = argmax_a [R(s,a) + V^*(T(s,a))]$ resulting from a maximum value function of V^* compared to all other policies π with values V^π. V^* is the unique solution of a system of non-linear equations.

These important formulae allow executing the useful Value and Policy Iteration algorithms of Sutton and Barto.

Introduction to Symbolic Dynamic Programming and Bounded rationality

Symbolic dynamic programming and symbolic reinforcement learning solve two main issues: curse of dimensionality by reducing the problem space at different levels of abstraction and symbolic learning which is very lightly reached in the scientific literature because of its complexity and of the difficulty of expressing the conceptual and contextual relations. Symbolic dynamic programming is a new research paradigm designed to find optimal policy in symbolic representation of information, like frames, events or predicates, of first or higher order logic. However, some attempts by Reiter (1991) and others, like Sanner and Boutilier (2009), Sanner and Kersting (2010), have been studied to mix reinforcement learning with first order logic in so-called fluent.

Because of the large number of possible transitions and resources constraints, some researchers work on bounded rationality (Simon, 1982; Horvitz 1988; Cherniak 1990; Russel and Wefald, 1991; Russell and Subramanian, 1995: Bratman et al., 1998; Doyle, 1998; Nobre et al., 2009) to reduce the logical operations and memory space required to solve a problem. We also can bound the rationality of our agents by using contextual roles of our frame and then limiting choices of action and by allocating few tasks or actions by agents in a distributed architecture.

Semantic Reinforcement Learning will permit to affect a value to events. It can be used to reduce or sort produced events by fusion agent following a contextual policy or to take decision by fission agents to act in the environment. Bounded rationality is equivalent for us to the storage of events of abstraction levels in the concepts and models ontologies. *Multilevel Learning* is learning at different layers of agents and levels of abstraction. It is also be use as a problem solver.

Semantic Reinforcement Learning

In our case, our leaning boxes want to correct the set of events produced at a time t by an agent, i.e. actions of agents in MDP. In our agent, a state s is a specific configuration of the agent's memory. The evolution of the configuration is deeply related to the incoming facts p stored in memory and integrated in s, the state becoming s'. The production of new events (actions a) directly depends on the rule models, the new facts in s' and the past facts in s related to the event models of the new facts in s'. They are read by the inference engine. These facts must have their event model in the precondition of the rule models. The inference engine of agents is the common transition process to make an agent change of state. As in dynamic programming, agents are MDP. In our semantic agents, the possible states correspond to facts of event models, and possible actions correspond to rule models which characterize the agent's behaviour. RL can rank all agents in a global view for a given objective and related to final actions done in the environment, or can be applied on agents individually. We will see the global learning in the next section. An agent obtains an individual optimal reward if it maximizes its rewards over time by acting in accordance with the environment. It means that the agent has the good rule models but not the good parameters of these rule models, or that the agent does not have the good rule. In the second case, the learning box should make next agents create new rules. If it is not the role of this agent to produce this type of events, then agent will ignore the modification. In the first case, RL can be applied with the following algorithms.

$V_p(s) \leftarrow V_p(s)+r$ where r=-.1 if action failed

$V_p(s) \leftarrow V_p(s)+r$ where r=+.1 if action succeed

Algorithm 6 – Semantic RL with state

Algorithm 6 increases the rank of the right applied rule models p. V_p is kept in memory of learning boxes for all states s. So, in state s, action a, i.e produced event from rule models p, of the agent will be reinforced with time, and this works in a parallel manner.

$V_p \leftarrow V_p+r$ where r=-.1 if action failed and $V_p>0$

$V_p \leftarrow V_p+r$ where r=+.1 if action succeed and $Vp<1$

Algorithm 7 – Semantic RL state-independent

As rule models are active only for their specific roles and arguments of the facts in s' are present, we may remove s from the algorithm 6 and only change the value of confidence in the database (Algorithm 7). The Value Iteration algorithm can solve the problem over execution or simulation time. No need to keep State-Action values in memory. Space complexity $O(SA)$ become $O(A)$ where S is all possible states of the memory configurations and A all rule models in memory. In addition, r can be chosen more accurately based on the total number of rule models or the number rules using the same information in s' to produce other events.

$V_p \leftarrow V_p(x_i)+r$ where r=-.1 if action failed and $V_p>0$

$V_p \leftarrow V_p(x_i)+r$ where r=+.1 if action succeed and $Vp<1$

Algorithm 8 – Contextual Semantic RL

Algorithm 8 is a Semantic RL version taking into account some possible contexts x_i readable in the situational context X for fission or in inputs P^+ for fusion. Now, space complexity is $O(XA)$ where X is all possible contexts and A all rule models in memory. This is very interesting for two reasons: the first is contexts are not so many, and the second is: compared to $O(SA)$, $O(XA)$ is a good compromise to act differently based on context.

Multilevel Semantic Learning

In this section, we are more interested by the synergy of agents through layers than individuals and this by the evaluation of the semantic meaning extraction, in other words, by the quality of the fusion and fission processes.

To find the optimal policy or optimal reward values, we need to affect a value to each agent of the grid cell. The environment gives a reward for the actions made by the whole system. Semantic learning should be applied to agent components.

Be the Formula 13, $P'=\sum_l \sum_i Inference_{l,i}(P)$ where i is the agent, l the layer of the agent and L *is* the number of layers of a grid. Under the assumption of fusion processes and fission process, i.e. two grids of agents, the abstraction level of the meaning produced by an agent depends of the layer l. And l is therefore the abstraction level increasing for fusion agent layers and decreasing for fission agent layers (Figure 3.19 and Figure 4.39).

The grid of fusion agents or fission agents receives a reward *r* given by the environment according to the quality of the response (events of meaning and orders) which is measured and compared to an objective of meaning. In the *Semantic Learning* section, we have seen that *X* is the meaning generated by fusion process and *P'* the meaning or orders generated by fission process. Thus learning box of a grid (fusion or fission agents) has to maximize the learning values *V* corresponding to outputs of agents at different levels *l* in the grid in order to reach optimal value and policy using the formula 22.

$$V^*(s) = \sum_l \sum_i \max_{a \in A_{l,i}} E\left\{\frac{r_{t+1}}{L} + \gamma V^*_{l,i}(s_{t+1}) \middle| s_t = s, a_t = a\right\} \qquad \textit{(Formula 22)}$$

The value $V^*(s)$ is distributed over the grid of agents, each agent of a layer obtains a reward $\frac{r}{L}$ and its affected value is:

$$V^*_{l,i}(s) = \max_{a \in A_{l,i}} E\left\{\frac{r_{t+1}}{L} + \gamma V^*_{l,i}(s_{t+1}) \middle| s_t = s, a_t = a\right\} \qquad \textit{(Formula 23)}$$

Note that the s_{t+1} is the next state after the meaning or action is started or finished and the effect is rewarded by the environment, $t+1$ indicates the next step in time, this is the reasoning time plus the perception time. Note too that no learning solution can be found in the case where two meanings are conflicting, so we assume there is no contradictory expected meaning. If it is the case, learning will tend to equilibrium between these meanings. This is an interesting property for game theory (Nash 1950).

Moreover, the evaluation is independent of the architecture, if a UDDI server may send the quality of service (QOS) or quality attributes of a service (OWL-S) to our learning box, learning box can be able to use these quality values to optimize the architecture following the workflows and mechanisms introduced in Section 4.3. Learning box can also measures performance of learning with a mean-squared error over a probability distribution *T* such as $MSE = \sum_{s \in S} T(s)[V^{\pi}(s) - V_t(s)]^2$ following a known policy π or $MSE = \sum_{s \in S} T(s)[V^*(s) - V_t(s)]^2$ following an optimal policy π^*.

Problem Solving

The previous multilevel RL algorithm gives us an enormous advantage for problem solving (Breese and Fehling, 1990). Knowing that we have the values related to different levels of abstraction, it is now possible to divide the problem in sub problems.

1. Find a solution at the higher level of abstraction $l=L$
2. Solution evaluation (states, actions) using RL values
3. If solution found at level *l*, return the solution
4. If $l=1$ (lower level of primitives), return "no solution, need more information"
5. If no solution at level *l*, decompose in states/actions at level $l-1$
6. If solution found, return to level $l+1$ else goto step 2

Algorithm 9 – Problem Solving

Beginning at the higher level of abstraction *L*, solving the problem is looking for a direct solution at the current level or finding an optimal path between high

level meanings to primitive percepts for the fusion agents and high level meanings to primitive actions for fission agents. To find the path, the algorithm checks the values $V_{l,i}^{*}(s)$of possible actions a of the agent i of layer l(Step 2).

4.5 Conclusion

In this chapter, we explained how to build complex architecture in interaction with many services, modalities and systems in the environment. We highlighted the possibilities of our architecture to be suitable for ambient intelligence and pervasive context in the search of considering expected requirements, and more specifically to manage multiple levels of abstraction of knowledge in different layers of fusion and fission agents.

We explained the mechanisms that show how our architecture can solve interaction issues and how agents are the pieces dedicated to fusion and fission processes. These mechanisms are discovery of new services or agents; composition of services and agents; extensibility and adaptation of the architecture; modality selection; audio, video and dialog interaction using EKRL events; perception, action and multi-contexts awareness management with fuzzy logic. We also presented our semantic learning which is symbolic and meaning learning in our semantic agent but in agency too. We applied this semantic learning to the fusion and fission processes. We then present how reinforcement learning and dynamic programming value function is applied to the different levels of abstraction of the knowledge and behaviours of entities in the environment. We propose a problem solving algorithm based on the previously presented value function. These mechanisms are very important to take account of all contextual variations and reinforce interaction in multimodal and pervasive contexts.

CHAPTER 5

IMPLEMENTATION AND ANALYSIS

« To think is easy. To act is hard. But the hardest thing in the world is to act in accordance with your thinking » Johann Wolfgang von Goethe

5.1 Introduction

Last chapters defined the state of the art, requirements, definitions, modelling of components of the architecture, modelling of interaction architecture and different interaction mechanisms. Semantic agents, Services and Editor of agents' memory have also been modelled.

In this chapter, we fully present the implementation of our Semantic Agents and services for Multimodal Interaction (SAMI) framework. It includes the EKRL concentrator, an important piece of the architecture to interconnect services available in several locations of an ambient intelligence system, some PC at home and Internet web services. The aim of this concentrator is to allow external systems communicating with our architecture and become part of our architecture through a network. We analyse the performance of the network, the knowledge base access, the inference engine time, consistency and robustness of the events management. Complexity of the information storage and retrieval is also analysed. To validate that inference engine of agents well manage the temporal aspects when matching sequences of facts with rule models into the knowledge base, we developed two formal coloured stochastic Petri nets, one specific to the "PutThatHere" scenario and one for the general case of any possible scenarios.

We finish this evaluation with the presentation of a comparative study of different approaches made with existing platforms, semantic architecture or rational agents, able to solve part of all the human environment interaction.

5.2 SAMI Framework

5.2.1 Framework Definition

From the modelling of the components of our architecture for multimodal interaction, we developed a framework for designers and developers in order to:

- Develop and plug virtual, network and physical services through a web interface embedded in a EKRL concentrator
- Design semantic agents working on a personal computer and edit their semantic memory
- Test our agents in several situations
- Check mechanisms previously introduced in this thesis
- Measure performances

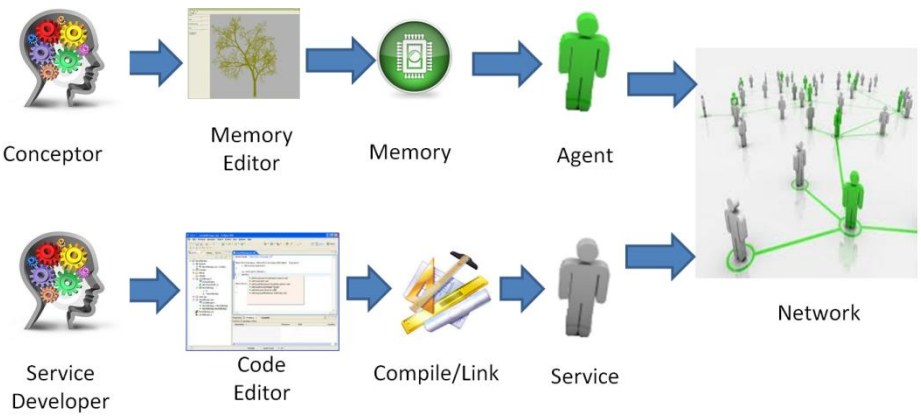


Figure 5.1 – Agent & Service Development

Agent and Service must be developed and integrated in a network to work together (Figure 5.1). Services are often implemented by third party. So designer and "knowledge" conceptor may focus on communication and interaction between agents and services using EKRL messages, and on the conception of agents. Communication between agents is automatic. Communication between agents and services is realized by the EKRL concentrator which requires a setting. We draw the conceptor's attention to the fact that no code writing is necessary for agents, only directly modifying concepts and models in the memory of agents. The system is not pre-programmed with a sequence of executable commands.

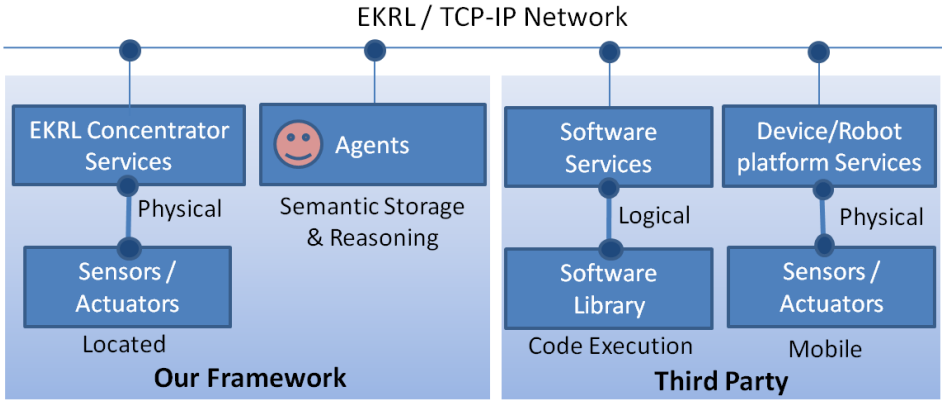


Figure 5.2 – Ambient Intelligence Environment

Figure 5.2 shows the parts of the framework interconnected on a TCP-IP networking protocol. Messages (requests and responses) are directly exchanged in EKRL. The EKRL concentrator is an embedded Linux system that allows setting and running services connected to this concentrator. One key point is that the EKRL concentrator is a wrapper system which translates services data into EKRL messages, and vice versa. The concentrator's objective is to make services communicate with agents in EKRL. These services can be virtual, network or physical. Virtual services are pieces of code to simulate hardware services which can be sensors or actuators. Virtual services are very useful to test the agent behaviours receiving the EKRL messages produced by them, and in general test the behaviour of the whole system. Network services are driving services accessible via TCP-IP network, like, for example, MS Robotics webservices controlling hardware or simulated hardware or any web services on Internet like unit converters, money converters, shop opening hours, and so on. Finally, hardware services are driving hardware parts directly connected to the board. On the figure, Services of physical device or robot platforms can also be driven by EKRL concentrator or directly communicate using EKRL messages. Other services are fully software executing required operations, they are distributed code libraries in the network, they can also be driven by EKRL concentrator or natively communicate using EKRL messages. The last components are our semantic agents. They are storing information, communicating and reasoning purely in EKRL.

Figure 5.3 presents the different blocks of our framework. Blue blocks have been developed for this framework. The hardware layer is in black; it comprises computers, RoBoard, Nao robot and parts, sensors and actuators. The common operating systems layer is in purple; they are MS Windows, Linux and Apple MacOS. Orange layers contain system and development tools like .NET and

JADE frameworks, Visual Studio, Robotics Studio; DSS/CCR and Nao API. Programming languages are PHP5, HTML, Python, C#, C++, VB.NET and SQL, Java. Green boxes are specific connector, library or webservices from tiers, like MySQL connector for .Net applications, FIPA library to manage physical agents and Robotics web services provided to drive robot parts. EKRL messages have been well defined in Chapter 3.

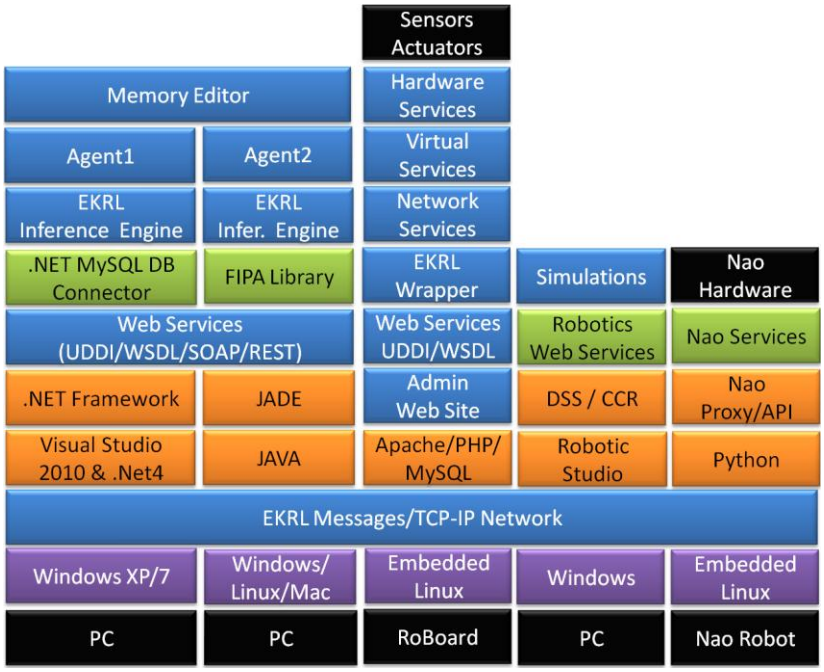
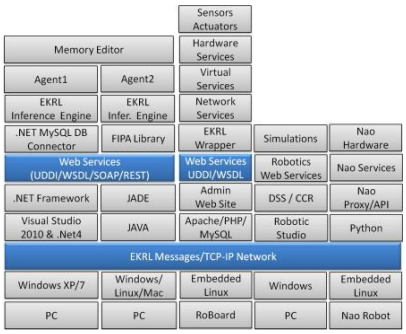


Figure 5.3 –Framework

5.2.2 Communication



EKRL messages are transferred over TCP-IP. They may be encapsulated in one of these standard protocols: SOAP, HTTP/REST or HTTPS/REST, and possibly in mails SMTP/POP. These protocols work in local area network and in wide area network on Internet. Web services can be parts of Internet web server, local web server and applications like Robotics Studio where robotics services are accessible through Decentralized Software Services (DSS) protocol. Interoperability of web services is an important and powerful quality and requirement to ensure openness and distributed computing. Existing Universal Description, Discovery, and Integration (UDDI) and OWL-S servers are used for the discovery of web services.

Communication module of web services (agents or services) manages 3 main operations: registration and publishing of its own description as a service provider to an UDDI service (Service registry) and, reception and dispatch of EKRL messages. The *Find* operation (Figure 2.9) is managed by UDDI services and *Bind* operation is virtually realized by fission agents.

Figure 5.4 (part A) presents the CPNTools schema of processing EKRL messages from arrival (part B) to processing in a web service (part C). EKRL

messages are classified by emergency and sent date, and then stored in a first in first out (FIFO) queue (Msgs). Execution of service's code or agent's inference engine will remove them successively from queue over working time. Agent will consider incoming messages as EKRL queries. Service will consider incoming messages as orders to execute now or at a scheduled time. In this latter case, service will keep the message in a jobs queue for a future execution (planned).

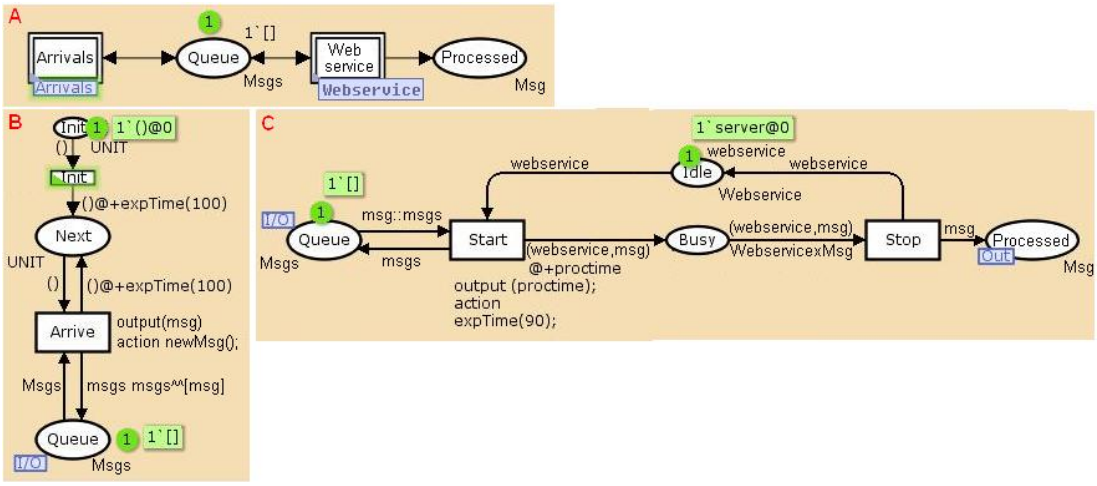
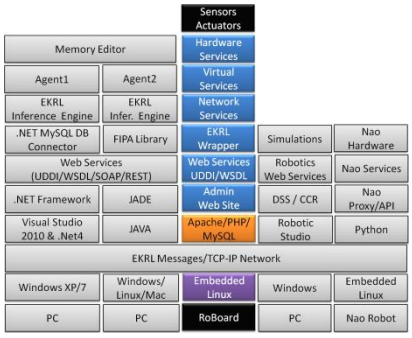


Figure 5.4 –EKRL Message Reception Queue in CPNTools

In contrast, there is no queue to send messages, they are directly sent to other components of the network - in a broadcast way (all recipients will receive and check if they are concerned by the predicate and root predicate of in the message) or - in a more specific way with the agent or service IP address (for example, 192.168.0.1), or if several components network address (for example, 192.168.0.0) is provided.

5.2.3 EKRL Concentrator



As things stand there is no web service, sensor, or actuator able to communicate in EKRL with our agents. We developed an EKRL concentrator in order to manage communication between services and agents. EKRL concentrator is an inputs/outputs platform to develop and connect different devices of a room or a house to agents of a TCP-IP network. Services can also be embedded in a robot for example. They communicate via the network (most often wireless). The concentrator can manage virtual services that are programs running on it and sending EKRL events at a specific time period emulating hardware devices. The concentrator can also manage networking services converting their information to EKRL messages. The concentrator plays the role of wrapper and can receive EKRL messages from agents and drive material devices using specific drivers. We will call the EKRL hub of events EKRL: "EKRL concentrator".

The concentrator (blue rectangle) comprises different modules (red figures): drivers of services with their configuration (virtual services are internal, physical are connected by wire to the RoBoard and network services are

connected to the TCP-IP network), a scheduler to run the services and drivers, a network reception to queue messages, a network emission and the website administrated by a manager or a developer (Figure 5.5).

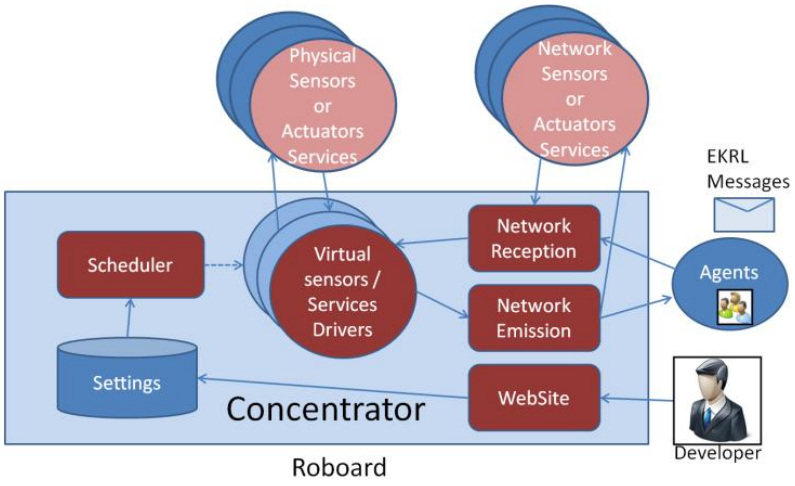


Figure 5.5 – EKRL Concentrator

Platform

EKRL concentrator runs under Debian Linux Lenny (release 5.0.2) in a RoBoard RB-110 card (Figure 5.6) containing a Vortex86DX processor (32bits, 1GHz) with a 256 MB RAM and 2GB Micro-SD memory card. The board works with a 9V battery spending little energy. We have also developed a Web graphical user interface (GUI) to manage the box and services. This Web site runs with Apache 2, PHP5, JQuery and a MySQL5 database to store settings (blue). All EKRL messages are sent on TCP-IP messages built from templates to be filled with drivers' data.

Figure 5.6 –EKRL Concentrator

For our application it is necessary to install the RoBoard an operating system, and there is a choice between Windows XP EC (embedded) or Linux. We chose to install Debian Linux 5.0.2 Lenny because Debian Lenny 5.0.2 is freeware and the RoBoard website provided the Linux drivers for the network adapter. Installing Debian 5.0.2 on the map can be done with several methods: Installation with VirtualBox or installing Linux from scratch (with a bootable USB drive). The RoBoard website also provides ISO images that can be directly put on a USB drive. During installation, we carry out several manual configuration steps:

1. We choose English for the installation because the image does not include maintenance of another language.
2. Set Network configuration (IP address, subnet mask, DNS ...).
3. Set the date and time is very essential for the application but we will configure the system once installed.
4. Create and configure two partitions (ext3 and swap).
5. Install the software base, and check that the NIC was connected to the Internet; the installation of missing software will be downloaded automatically.
6 Set the root password.
7 Install SSH to be able to connect to the system remotely and work on it.
6. Install and configure Iptables (firewall) to secure the system.
7. Install Apache HTTP Server to host the website and webservices.
8. Install PHP5 to develop the website and PHP_SOAP module is required to manage webservices.
9. Install Mysql to create the database for the website and make it dynamical.
10. Install the phpmyadmin interface to access the Mysql database.

Once finished installing Linux, we made a copy of the memory card containing the configured operating system for future use.

Types of Service

EKRL concentrator can manage three kinds of service:

- *Virtual Services*. These services simulate behaviours and send EKRL messages to agents. They let us check our agents' performance without actual service information. They help check agents' responses. Several tests are made: identifying bad facts (consistency checking), maximum EKRL messages accepted by agents, and execution time.

- *Hardware Services*. These services directly drive sensors or actuators connected to the multiple ports of our RoBoard. They get data from sensors and send EKRL messages to agents. They also receive EKRL messages from agents and send orders to actuators. For example, they might read temperature and light sensors while driving servo motors.

- *Network Services*. These services manage software or hardware drivers connected to the same TCP/IP network. For example, MS Kinect gesture recognition hardware can be connected to the USB port of another PC on the network. This PC runs a virtual reality peripheral network (VRPN) that sends human gestures or postures to our concentrator. Another example is a Web service able to perform complex mathematics operations, convert currency, or convert units. Values and units to be calculated or converted are sent to the Web service. The latter returns one or more resulting values along with their corresponding units to the driver on the concentrator. The driver inserts the data into an EKRL template message and sends it to the agents.

Website Interface

The administration website allows managing EKRL box settings; virtual sensors creation, modification and deletion; hardware sensors creation, modification and

deletion; network devices creation, modification and deletion; actuators sensors creation, modification and deletion; monitoring of the activities (Figure 5.7).

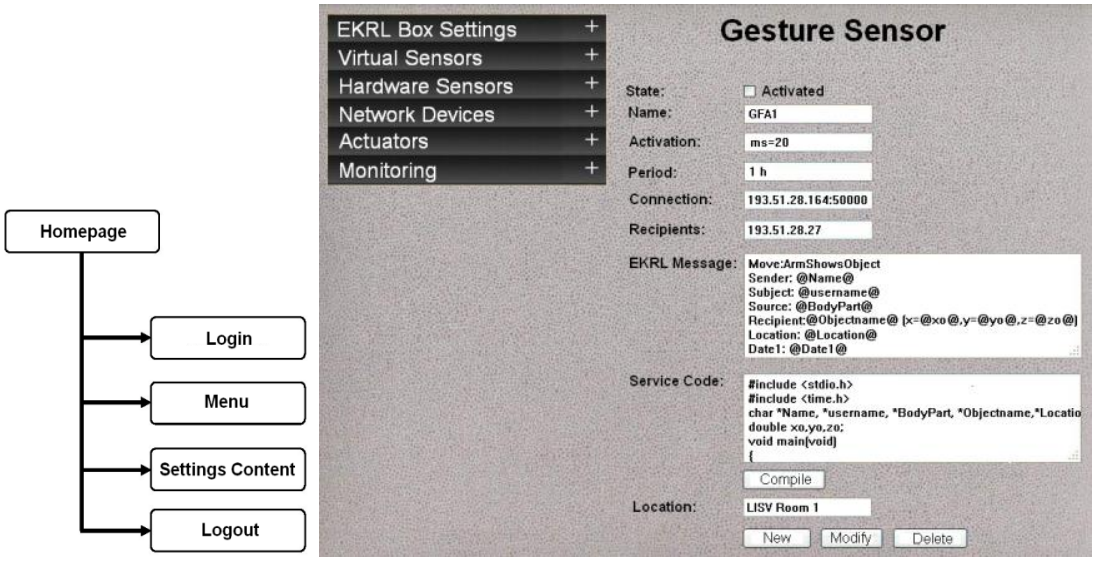


Figure 5.7 – Web Site and graphical interface

Users can surf the website with any web navigators. The homepage proposes to log into the site, a page appear with a menu at left and corresponding contents to menu items at right. The website is secured by user and password to open a session. At any time, it is possible to logout. Once logged in, it is possible to modify settings of the box, and monitor the activity of running services and network load. Other menu items allow the modification of services and actuators information. The 3 buttons at bottom of the page are used to create, modify or delete items of the menu (i.e. services).

Each code of a service is associated to a configuration: a state (Boolean), a name (String), an activation condition (string), a trigger period, connection IP address and port (for network service), recipient IP addresses (forced agents or services, if empty, broadcast), EKRL message with variables to be filled, C code of driver and location of the service. The *Compile* button allows checking and building an executable program. Source code for the services is directly written in the textbox. All these information are stored in the MySQL database for the administrator and in configuration files for the scheduler.

Website Code

Administration website is developed in PHP5 and HTML for the pages structure and display. The menu has been implemented with JQuery javascript library. PHP Program has the behaviour presented in the following flowchart (Figure 5.8).

This structure is general and can work to build many websites with different contents. A homepage asks for user login and password. Session is managed by the apache server for the authenticated user. Menu and content is displayed on the same page. For each menu item, a dynamic content is generated. All fields can be modified and saved. A service can be activated and deactivated manually. EKRL message template can be written in order to be filled by data given by service and sent by the scheduler. Service code can be modified and compiled.

Once activated, the website generates a .conf file containing the complete configuration of the service which will be used by the scheduler to execute the compiled code.

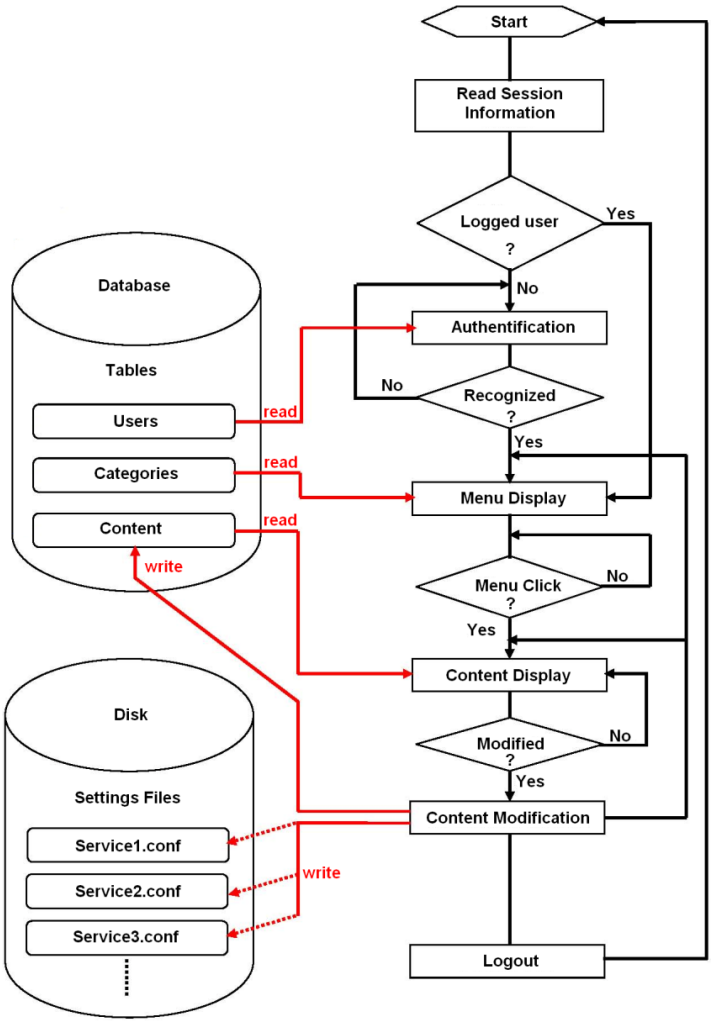


Figure 5.8 –WebSite Flowchart

Website Database

We have created a Mysql database to make the website dynamic. Figure 5.9 shows the relational database built. All information about users, menu items and content of the site are managed in the database.

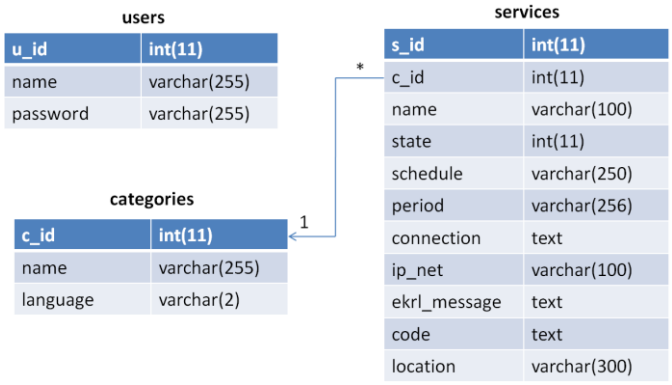


Figure 5.9 – Relational Database

We made the website secure with a login phase where the user has to enter a name and a password to access the website content. Table *users* allow logging. Table *content* is named here "*services*" to contain services' information of this website. Table *categories* contains the items of menu and Table *services* contains the sub items of menu.

Services Code

The website appears like a development environment where settings, EKRL templates and code of the service can be managed. The service's code is written in C language and is compiled with *gcc*. A window indicates if there is a compilation error. The configuration files generated by the website will be read by a scheduler that must run the service at a predetermined date and time.

For example, once the temperature sensor service is activated, it is launched by the scheduler, it gets the temperature value from the hardware sensor through the input ports of the RoBoard, it takes the EKRL model and replaces the *@@temperature@@* string by the obtained value *temperature*, and finally sends the well-formed EKRL message to agents in the communications network.

In another example, EKRL concentrator receive an EKRL from agents, "RECEPIENT" role's content is compared by *network reception* module to service names in the database to know what are the concerned services and if these services are currently activated. If it's the case, message is renamed with name of the services and the current date and saved into the *messages* directory. Each service has a piece of code to check if one or several messages are present in it's the *messages* directory. If yes, it opens it and search for the roles, arguments and values that can be used to control an actuator, most often values are stored in the CONTENT role.

We realized an example to drive a +5V servo motor connected to an output port of the RoBoard. The angle is determined by the duration of a pulse that is applied to the control wire also called pulse-width modulation (PWM). Each 20 ms, the service is executed by the scheduler. Once the EKRL message is received, the pulse changes. For example, a 1.5 ms pulse will make the motor turn to the 90 degree position (neutral position).

Scheduler

The scheduler runs at start of the Linux system. The scheduler begins to load all configuration files (Figure 5.10). It executes all activated services at the required date, time or period specified in the configuration file.

For the servo motor example previously introduced, we take the vsc.conf file from the *Vehicle Speed Control* service. The first line is the name of the concerned service. The second line is the activation value, 1 is on, 0 is off. §In the third line, we defined a specific language to read date, time and period. "Sat,Sun" indicates we will run the service only Saturday and Sunday of week. "Jul" means july. "15:00:" means any seconds at 15:00, "14:" would mean any minutes and seconds of 14 o'clock. 2011 indicates we limit the execution to year 2011. In the fourth line, ms=200 is the period in milliseconds, so each 200 ms, of the date in the third line, we will run the vsc service.

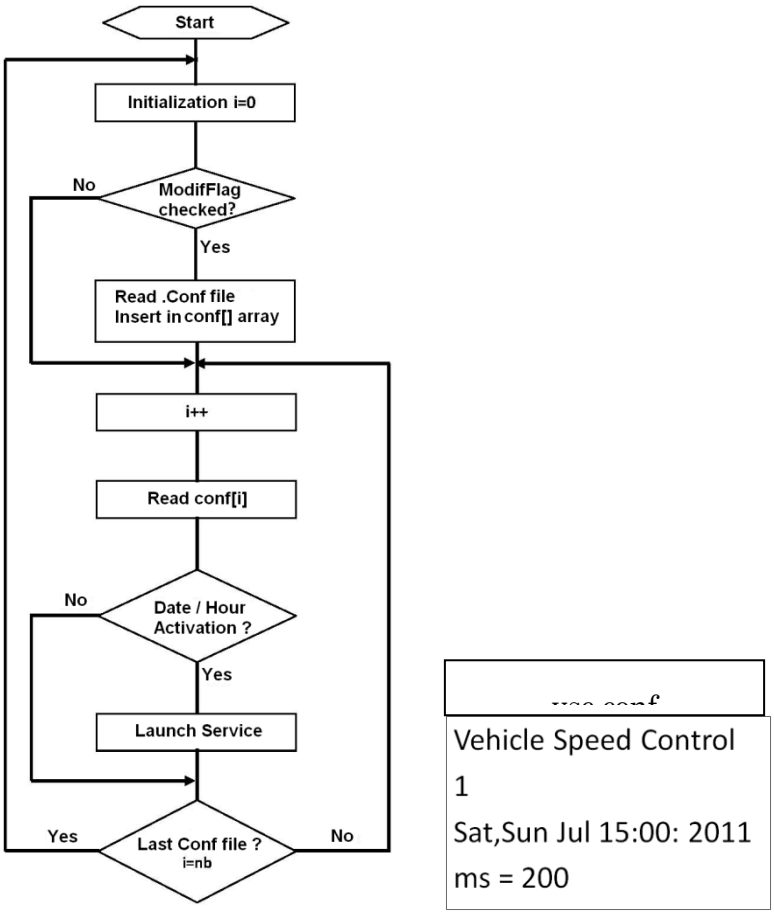
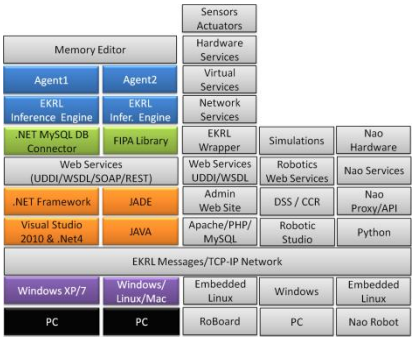


Figure 5.10 – Scheduler Flowchart & VSC conf file sample

EKRL Wrapper

EKRL Wrapper allows the EKRL interaction between services in the EKRL concentrator and the agents outside the box and in the same network. As we explained, an EKRL model is stored in the settings page of each service. This model is only used to send EKRL messages with service data to agents. Agents may also drive output service by sending EKRL messages. This message will be directly transmited by the *network reception* module to the service which must be able to parse it and use it to drive the hardware part or the start the execution of a software function.

5.2.4 Semantic Agents



Semantic agents are deeply modelled in Chapter 3. We present here the implementation of an agent and its components. Agent contains a communication module, an inference engine and a memory (its knowledge base). Communication module has already been presented in section 5.2.2 and agent communication language is simply EKRL. So we will focus on platforms, the inference engine which has been developed as a library and the agent's memory stored in a Mysql database.

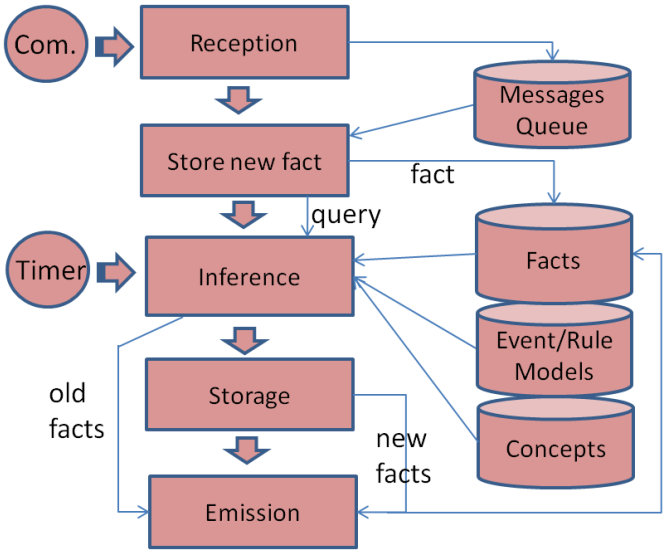


Figure 5.11 – Agent Flowchart

Figure 5.11 is the general flowchart of the agent. Agent continuously wait for a query message containing a question or a new fact, or wait for a scheduled timer event to check if previously stored events are matching current date and time, in order to produce new events. *Reception* box listens to communication module (Com.) and stores EKRL messages in queue. *Store new fact* box removes a message of the stack. If it's a fact, it stores it in the event ontology as event instance (facts). If it's a query, there is no storage. Then it calls the *inference engine* for a matching operation with the query if it's a query or with rule models in the event ontology matching the root predicate and predicate name of the query or of the fact. If matching facts exist in the knowledge base, they are directly sent to the network via the *Emission* box. Other facts are created following a matching with rule models. They are corresponding to meaning at higher level of abstraction if the agent is a fusion agent, and they are corresponding to meaning or order at lower level of abstraction if the agent is a fission agent. They will be stored in the ontology by the *Storage* box and then sent by the *Emission* box.

Third Party Platforms

Our agents are programs embedded in a single or several computers with one or more cores. In our case, we use a single computer with a 4-core Intel i7 920 processor, one core has two threads and one thread is running one agent. On Windows operating system, we developed 8 agents with VB.NET (native code) in Visual Studio 2010 connected to a MySQL 5 database using the MySQL.Net connector 6. The MySQL server can be installed on any platforms. It works on .Net 3.5 and 4 Frameworks. On other platforms, we used JADE 4, an open source platform developed by Telecom Italia Lab, with the FIPA[21] Library and agents have been developed in JAVA[22]. With JADE and Eclipse, agents read and write data directly in OWL DL file using the JADEOWLCodec[23] or the

[21] http://www.fipa.org

[22] http://jade.tilab.com/

[23] http://www8.informatik.uni-erlangen.de/en/demosdownloads.html#jadeowlcodec

AgentOWL[24] libray. Please note that JADE version is slower and limited in terms of third party libraries and multicores management. Our future experiments and performances have been measured on Windows platform.

Inference engine

The inference engine allows storing and querying memory. Algorithms of the Section 3.7.3 have been implemented as a C# class. Its code is very fast and lightweight. The inference engine has several roles: All EKRL requests are transformed in SQL queries. It works only with reference terms (nodes) of the ontology stored in database, thus reasoning is independent of human languages like English and French. Note that EKRL messages containing facts or queries are in text format. Role Concepts types are compared such as texts, numeric values, dates and concept indexes during the matching process.

InferenceEngine Class contains the following public functions:

- *ParseRole(string)* to check and parse the role content in a line of a frame.
- *ExplicitParseRole(string)* to parse the role content in a line of a frame without checking the role format.
- *CheckRole(string)* to check the role format.
- *StoreEvent(EventMessage)* to store a fact in the model ontology.
- *Query(QueryMessage)* to store a fact and call matching function.
- *Query(QueryId)* to call matching function with a known event model.

InferenceEngine Class contains the following private functions:

- *GetModelId(reference,rootmodel)* to find the index of a model.
- *GetRoleId(reference)* to find the index of a role.
- *CleanMessage()* to check and correct the format of a fact.
- *StoreRA()* to store the fact in the Role-Argument table.
- *SetMemoryName(name)* to select the memory of an agent.
- *Matching(queryid,textoutput)* to match a query with previously stored facts.
- *MatchSubTree(parent)* to match a subtree of the model ontology when parent model (ontology node) is known.
- *MatchEvent(FactId)* to compare two events.
- *ApplyModifier(ModifierId)* to apply a (modal logic) modifier to a matching result.
- *MatchConcept(ConceptId1,ConceptId2)* to compare two concepts of the same type.

Memory

Memories of agents are independent components. To interconnect the inference engine of an agent and a memory in the database engine, we developed a library called BDConnection which uses .NET Mysql Connector. MySQL can store many memories. So to connect, disconnect, query or store a memory, an agent may call the following functions:

- *ConnectDB(MemoryName)* to connect an existing memory (i.e. database)
- *ExistDB(MemoryName)* to check if a memory exists

[24] http://agentowl.sourceforge.net/

- *ListDB()* to list all memories in the database engine
- *ResetDB(MemoryName)* to reinitialize a memory to an empty content
- *DisconnectDB()* to disconnect the memory
- *QueryDBToCombobox(Sql,Combo)* to query the memory and store the results in a combox
- *QueryDBToListbox(Sql,List)* to query the memory and store the results in a listbox
- *QueryDBToString(Sql)* to query the memory and store the results in a string
- *InsertDB(Sql)* to insert data in the memory

To build a memory, we prepared 2 SQL files: - a minimal version of 57 tree nodes containing the structure of the database and the meta concepts; - a basic version of 752 tree nodes containing the minimal version plus many concepts and models. The first version is the most suitable for developers to only integrate the concepts and models required by the desired application. Each node has a unique reference. We choose to put the reference in English. All node labels are for now in English and in French but any languages can be inserted. We have also developed and tested a version with all dictionary words and definitions imported in more than 50000 tree nodes but this version is not optimized to develop multilevel structure because it contains all possible concepts at all levels. It takes about 30 seconds to load on a Pentium double core. Note that once the data are loaded in the computer's memory, agent still stays fast for querying and storing data.

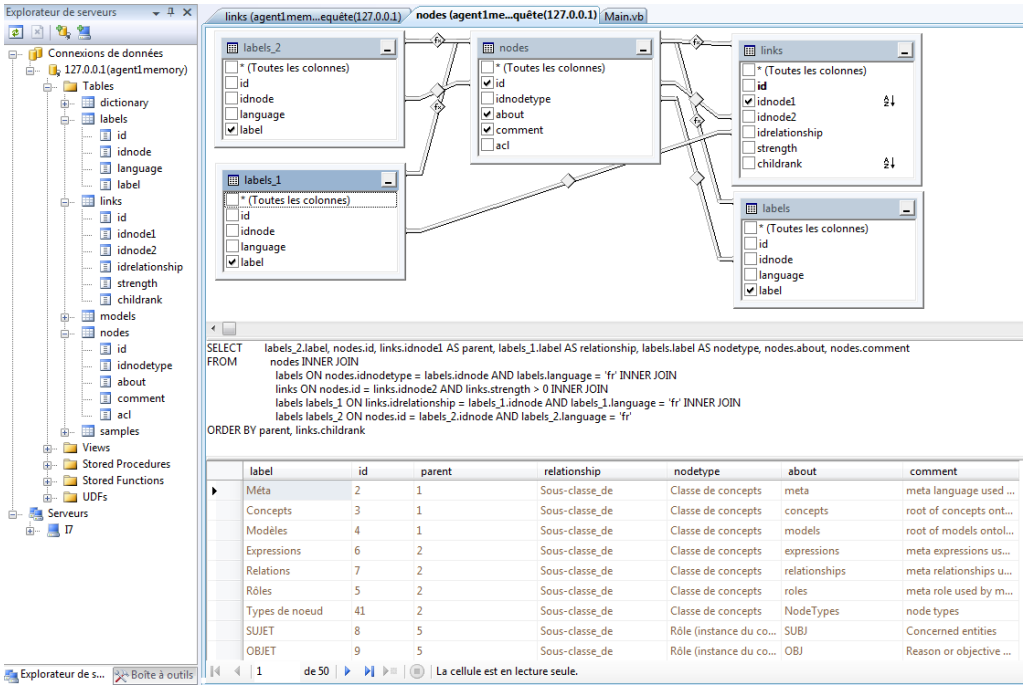
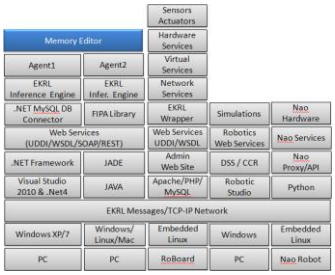


Figure 5.12 – Querying agent's memory

Figure 5.12 presents a screenshot of a SQL query to get all nodes in French in a n array. For a better ergonomic use, we develop the agent's memory editor. All concepts and models of the memory are modified in the memory editor. All concepts can be imported and exported from Protégé or other OWL editors like SWOOP.

5.2.5 Memory Editor



The memory editor can load a memory, allow modification of ontologies (nodes, labels, relationships, concepts and models) and is able to import and export data in OWL. Description is in Section 3.7.4 . The interface is Multilanguage and you may choose the language to edit the labels. The inference engine queries and stores facts independently of the memory editor. And, in general, sub components of agents and memory editor are independent (Figure 5.13). It means memories can be shared by the inference engine of an agent (or several agents) and in same time by Memory editors. Memories are permutable between agents. They also can be cloned or duplicated. It is an offered option to replace temporarily or indefinitely the memory of an agent by another more appropriate to a situation where a specific knowledge and know-how is required. SQL connector and *DBConnection* class allow changes of IP address and SQL database name.

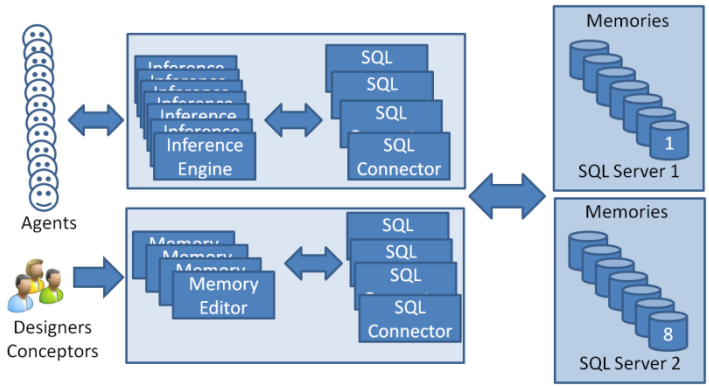


Figure 5.13 – Sub components interconnection

Platform

Memory editor is working on Windows operating systems. It is developed with VB.NET in Visual Studio 2010 connected to the MySQL 5 database using the MySQL.Net connector 6. It works on .Net 3.5 and 4 Frameworks. Concepts in OWL files can be developed in standard editors like Protégé or SWOOP but our editor is still required for models of events. These java editors work on any platforms.

Interface

The interface of the agent's memory editor (Figure 5.14) contains:

- A title bar with the name of the current opened memory
- A menu bar to edit a memory (Memory, Node, Edit, View menu items), change language (Tools menu item) and an help (? menu item)
- A tree structure at left that displays the Memory ontology which contains the Meta ontology (Modifiers, Event Types, Relationship Types, Role names, Node types of all ontologies and itself, Properties), Ontology of concepts (concept classes and instances) and the Ontology of models (rule models, event models and facts).

- A frame (at right up) that displays the OWL information of the selected node in the tree. An OWL node contains a label, a node index, a node type (meta concept, concept class, concept instance, model class, model instance), an OWL relationship type (for example, *SubClassOf*), an about tag (i.e. reference) and a comment tag.
- A frame (at right bottom) that display all information of the node to be modified in the current language (node information, model information if the node is a model, properties if the node is a property, instances to view the list of all instances associated to a class, a set of concepts which is a list of independent concept nodes, dictionary definitions, dictionary samples or media URL)
- A status bar to display loaded nodes for example.

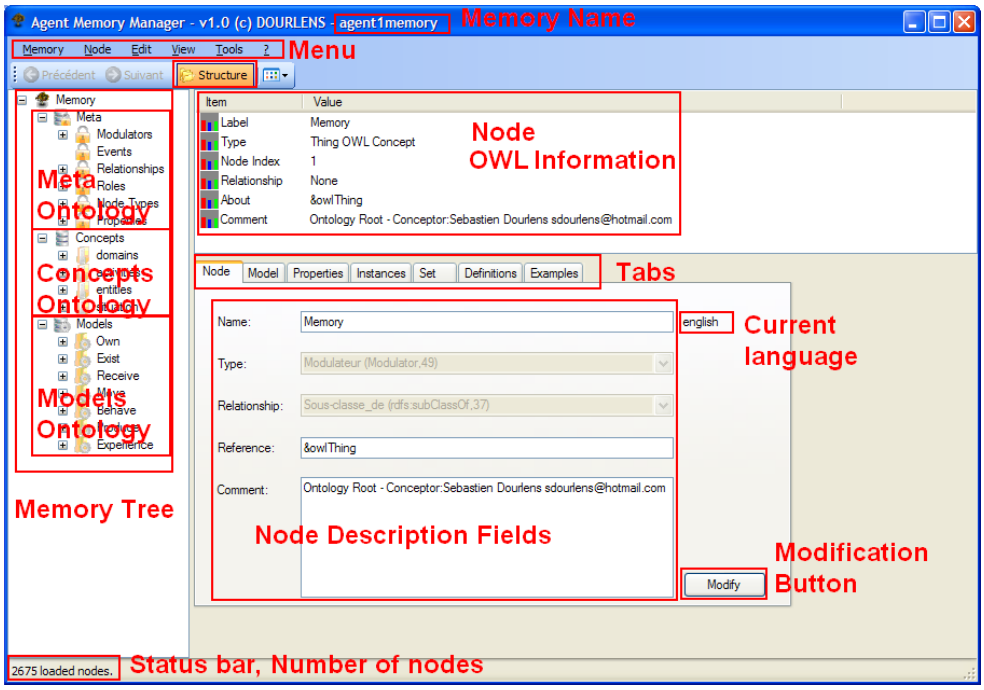


Figure 5.14 – Editor of Agent's Memory

Once started, the editor load the default data of the memory (named Agent1Memory) in a tree view with the *LoadTree*() function. The tree structure is based on the *ra* sql table. The nodes of the tree view have labels in the selected language in the *Tools* menu (Figure 5.15).

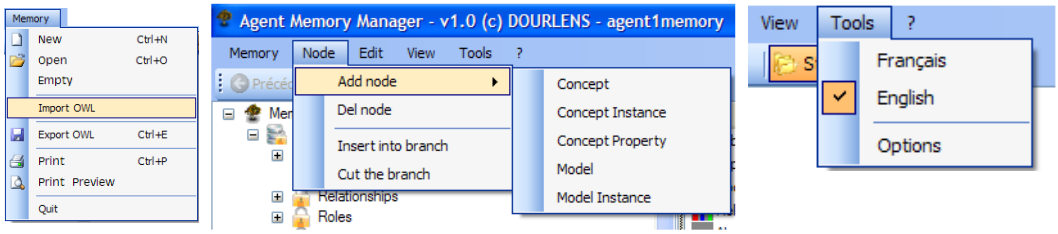


Figure 5.15 –Memory Editor Menu

OWL Import/Export

Concepts of OWL Files can be imported and exported. For these operations, we develop the function *OWL2DBImport(file)* and *DB2OWLExport(file)* to realize

this operation once the chosen agent's memory is loaded. It is accessible in the *Memory* menu (Figure 5.15).

Ontologies Edition

Memory Editor allow designer to modify the nodes of the ontologies. It is possible to add or delete a node (of types Concept, Concept Instance, Concept Property, Model, and Model Instance), cut a branch of the tree and insert a node into a branch (menu in Figure 5.15). To modify node information, designer has to select a node by a click in the tree view and uses the (left bottom) frame and tabs to change information. The "Node" tab contains the general OWL information about the node (Figures 5.16 and 5.17) and The "Model" or "Fact" tab allows to write the EKRL model and facts by filling several roles with formulae on arguments, it is the case of event model and facts modification, and other models, it is the case of rule models modification. For instance, Figures 5.17 and 5.18 show the information of the "Nao1 robot is walking at home" fact.

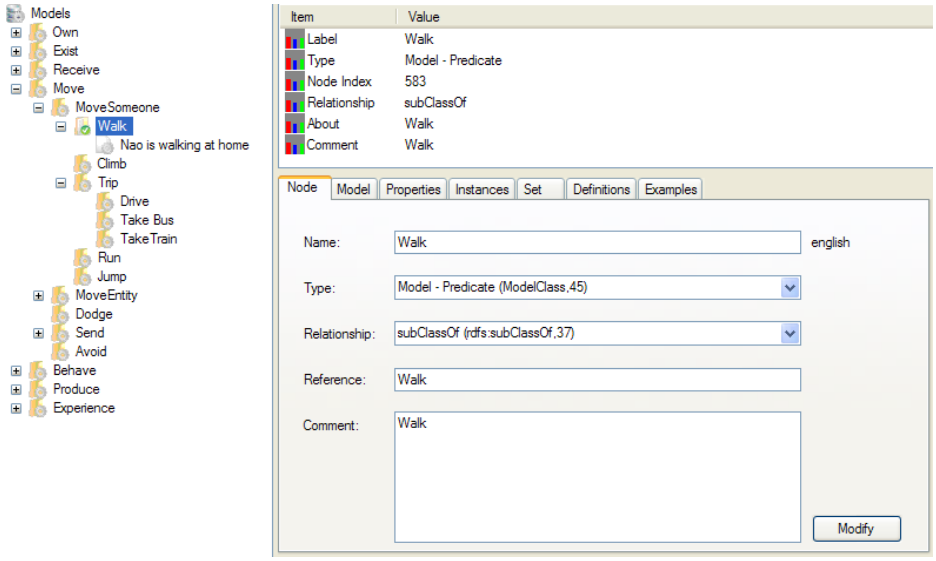


Figure 5.16 – Model Node Edition

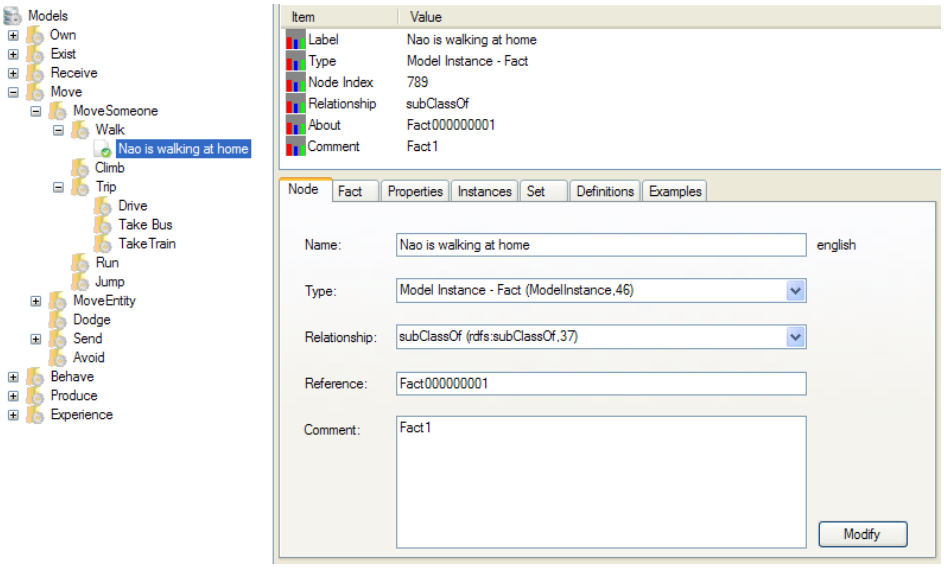


Figure 5.17 – Fact Node Edition

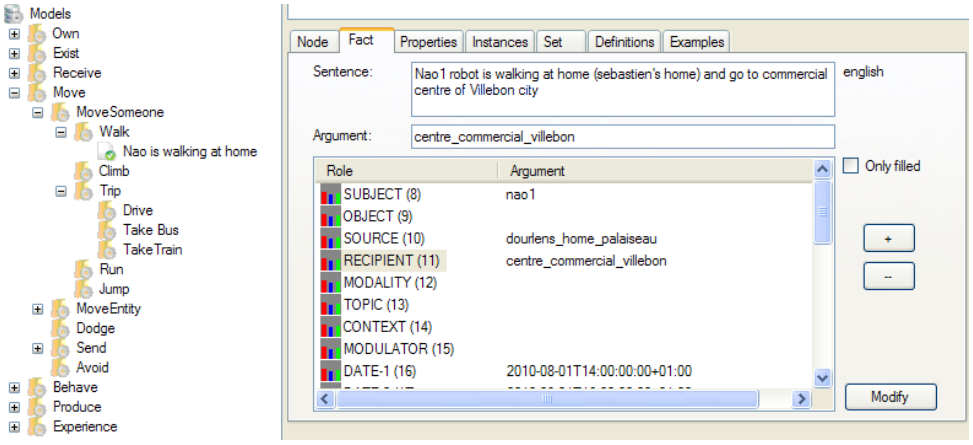


Figure 5.18 – Fact Roles Edition

5.2.6 Robotics Platforms

Robotics Studio Simulation platform

We introduced the Robotics Developer Studio simulation platform at the end of the chapter2. The most important aspect of this platform is the concurrency (CCR) and management of robotics web services (DSS). Concurrency and Coordination Runtime helps make it easier to handle asynchronous input and output by eliminating the conventional complexities of manual threading, locks, and semaphores. Lightweight state-oriented Decentralized Software Services framework enables to create program modules that can interoperate on a robot and connected PCs by using a relatively simple, open protocol. Once a simulation or any hardware controllers are launched, it is possible to access these services using a network connection. They are *network services* of our platform and can be managed by EKRL concentrator. Visual Simulation Environment (VSE) provides the ability to simulate and test robotic applications using a 3D physics-based simulation tool. Visual simulation environment (VSE) allows developers to create robotics applications without the hardware. Sample simulation models and environments allow testing our application in a variety of 3D virtual environments. This platform provides a large choice of real and virtual environments, actuators, sensors and featured robots (Figure 5.19).

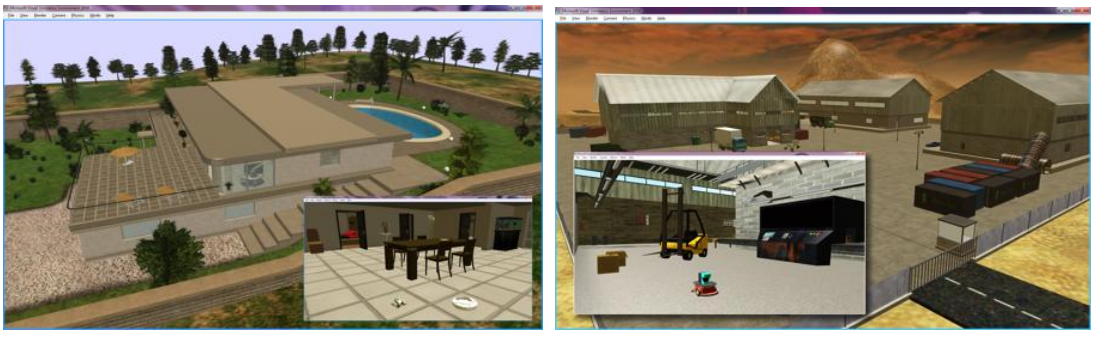


Figure 5.19 – Environment Simulation

We will use it to drive cars in a city, simulate a smart house and test applications with the simulated nao robot.

Nao Robot Platform

Nao robot is a hardware platform controllable from a TCP-IP network connecting the NaoQi Proxy (by wifi or wire). We also use the SDK 1.10.37 to drive the robot in more complex tasks than those already programmed.

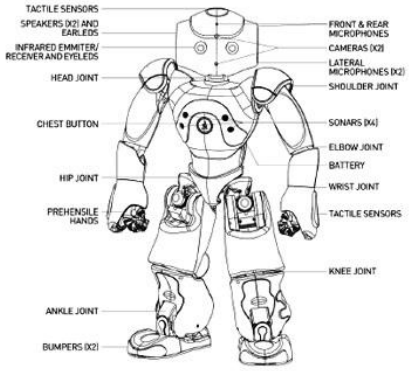


Figure 5.20 – Nao Robot Platform

Nao robot platform[25] includes a lots of sensors and actuators: Tactile sensor on its head, Infrared emitter and receiver, 2 video camera, 4 microphones, 2 Loudspeaker, 2 sonar transmitters, 2 sonar receivers, 20 coloured diodes, chest button, feet bumpers, 8 feet pressure sensors (Force Sensitive Resistors), 25 degrees of freedom, 2 axis gyrometer and 3 axis accelerometer.

We use this platform to interact with human users and environment, and mostly demonstrate our use cases in Chapter 6 (Figure 5.20).

5.3 Performance

5.3.1 Introduction

Measures of performance are necessary to evaluate the solution. We measure three kinds of performance on a test platform: *networking load*, *database access time*, and *inference time*. And we introduce the *cognitive workload* to check the overall system regarding a task or an activity.

For these experiments, we use the 2 previous computers: a PC with an i7 (64 bits, 2.67 GHz, 6 GB RAM, Windows 7), a RoBoard with a Vortex86DX (32 bits, 1GHz, 256 MB RAM, Linux Debian 5.0.2) and another PC with a Pentium 4 (32 bits, 3 GHz, 2 GB RAM, Windows XP). A 1 gigabit/second network card is shared between computers. Different performances can be measured depending on the hardware solution chosen by the developer.

5.3.2 Networking load

The networking load indicates the maximum number of messages per second that can be transmitted from our RoBoard to the PC i7 and added to a reception queue with 2 methods. The first method consists of executing the CURL command on the RoBoard to send messages and receive them with Apache in a Webdav directory. The second method involves developing our own client-server

[25] NAO Full Documentation http://users.aldebaran-robotics.com/docs/site_en/index_doc.html

protocol based on TCP/IP sockets. The results are shown in Figure 5.21. Attention must be paid to the distribution of events between the hardware parts and other parts of the complete system in order to meet message queuing limitations. This is why lower abstraction levels working on low level data must be wisely combined in the hardware then sent to the concentrator. A higher level of abstraction is then required. Another solution is to interconnect computers and hardware components with optic fibber and build a faster processor into the networking card, with the disadvantage of cost and some inter-operability problems with respect to other computers on the slower Internet.

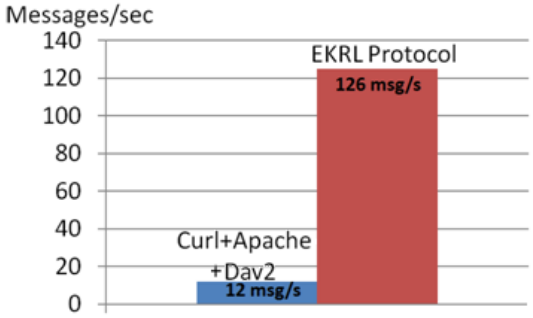


Figure 5.21 – Networking load – Event message transfer rate

5.3.3 Knowledge base access time

Knowledge base access time is the worst measurement obtained from 10 read access attempts on the *ra* SQL table containing different numbers of events. We have to avoid the disk/memory buffering effect, where subsequent measurements are faster than the first one; we can do this when the amount of events exceeds the buffer size (greater than 100000 events in the 3 machines). All events are read in a single query.

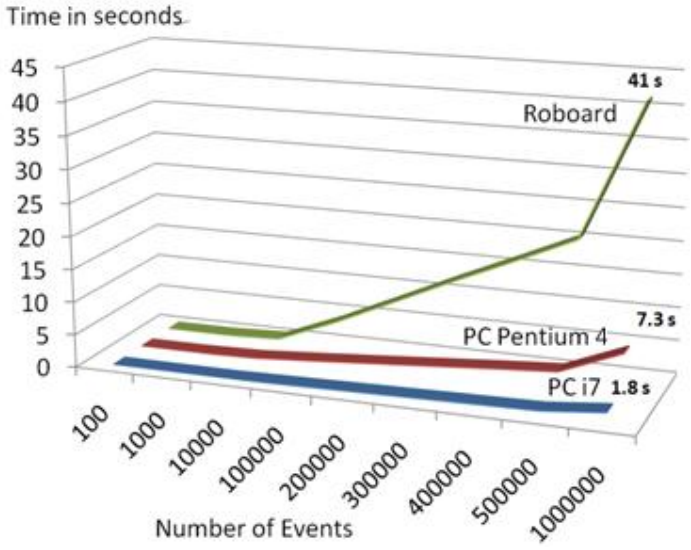


Figure 5.22 – Knowledge base access - Rate of events read into the database

The results are displayed in Figure 5.22. Even though we are reliant on database engine performance, our system is still very fast. Obtaining better performance means developing our own database engine limited to query operations on the SQL *ra* table. Note in this section that we focus on querying

the agent's memory and we consider all past facts stored at the time of reading the memory. Actually, there is some latency before new events appear in memory, but we consider the memory contains a larger number of events than the number of incoming events.

5.3.4 Inference time

Inference time is a measurement of the maximum number of inferences performed per second. It depends on the number of data read from the database. We build a data set to insert the concepts and models into the ontologies of our agents (same memory content). Considering there is no queuing event, we send a query on facts to the agent. The query is an EKRL event with all possible roles filled, containing a combination of five arguments taken from the concept ontology. The total execution time of a single inference is equivalent to a database access time added to a complex comparison processing time (Algorithms in Section 3.7.3). Our query takes very little time, especially with the i7 processor, so we measure the required time for the agent to execute *nbe* inferences, where *nbe* is the maximum number of events that can be returned by the database. Measurements are displayed in Figures 5.23 and 5.24.

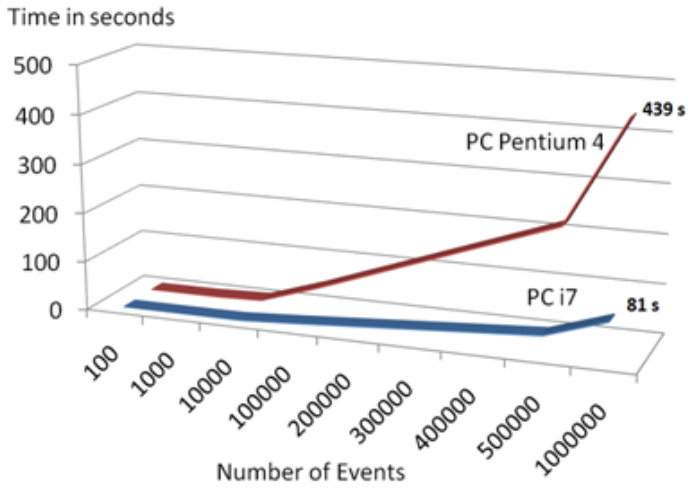


Figure 5.23 – Inference time needed to process the maximum number of events

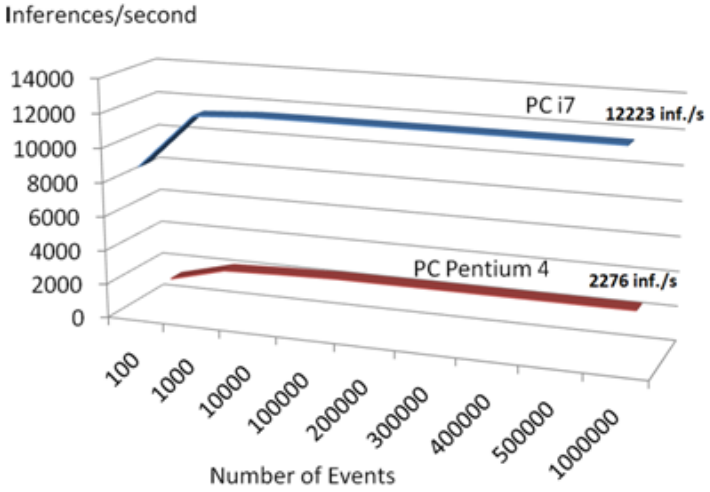


Figure 5.24 – Rate of inferences by second

5.3.5 Consistency and Robustness

The major drawback of models is that development concerns cannot truly be investigated by themselves, since concerns tend to affect one another. Successful and precise product development supported via models thus requires that common assumptions and definitions are recognized and maintained in a consistent fashion. In other words, having models with inconsistent assumptions about a system's expected environment reduces their usefulness and possibly renders invalid all solutions based on them (such as analyses and simulations). To date, however, transitioning information between models is still not a straightforward task despite the massive attention this problem has received from the software engineering community. In our system, conceptual and behavioural models are fixed knowledge at different levels of abstraction and models' definition is unique. All unknown event models are ignored. Two rule models can produce contradictory events which can be detected *a posteriori* by other agents and services. Model checking of concept classes (OWL classes) and concept instances (OWL individuals), is a standard ABox. Gravier et al. (2011) studied reasoning with context and A-Box revisions in semantic context-aware systems. Automatic analysis of models and subclass checking can be realized by verifying logical entailment of OWL TBox and by execution of the matching function of the inference engine. The latter requires sending a sequence of events to agents and checking produced events stored in the agent's memory. Consistent theory in second-order logic (Arnborg, 1994) is equivalent to check satisfiable OWL classes (true propositions in determined conditions). Our framework provides strong support for logical heterogeneity in models.

To check consistency and robustness, we define a test procedure during 25 s. We control the number of generated events sent to input by a virtual service in the EKRL concentrator. We monitor output events of the agent during the test by looking in the messages queue of a targeted service. We monitor also agent's memory after the test by checking « correct » events found in the agent's memory (knowledge base in the database). We focus our analysis on ignored events compared to well compose events to insure generated meaning (i.e. consistency). After a verification of event models, we check the good correlation between expected output and inputs. We also wanted to check robustness with noisy events and by increasing events load in time.

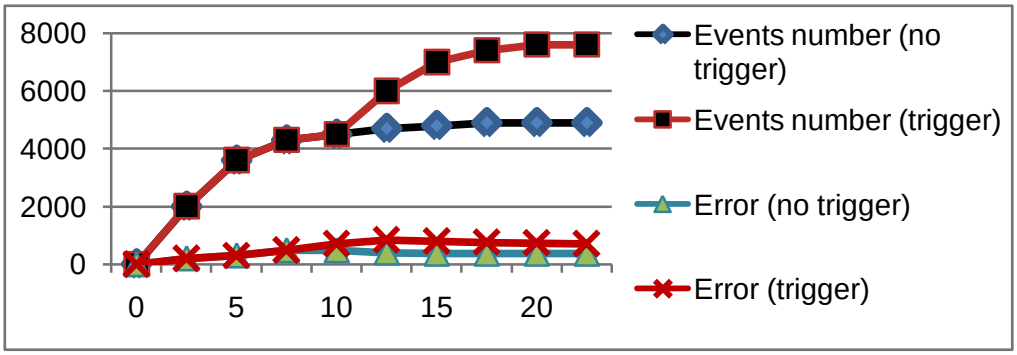


Figure 5.25 – Rate of inferences by second

Results of the experiment on this agent are presented in Figure 5.25 and Table 5.1. Consistency decreases if too much events occur. Performance of agents decreases due to their processing speed but robustness is always good because,

in this human situation, events are very redundant so there is no impact on outputs. Agents are able to ignore events when they are not matching predicate name, time, location and other roles in event. Robustness remains good in case of noisy data without taking into account uncertainty measure coming from sensors and hence corrupted data. Consistency is weak because false data in events impact correct events at 8%.

Events	Consistency	Robustness
Normal **<1000 events/s**	100%	100%
Overload **>1000 events/s**	71% 29% ignored	100%
Noisy **50% unknown events**	50% 50% events ignored	100%
Noisy **50% false data in known events**	42% 17% events not in time so they are ignored	96%

Table 5.1 – Consistency and Robustness Results

5.3.6 Cognitive Workload

In order to manage the energy, the awareness in the priority of contexts and the stability of the architecture, it is very important to measure the cognitive workload (mental, physical and temporal demands) of the system in the realization of tasks. Hart and Staveland (1988) have proposed multi dimensional rating scales after a study lasting several years and a lot of patients. Finally they found 10 major factors (Table 5.2) that can be combined into 16 different experiments. TLX means Task Load Index.

$$Measure\ workload = \left(\frac{\sum_{i=1}^{6} Rating_i * Weight_i}{15}\right) \qquad (Formula\ 24)$$

where $Rating_i$ is a score [0,100] for the i^{th} measure of Table 5.2, $Weight_i$ corresponds of one of the 15 factors pairs of Table 5.3 (Select the members of each pair that provided the most significant source of workload variation in the measured tasks). In our architecture, Mental and physical demands can also be communications load (network latency and bandwidth) and, agents and services management load (composition, orchestration and maintenance effort).

This measure is subjective, but well adapted to human workload and our agent workload. Benefits of subjective evaluation are:

- *Direct measurement*: no inference made from data (e.g. physiological or behavioural)
- *Measures readily available*, easily adaptable to different work environments
- *Low cost* method
- *Atheoric* because it is not necessary to keep a theoretical framework or conceptual sophistication to use the method.

It is often used as a means to validate more objective methods. Comparatively, other performance measures are:

- Cooper-Harper Scale or MCH: Modified Cooper-Harper scale[26]
- Bedford scale (one-dimensional) (Roscoe and Ellis, 1990)
- OW: Overall workload (one-dimensional scale from 1 to 20)
- SWAT: Subjective Workload Assessment Technique (multidimensional) (Reid et al. 1981)
- ISA: Instant Self Assessment of Workload (Jordan, 1992),
- MACE: Malvern Capacity Estimate (Goillau and Kelly, 1996),
- DRAWS: Defence Research Agency Workload Scale (Farmer et al. 1995).

Barrouillet et al. (2004) define the cognitive load *CL* of a task involving an uninterrupted series of retrievals as a function of the number and difficulty of retrievals to be performed in a given period of time:

$$CL = \sum_i \frac{a_i * n_i}{T} \qquad \textit{(Formula 25)}$$

where n_i is the retrievals of type i, a_i the difficulty of these retrievals i, i.e. the time each of these retrievals totally capture attention, and T the total duration during which these retrievals are to be performed. This work can be realized by querying facts stored in memory of agents with specific query models.

Title	Endpoints	Descriptions
OVERALL WORKLOAD	*Low/High*	The total workload associated with the task, considering all sources and components.
TASK DIFFICULTY	*Low/High*	Whether the task was easy or demanding, simple or complex, exacting or forgiving.
ACTIVITY TYPE	*Skill Based/ Rule Based/ Knowledge Based*	The degree to which the task required mindless reaction to well-learned routines or required the application of known rules or required problem solving and decision making.
MENTAL DEMAND	*Low/High*	How much mental and perceptual activity was required (e.g., thinking, deciding, calculating, remembering, looking, searching, etc.)?
PHYSICAL DEMAND	*Low/High*	How much physical activity was required (e.g. pushing, pulling, turning, controlling, activating, etc.)? Was the task easy or demanding, slow or brisk, slack or strenuous restful or laborious?
TEMPORAL DEMAND	*Low/High*	How much time pressure did you feel due to the rate or pace at which the tasks or task elements occurred? Was the pace slow and leisurely or rapid and frantic?

[26] http://en.wikipedia.org/wiki/Cooper-Harper_rating_scale

PERFORMANCE	*Good/Poor*	How successful do you think you were in accomplishing the goals of the task set by the experimenter (or yourself)? How satisfied were you with your performance in accomplishing these goals?
ENERGY	*Low/High*	How much energy or how hard did you have to work to accomplish your level of performance?
FRUSTRATION LEVEL	*Low/High*	How insecure, discouraged, irritated, stressed and annoyed versus secure, gratified, content, relaxed and complacent did you feel during the task?
STRESS LEVEL	*Relaxed/Tense*	How anxious, worried, uptight, and harassed or calm, tranquil, placid, and relaxed you felt.

Table 5.2 – NASA-TLX Rating Scale Definitions (Hart and Staveland, 1988)

Energy or Performance	Temporal requirement or Frustration	Temporal requirement or Energy	Physical requirement or Frustration	Performance or Frustration
Physical requirement or Temporal requirement	Physical requirement or Performance	Temporal requirement or Mental requirement	Frustration or Energy	Performance or Mental requirement
Performance or Temporal requirement	Mental requirement or Energy	Mental requirement or Physical requirement	Energy or Physical requirement	Frustration or Mental requirement

Table 5.3 – NASA-TLX Sources of workload

5.4 Inference Engine Complexity

It is critical that we define complexity metrics that focus strictly on the nature of the problem and not on the nature of a particular solution. The problem of the inference engine is to manage knowledge answering queries and producing new events with rule models using matching function. A metric based on counting the number of rules, to take one example, is equivalent to making the assumption that the semantic architecture that does the reasoning is, at its core, a production system. We need avoid this trap. There are various general sets of factors to consider when deciding how complex a problem or task is. First, we want to distinguish between the complexity of a problem (a.k.a. a semantic task) as opposed to the context (i.e., problem space) in which the problem exists. In any system, (e.g., an automobile) there are simple problems that need to be dealt with (e.g., we're out of gas) and complex problems (e.g., an intermittent short in the wiring). System complexity and problem complexity are not totally independent however. Fuelling a space shuttle is more complex than performing the equivalent task with the family car. The complexity of problem solving and decision making is not the same as the complexity of learning. An additional issue is the complexity of executing a course of action (i.e., is cooperative

behaviour by a group of agents required?). A good metric for *problem complexity* incorporates properties of both the specific problem at hand as well as the overall context (i.e., the problem space). To cover both aspects, we use two sets of characteristics:

- *Problem Space* Characteristics: How many concepts & classes (i.e., symbols) in the ontology (i.e., degree of heterogeneity)? How many objects and facts in the problem space (i.e., size of the problem space as presented to the observer)? How dynamic is the state of the objects in the problem space? How tightly coupled (i.e., interdependent) are object states? How transparent is the observable problem space (i.e., expected ratio of known to unknown facts under best-case/normal/worst-case operating conditions)?
- *Problem Characteristics*: How 'noisy' is the problem (i.e., ratio of relevant to non-relevant facts for any given problem/task/goal)? How visible is the problem (ratio of relevant facts that may be obtained via observation or direct questioning of other actors vs. those that require inference & reasoning to obtain)?

These characteristics will be useful to make choices of architecture in order to implement the application in the next chapter. And, especially, take care of requirements and other criteria like heterogeneity, size, and interdependence of the problem space.

Inference engine of Agent (Section 3.7.3) has been explained. We propose to detail the computation of the complexity. We remind the reader that there are several types of models in the ontology of models: event models, query models and rule models. In querying the agent's memory, we are interested in two types of model:

- Direct *Query models* to find matching facts where roles and arguments of a query match up with the roles and arguments of facts stored in the agent's memory. In this case, the logic of the inference engine is first order. We may note that only a sub-tree of the model ontology is searched, depending on the predicate location of the query model.
- *Rule models* that are sent (post-condition) by the agent if some facts have expected event models defined as a combination of arguments for the first role of these rule models (pre-condition). The argument thus contains a composition of other event models of lower abstraction levels (in the case of a fusion process). Such a matching operation is recursive. Normally, an agent is designed to manage fusion (or fission) at a defined abstraction level in the architecture and fuses (or separates) events of the previous (or next) level only. This advantage limits the recursion depth to a second level of abstraction and thus the logic of the inference engine is limited to second order.

For a direct facts query, in the first case, we query the agent (its inference engine) on facts whose arguments contain a composition of concepts. The algorithm complexity is based on the number N of facts stored under the event model predicate in the event ontology, which is a tree structure. As our roles and arguments are indexed in a row of the *ra* database table, complexity in space is $O(N)$ and maximum time of execution is $N*ACT$, where ACT is the argument calculation time depending on the type of this argument (string, number, time value, space value, and so on). For example, Figure 4.9 shows the

Exist:Available Service model. This model can be used as a direct query to search facts corresponding to new available Web services over a specific period of time and limited to a location with the SENDER, DATE1, DATE2 and LOCATION roles.

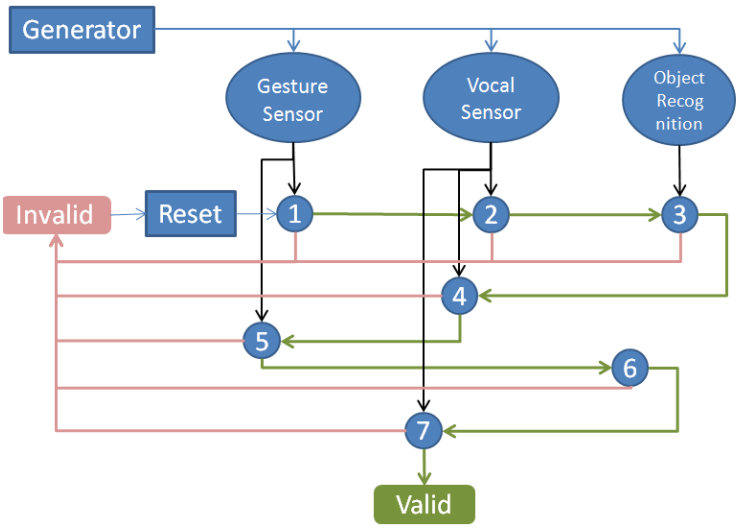
For a rule model, we extract all models combined in the first argument and we query the agent on facts directly matching these models. Complexity in space becomes *O(M*N)* where *M* is the number of combined models and *N* is the number of facts under all combined models. Maximum execution time is *M*N*ACT*, where *ACT* is the *argument calculation time* of the rule model. For example, Figure 3.34 shows the Rule model to compose the "*Behave:PutThatHere*" fact if the precondition is validated.

In our architecture design (Figures 4.4 and 4.5), agents can be arranged by layers as shown Figure 3.19. Each layer is restrained to an abstraction level of fusion or fission. The agent's memory can be made to contain only the relevant event models to reduce search space and increase execution speed.

5.5 Inference Engine Temporal validation

5.5.1 Introduction

Our system is asynchronous. In order to validate formally our temporal logic integrated in the aggregation function of our inference engine. We decided to check it on the "PutThatHere" fusion scenario defined section 1.3 with CPNTools[27] (Jensen, 1991). A CPN model of a system is an executable model representing the states of the system and the events (transitions) that can cause the system to change state. CPN Tools is an industrial-strength computer tool for constructing and analysing CPN models. Using CPN Tools, it is possible to investigate the behaviour of the modelled system using simulation, to verify properties by means of state space methods and model checking, and to conduct simulation-based performance analysis.



Comparators

N°	Validation Test	Time start (s)	Time end (s)
1	Point	.3	2.2
2	Put	.5	1.5
3	Glass	1	2.1
4	That	1.5	2.5
5	Point	3	4.4
6	Table	3.4	4.5
7	Here	3.8	4.5

Figure 5.26 – "PutThatHere" Temporal Validation

[27] CPNTOOLS http://cpntools.org/start

Figure 5.26 represents the temporal validation synoptic of the inference engine processing the matching of the rule model "PutThatHere". Comparators at right present the exact name and time (minimum start, maximum end) of the expected events in this time line. Figure presents only the useful events but in the network simulation we will generate these events and many other false events. In the "PutThatHere" scenario, two modalities: *vocal* for saying "*put*" "*that*" and "*here*", and *gestural* by pointing to the object (corresponding to "*that*") and the place where the object must be placed (corresponding to "*here*"). Our schema is equivalent to the precondition of Figure 3.33 and is composed of 3 sensor services, a generator of events and 7 comparators. If a comparator returns false then the sequence is invalidated and the loop is reset. If the 7th comparator is true then the event is validated. The generator randomly activates the 3 services and gives it a random number. The random number allows service selecting a correct or an invalid event of its database. Then the service generates an event name attached to a start time and an end time, and sends it to the inference engine of the agent modelled by the comparators of the figure.

We want to verify the following conditions:

- *Events Completeness* to check if all events required in the precondition are present to evaluate the corresponding predicate (to produce composite event);
- *Event arrivals order* to verify if events sent from modalities are respecting the expected order;
- *Temporal aspect* of vocal and gestural commands, taking into account the time needed by a modality to detect a command and the time to merge data;
- *Consistency* to check if recognized entities exist in the ontology of concepts.

5.5.2 "PutThatHere" Validation

Coloured Stochastic Petri Nets

A Petri net is a quadruple (P, T, F, W), where

1. P is the finite set of *places*;
2. T is a finite set of *transitions*;
3. $P \cup T \neq \phi$;
4. $F \subseteq \{P \times T\} \cup \{T \times P\}$ is the flow relation; and
5. $W: F \rightarrow N - \{0\}$ is the weight function, which associates a nonzero natural value to each element of F. If no weight value is explicitly associated with a flow element, the default value 1 is assumed for the function.

This notation is only used in this section and specific to Petri nets.

Events Generator

Figure 5.27 represents a part of the coloured Petri network describing the generator responsible of generating random events designed by Djenidi (2004). We use CPN Tools version 3.2.2.

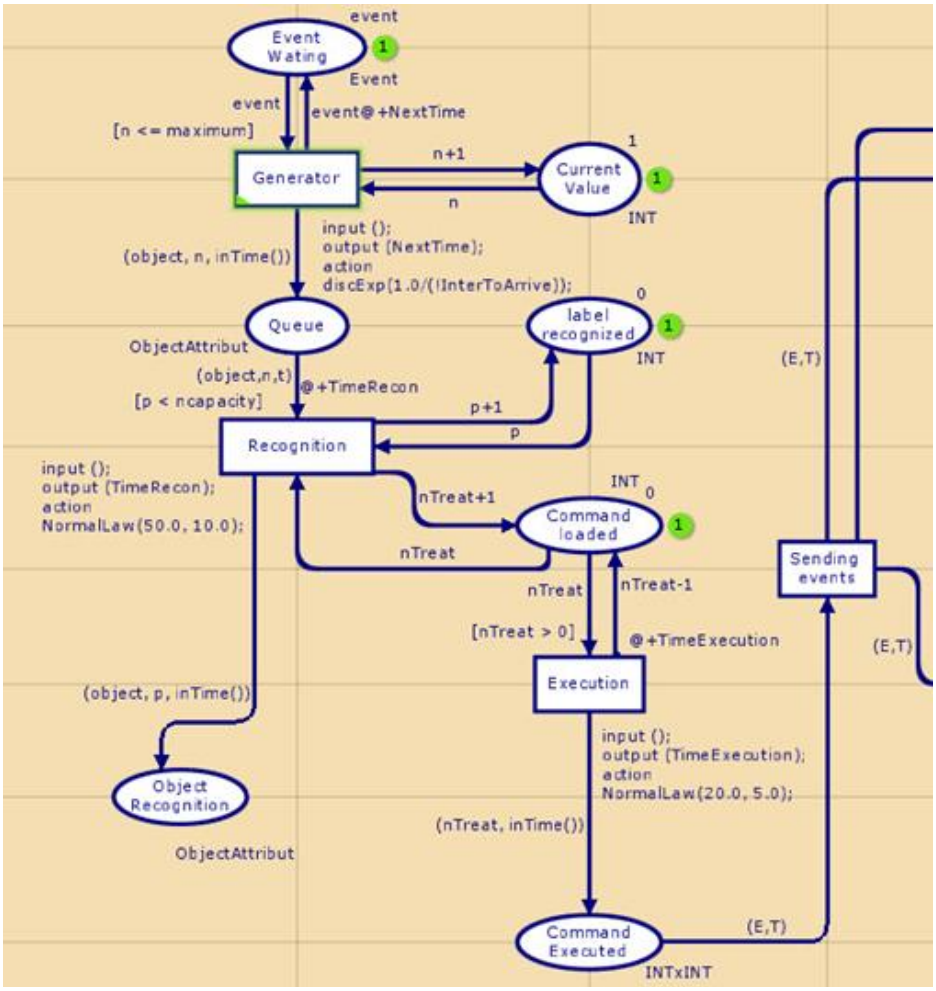


Figure 5.27- Random Events Generator

```
Declarations
  colset INT = int;
  colset Event
  colset result= product INT*INT*INT*INT*INT*INT*INT;
  colset Object = with object timed;
  colset ObjectAttribut = product Object * INT * INT;
  colset INTxINTxINT= product INT*INT*INT;
  colset INTxINT= product INT*INT;
  fun round x = floor(x+0.5);
  fun inTime() = IntInf.toInt(time());
  fun discExp x = round(exponential(x));
  fun NormalLaw(x,y) = round(normal(x,y));
  val ncapacity = 40; val InterToArrive = ref 0.5; val maximum = 50;
  var NextTime, TimeRecon, TimeExecution, n, p, nTreat,t:INT;
  var E,T,W,W2,W3,Tb, Te, Loc,Loc2, En, En2: INT;
```

Figure 5.28 – "PutThatHere" CPN Declarations

The stochastic coloured Petri net allows modelling processing time which follows laws of probability (see codes associated with the Recognition and Execution

transitions) and, in the same network, the Generator transition associated with the place EventWaiting models a stochastic random generator. When this transition is fired after a time period which follows an exponential law (see code associated with the Generator transition), a token of colour (i.e. a type) ObjectAttribut specified by the expression in CPN-ML (object, N, intTime()) is generated in the Queue's place. At the same time the n+1 and event tokens are generated in the EventWaiting and CurrentValue places, respectively. The token in *Queue* is composed of three attributes. The first, *object*, symbolizes the nature of the fragments of information arrived in the place. The second is the value of the number of arrivals and the third is the value of the arrival time of the tokens in the place. This modelling of events, their temporal characteristics and activities with random times represent temporal and stochastic aspects used in the coloured Petri nets model presented below.

"PutThatHere" Matching Solution

The second part of the network (red part Figure 5.29) is dedicated for the vocal commands "put", "that" and "here". According to the table in figure 5.26, we gave each vocal command an index and an interval of time (for example "put", 2, 0.5, 1.5). These data can be seen in the state *List Of Words*. The *Vocal Sensor* State receive events *E* in a certain time *T* randomly (the function *inTime ()* in declarations convert the time to an integer, the converted value is *T*). *Vocal Sensor* is related to 3 comparators: *Comparison of 1st word* place, *Comparison of 2nd word* place and *Comparison of 3rd word* place. Each place receives a random event *E* in a certain time *T*. *E* and *T* are compared with the data of *List Of Words* state that sends *W* which is the index of a command, *Tb* and *Te* which are the beginning time and the end time of a command respectively. The *if condition* compares these data, so if *E* equal to *W* and *W* equal to 2 (index of the command "put" according to table) and *T* (the time of a random event) is between *Tb* and *Te*, then 2, which means that the command "put" is fired and the word is recognized, else *E* (the random event) is fired and sent to *Feedback* and the word is not recognized. This means that the vocal sensor is sending another word that doesn't have a meaning according to our system. The same comparison is done for "that" (index 4) and for "here" (index 7) commands using *Comparison of 2nd word* and *Comparison of 3rd word* places.

The green part (Figure 5.30) represents the gesture models like "point to a location". The state *locationOntology* is a knowledge base that contains all recognized locations before. The state *GestureSensor1* receives random timed events from the generator via *E* and *T*. In this part, two comparisons are done, a comparison for the old location where the entity stands, and the new location where the entity must be placed, this comparison is done by using the two places *Comparison Of Location1* and *Comparison Of Location 2*. The *if condition* compares each location to the random timed event (*E*, *T*) sent by the *GestureSensor1* state. For *location1*, if *E* equal to *Loc* and *T* is between *Tb* and *Te* then 1 (index of point defined in the table) "point" is fired and the initial location is located, else *E* (the random event) is fired and sent to *feedback* and the location is not recognized. The same comparison is done for the second location *Loc2* which is the new location where to place an entity.

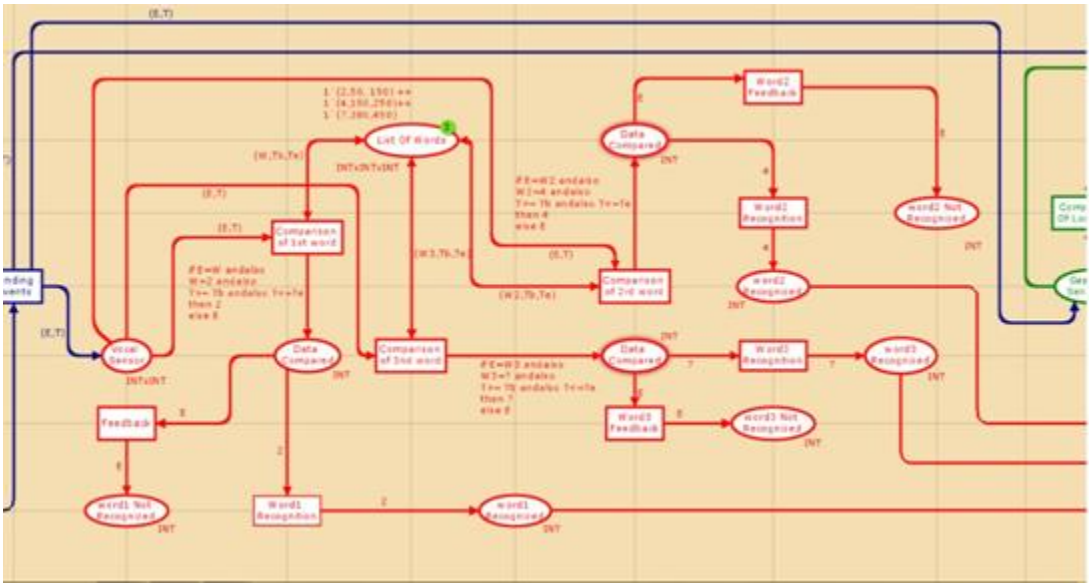


Figure 5.29 – comparators for vocal sensor

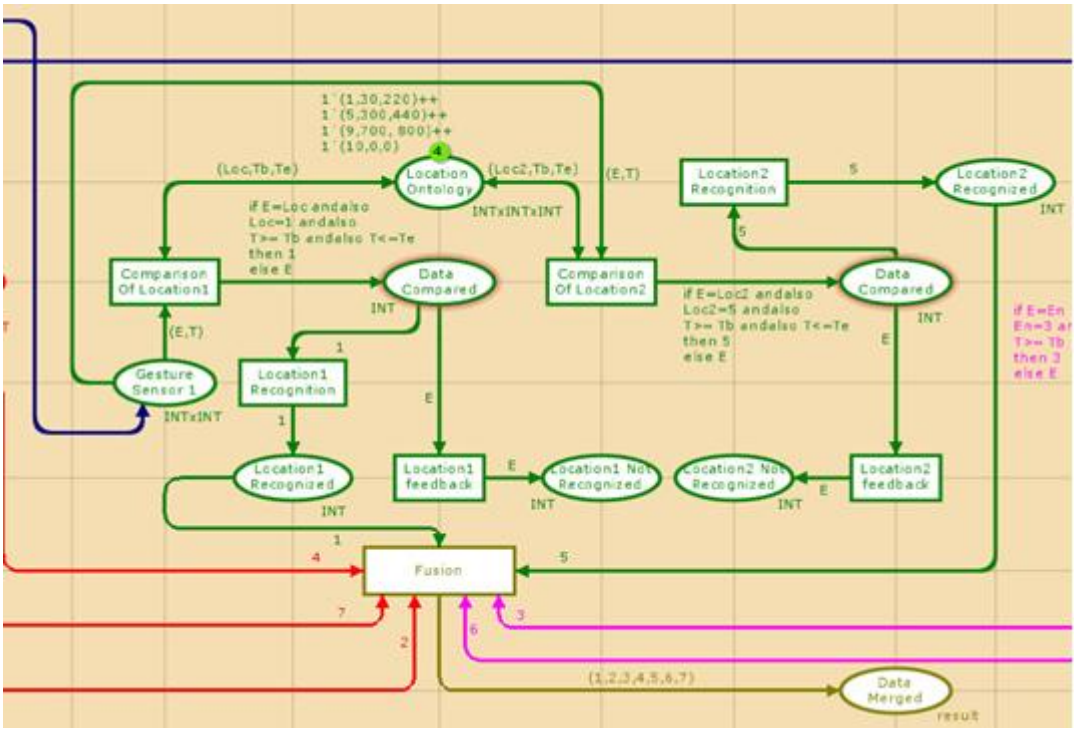


Figure 5.30 – gesture sensor and its comparators of location

The rose part (Figure 5.31) represents the part of the entity to place. It could be an object, a human or an animal (in our scenario the entity is an object). *ThingOntology* state is a base knowledge that contains all recognized things before. The state *Object Recognition* receives random timed events from the generator *(E, T)*. In this part two comparisons are done, the first one is for the object to place ("glass" which has the index 3 in the table) and the second one is for the place to put the object on ("table" which has the index 6 in the table). These comparisons are realized using the two places *Comparison Of Entity1* and *Comparison Of entity 2*. The if condition compares each entity to the random timed event *(E, T)* sent by the *Object Recognition* state. For entity1, if E equal to E_n and T is between T_b and T_e then 3 (index of "glass" defined in

the table). "glass" is fired and the object to place is recognized, else the random event *E* is fired and sent to feedback and the entity is not recognized. The same comparison is now done for the second location En2 which is the other object or location where to place the 1st object on.

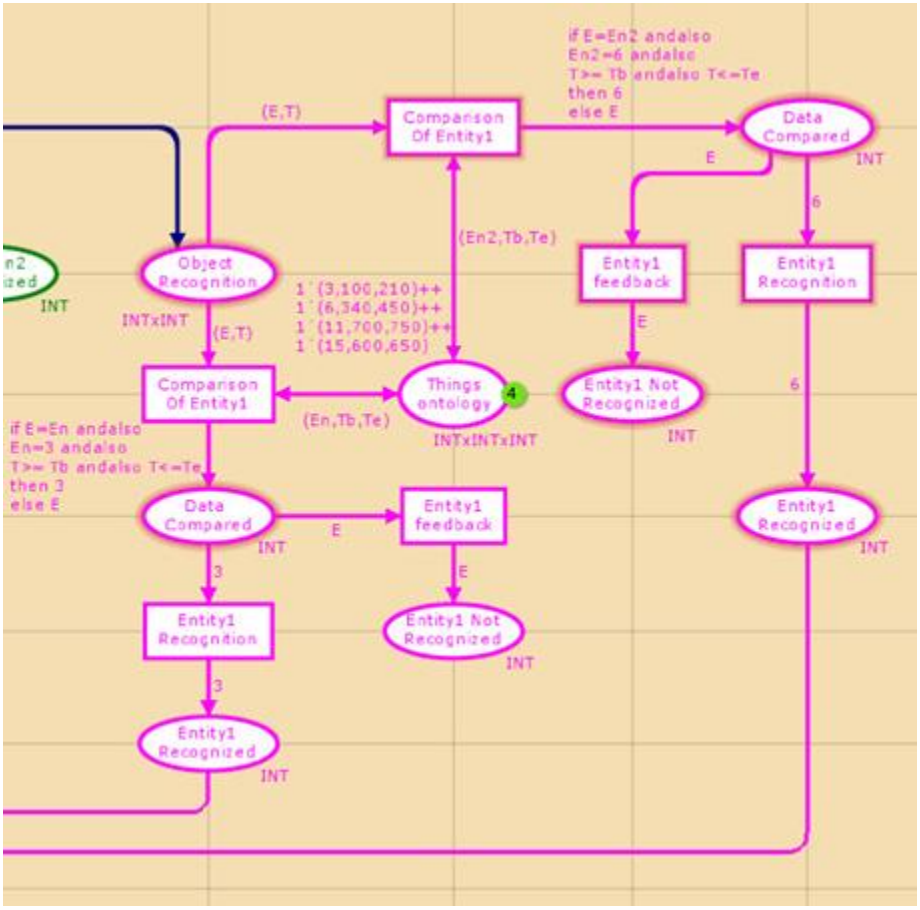


Figure 5.31 – Things ontology and its comparators of entities

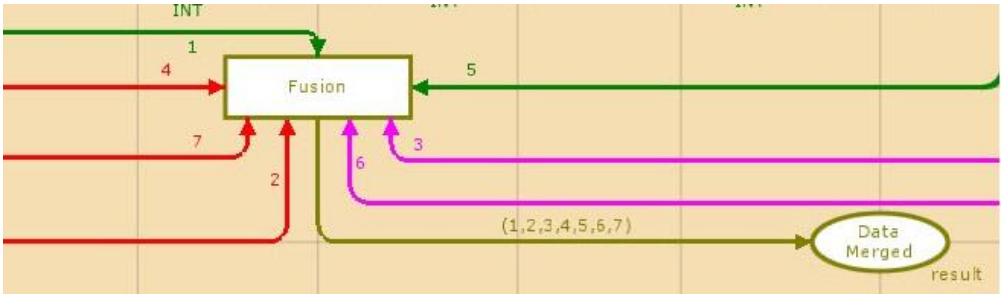


Figure 5.32 – "PutThatHere" Fusion

Once the seven comparisons are successful and the seven indexes 1 to 7 of the table are sent to the *Fusion* place (Figure 5.32), we are ready to merge these data so that the state *Data Merged* is fired and the result will be the "PutThatHere" event { "put", "that", the object to place, the initial location, "here", the new location or object to put on }.

5.5.3 General Case Validation

We would use the last experiment in order to generalize the procedure to any events case and thus fully validate our matching algorithm. For this purpose, we

use the same timed event generator. Event and time of the event are received and sent to the Matching function of the inference engine. Unlike the previous network limited to the recognition of the "PutThatHere" event, this new network manages three main functions:

- *Event model matching* compares the incoming event containing the model name of the event or fact (predicate name) and its time (DATE1 and DATE2 roles) to the event models in the ontology of models. This matching is used to extract past facts regarding to a query event. This function has been fully detailed in Chapter 3 and in Section 5.4.
- *Rule model matching* compares the incoming event containing model name and arguments of the roles to the precondition of the rule models in the ontology of models. For instance, it is the case of "PutThatHere" fusion model. This matching is used to check precondition of a rule model and send this rule model after aggregating the roles of this event with role contents of precondition events. This algorithm is not limited to fusion process but works also fine for fission process. For fusion process, several incoming events fire only one event. For fission process, several events are fired from only one sequence of incoming events. Aggregation still remains the same. This function has also been fully explained in Chapter 3 and in Section 5.4.
- *Sequences' cleaning cleans* saved sequences of events required for matching rule models precondition with current event and time. A sequence is a combination of event names used to compare the events in the precondition to those that are received and respecting the time order of events in the sequence. These sequences are saved in a list for a limited time (if *TimeWin* variable>0) and for an unlimited time (if *TimeWin* variable=0). The first case means only last events in the time window stays in the sequences list, from Now minus TimeWin to Now. A future work could be managing this time window regarding to the concerned event. The latter is useful if the agent is limited to some rule events. And it means, after sometimes or long time, size of the sequences list will be same as the number of rule models. The problem in this case will be that after receiving any event, all matching sequences will be found and a large amount of aggregated models will sent to other agents and services.

Figure 5.33 present a general coloured stochastic network. If the event name is in the ontology of models, the incoming event is a query. If facts related to this event model (i.e. query model) exist in the time period defined by DATE1 and DATE2 roles of the event, they can be sent to other agents and services (red block in the figure). If the event name is in the precondition of rule models, the event may be the only model to be in rule models precondition or may be a part of a sequence of events in the rule models precondition. Name, precondition and time of the matching rule models are extracted of the ontology of models. Sequences of events of the rule models are then added to the Sequences if they are not in. Previous sequences in the *Sequences* list and event are concatenate to build a new sequence. Each new sequence and time is also compared with rule models. This is realized by the *Matching – InsertSequenceOnce - GetSequence* loop. Finally, all matching rule models which are aggregated with all received events of the precondition are also sent to other agents and services.

From this schema, we generate a source code. We execute the applications on several scenarios for different models in the ontology of models, different sequences of events generated and we check the recognition of composite events related to existing models. We will now explain our experiments and present how the matching function generates the results.

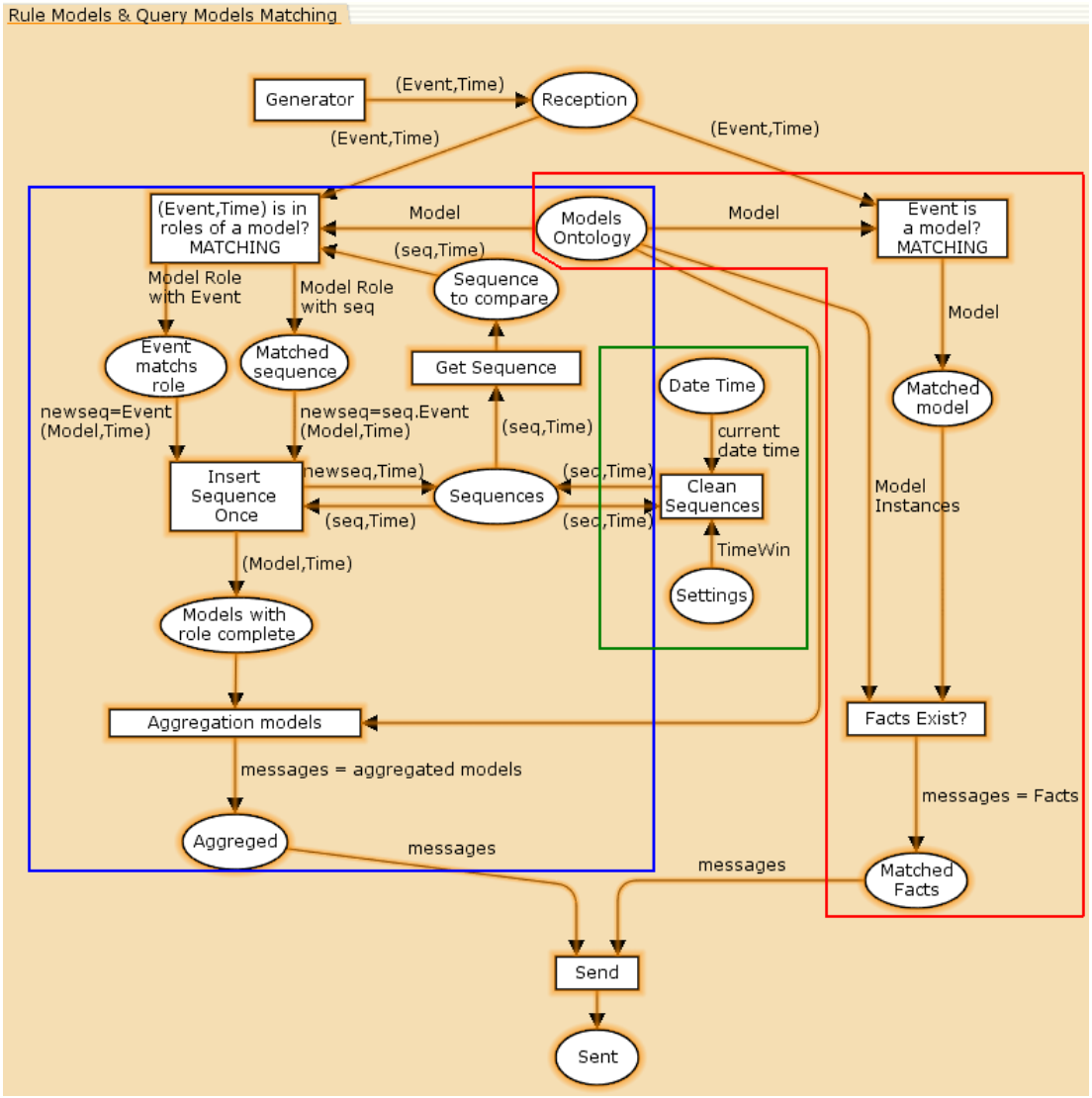


Figure 5.33 – Matching of events

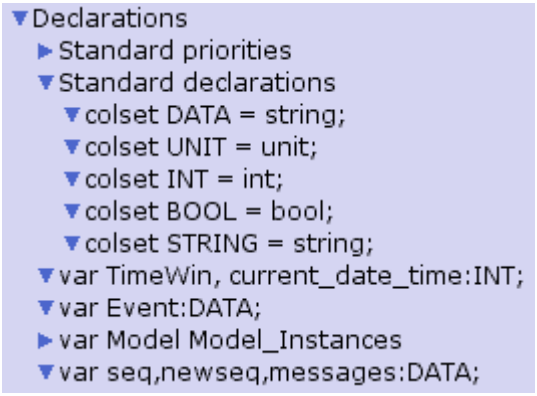


Figure 5.34 – Matching CPN Declarations

Rule Model Matching Experiments

We experiment our rule models matching algorithm. We don't present direct matching as it is a simple comparison of model name and all its roles with existing facts already present in the ontology of models. We will send some events in a specific order to check the matching through the building of sequences of events present in the precondition of the rule models.

Rule Model Name	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
Sequence of events in precondition	A	AB	ABCD	ABCD	ABCF	AC	B	DE	ED	D

Table 5.4 – Rule models in the ontology of models

Table 5.4 is the list of rule models and the sequence of events in the precondition. Rule models names are M1 to M10. Sequences are composed of event models A to Z where a letter stands for a fulfilled predicate frame. These sequences are in the precondition role of the rule models. We take several significant examples of sequences: all event models are not in the list so should be ignored (for example, there is no sequence with only C), order of events (A before B in AB or A before C in AC) should be respected, some sequences begin with the same subsequence (M2, M3, M4, M5 contain A and AB) and are even equals like for rule models M3 and M4.

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Event	E	C	E	C	A	A	B	C	D	E	D	B	C	D	F

Table 5.5 – Incoming Events

Table 5.5 shows the incoming events at each time. To simplify the writing for a better readability and understanding, we assume one letter (instead of a complete string) is used to identify an event and its roles including time. We also assume *Time* as an integer index. Discrete time *Time=1* means any real time *t1* in seconds before *Time=2* (real time *t2* in seconds) where t1<=t2. At each discrete time, a new event is coming to be compared alone with rule models precondition sequences and to compared in association with the other sequences already present (concatenation). If two events A are coming at *Time=1* and at *Time=5*, the *InsertSequenceOnce* function only keeps the last one at the end of the list of sequences. This function inserts a new event in the Sequences list only once. So *A(Time=1)* will be replaced by *A(Time=5)*. There is absolutely no impact on the algorithm because roles of DATE1 and DATE2 of A are the same for *Time=1* and *Time=5*, it's the same event with the same roles (including dates) to be compared. This table is interesting because main cases are represented to ensure algorithm works well: it must ignore unknown and noisy events (Time 2 to 4), and manage the order of sequences (Time 6 to 9) and repeatability of produced events for a time period (Time 11 to 14).

Infinite Time Window Execution

Table 5.6 presents the evolution of the list of sequences with an infinite time window and the fired rule models in outbox where sequences of events are matching these rule models. Comments give explanations on what is happening.

Time	List of Sequences	Outbox	Comments
1	E in M9		E exists in rule model M9
2	E in M9		C ignored because no rule models begins with C and there no sequence EC in rule models.
3	E in M9		
4	E in M9		C ignored
5	E in M9 A in M1, M2, M3, M4, M5, M6	A in M1	Precondition of M1 fired by the first A. Sequence in light red is the new one.
6	E in M9 A in M1, M2, M3, M4, M5, M6	A in M1	Precondition of M1 fired by the second A. List of sequences not changed (normal).
7	E in M9 A in M1, M2, M3, M4, M5, M6 AB in M2, M3, M4, M5 B in M7	AB in M2 B in M7	B has fired preconditions of M2 and M7. Previous A with B give AB.
8	E in M9 A in M1, M2, M3, M4, M5, M6 AB in M2, M3, M4, M5 B in M7 AC in M6 ABC in M3, M4, M5	AC in M6	C has fired precondition of M6. Sequences AC, ABCD, ABCF have been found. Note the two different models M3 and M4 are found.
9	E in M9 A in M1, M2, M3, M4, M5, M6 AB in M2, M3, M4, M5 B in M7 AC in M6 ABC in M3, M4, M5 ED in M9 ABCD in M3, M4 D in M8, M10	ED in M9 ABCD in M3 ABCD in M4 D in M10	D has fired preconditions of M9, M3, M4 and M10
10	A in M1, M2, M3, M4, M5, M6 AB in M2, M3, M4, M5 B in M7 AC in M6 ABC in M3, M4, M5 ED in M9 ABCD in M3, M4 D in M8, M10 DE in M8	DE in M8	E with previous D fired precondition of M8. Note there is no rule model with only E, that's why it is not in outbox.

	E in M9		
11	A in M1, M2, M3, M4, M5, M6 AB in M2, M3, M4, M5 B in M7 AC in M6 ABC in M3, M4, M5 DE in M8 E in M9 ABCD in M3, M4 ED in M9 D in M8, M10	ABCD in M3 ABCD in M4 ED in M9 D in M10	D has fired again preconditions of M3, M4, M9 and M10. This is due to infinite time window. Note the list of sequences has not changed, only sequences ending by D moved at the end of the list (brown colour).
12	A in M1, M2, M3, M4, M5, M6 AC in M6 ABC in M3, M4, M5 DE in M8 E in M9 ABCD in M3, M4 ED in M9 D in M8, M10 AB in M2, M3, M4, M5 B in M7	AB in M2 B in M7	B fired again preconditions of M2 and M7. This is due to infinite time window. Note the list of sequences has not changed.
13	A in M1, M2, M3, M4, M5, M6 DE in M8 E in M9 ABCD in M3, M4 ED in M9 D in M8, M10 AB in M2, M3, M4, M5 B in M7 AC in M6 ABC in M3, M4, M5	AC in M6	C fired again precondition of M6. Note the list of sequences has not changed
14	A in M1, M2, M3, M4, M5, M6 DE in M8 E in M9 AB in M2, M3, M4, M5 B in M7 AC in M6 ABC in M3, M4, M5 ED in M9 ABCD in M3, M4 D in M8, M10	ED in M9 ABCD in M3 ABCD in M4 D in M10	D fired again preconditions of M9, M3, M4 and M10. This is due to infinite time window.
15	A in M1, M2, M3, M4, M5, M6 DE in M8 E in M9 AB in M2, M3, M4, M5 B in M7 AC in M6 ABC in M3, M4, M5 ED in M9 ABCD in M3, M4 D in M8, M10 ABCF in M5	ABCF in M5	A new rule model M5 is inserted in the list because of the incoming event F.

Table 5.6 – Infinite Time Window Results

Finite Time Window Execution

TimeWin=5 is set to limit matching to sequences of events in the window [*Now-TimeWin*, *Now*]. The *Clean Sequence* function removes the events outside this time window. Table 5.7 presents the new results. Red comments present the changes.

Time	List of Sequences	Outbox	Comments
1	E in M9		
2	E in M9		
3	E in M9		E of Time=3 replaced E of Time=1 so it has a new place in the time window.
4	E in M9		
5	E in M9 A in M1, M2, M3, M4, M5, M6	A in M1	Precondition of M1 fired by the first A. Sequence in light red is the new one.
6	E in M9 A in M1, M2, M3, M4, M5, M6	A in M1	Precondition of M1 fired by the second A. List of sequences not changed (normal).
7	E in M9 A in M1, M2, M3, M4, M5, M6 AB in M2, M3, M4, M5 B in M7	AB in M2 B in M7	B has fired preconditions of M2 and M7. Previous A with B give AB.
8	A in M1, M2, M3, M4, M5, M6 AB in M2, M3, M4, M5 B in M7 AC in M6 ABC in M3, M4, M5	AC in M6	C has fired precondition of M6. E in M9 has been removed
9	A in M1, M2, M3, M4, M5, M6 AB in M2, M3, M4, M5 B in M7 AC in M6 ABC in M3, M4, M5 ABCD in M3, M4 D in M8, M10	ABCD in M3 ABCD in M4 D in M10	D has fired preconditions of M3, M4 and M10. ED in M9 is not fired!
10	A in M1, M2, M3, M4, M5, M6 AB in M2, M3, M4, M5 B in M7 AC in M6 ABC in M3, M4, M5 ABCD in M3, M4 D in M8, M10 DE in M8 E in M9	DE in M8	E with previous D fired precondition of M8.
11	AB in M2, M3, M4, M5 B in M7 AC in M6 ABC in M3, M4, M5	ABCD in M3 ABCD in M4 ED in M9 D in M10	A in M1, M2, M3, M4, M5, M6 is removed but not AB, AC, ABC

	DE in M8 E in M9 ABCD in M3, M4 ED in M9 D in M8, M10		
12	AC in M6 ABC in M3, M4, M5 DE in M8 E in M9 ABCD in M3, M4 ED in M9 D in M8, M10 B in M7	B in M7	AB is removed in list. AB in M2 is not fired again.
13	DE in M8 E in M9 ABCD in M3, M4 ED in M9 D in M8, M10 B in M7		AC in M6 and ABC in M3, M4, M5 are removed. AC in M6 is not fired again!
14	DE in M8 E in M9 ABCD in M3, M4 B in M7 ED in M9 D in M8, M10	ED in M9 D in M10	ABCD in M3 and ABCD in M4 are not fired again.
15	ABCD in M3, M4 B in M7 ED in M9 D in M8, M10		ABCF in M5 is not fired with a time window of 5 but could be still fired with a time window =8

Table 5.7 – Finite Time Window Results

The use of the time window limits the repetition of already sent events. It can be a problem (case where M5 is not fired) or be normal (too late to understand M5 has happened because F has arrived late after ABC to compose ABCF). We deduce that time window should be tuned. It greatly depends on the meaning and context of rule models and thus on the precondition. It also depends on the use of an event in several rule models with very different meanings. In fact, cleaning of the sequences list must be learnt by experience or manually defined in order to clean some sequences of events after a rule model is fired. Incoming events, well recognized, i.e. not ignored, are kept as facts in the ontology of models so a first optimization of the algorithm could be that the list of sequences will be a specific database table with fact indexes.

A second optimization could be to insert sequence of events used in the precondition of a rule model in this table directly at the rule model creation like in Figure 5.35. *idrulemodel* is the rule model index in the list of nodes of the ontology of models. *idevent* is the index of the fact or model of event. *eventrank* is used for the order of events in the sequence indexed by *idrulemodel*. *status* tells if the event has been received. *teceivedate* is the date of reception of the event. We added *activationperiod* to manage the time window before deactivating the current sequence containing this event.

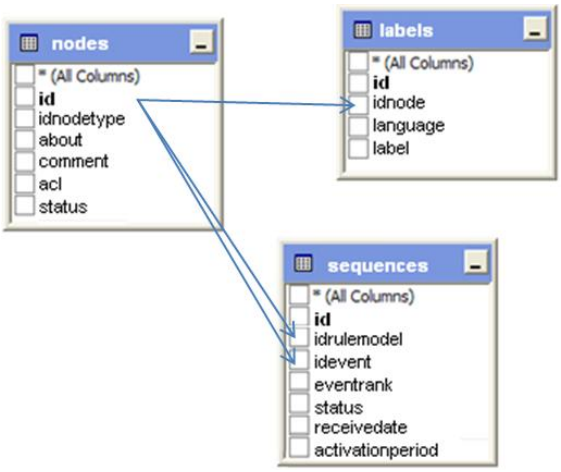


Figure 5.35 – Sequences table

In conclusion, precondition of a rule model with only one event is immediately fired and this event, once aggregated with event information in its roles, is sent to other agents and services. For a rule model with a precondition containing several events, the sending of the aggregated event (instance of the rule event) will be done when the last event of the sequence will be received.

5.5.4 Results

After running the CPN models several times on a period of 30 seconds, we obtain the results in Table 5.4.

Constraint	Result
Events Completeness	100%
Event arrivals order	100%
Temporal aspect	100%
Consistency	100%

Table 5.8 – Validation Results

We may explain these results by the fact that unknown events are ignored and the validation can't continue. If the event arrivals order is not respected, the engine doesn't wait and restarts again to the first event. The temporal validation allows validating the event arrivals order and the start time and end time is corresponding to the table in Figure 5.26. Consistency is respected because of the presence of the concept and model in our ontologies, and because no generated events is altered by the network transmission or by a limitation of the reception queue in this program. If it is not the case, the event is ignored and the matching process must keep on. Inconsistency doesn't mean there is no repetition of the same output event. We presented how to manage this problem.

Our inference engine is validated for all scenarios because, if the rule model changes, only the comparators of the schema changes. Only the arguments of the roles of the incoming events will be matched with the rule model precondition.

5.6 Comparative Study of Architecture

We performed a non-exhaustive comparison between our architecture and MAS architectures (Bayardo et al., 1997; Tambe et al., 2000; Esteva et al., 2004; Gil Iranzo, 2004; Heckmann 2005; Lech and Wienhofen, 2005; Lin and Hsu, 2006; Gateau, 2007; Mastrogiovanni et al., 2007; Bach, 2008; Alain et al., 2009; Billington et al., 2009; Kameas et al., 2009; Kapahnke et al., 2010; Subercase and Maret, 2010 & 2011) presented in Chapter 2. In the case of these semantic architectures, performance measurement in terms of quantitative and qualitative satisfaction is quite rare. These MAS architectures have some of our properties but not all (columns of Table 5.9). To justify our choice, we focused our appraisal on the following properties:

- *Architecture*: Agents may be part of - a distributed system where agents are autonomous and communicate through a computer network; - a collaborative system where autonomous agents are managed by a centralized server around a centric task; - a fixed architecture where agents are structured like functional parts of a computer program fixed by the developer. In addition, architecture and components of the architecture may be adaptable and reusable by using adaptation and composition mechanisms.
- *Pervasive or Ubiquitous Computing Architecture*: The architecture can be automatically extended using a discovery mechanism and current standards for webservices.
- *Multimodal Interaction*: The architecture is designed to manage multiple input/output modalities (audition, visual, textual, dialog) or devices.
- *Platform independent*: The architecture is independent of any operating systems and hardware platforms (robotics for example). Our components are autonomous pieces of software working in a computer network, not necessarily embedded in the system to control or manage.
- *Types of agent*: agents are distinguished between different independent properties like autonomy, knowledge and know-how (eventually offered as services), decision capacity.
    - Rational agents are based on rational behaviours using logical rules and a knowledge base. Reflection is done by a reasoning engine before acting. They try to maximize a utility function.
    - Cognitive agents are often rational agents but with designed cognitive functions similar to those in the human brain like short term memory, long term memory, human reasoning mechanisms. They are built by psychologists to check their assumptions on the study of human behaviours.
    - Belief-Desire-Intention agent (BDI) is another specific kind of rational agent where beliefs are inputs, desires are goals and intentions are planned actions. Belief, Desire, Intention are logical function to dissipate ambiguities in managed information during reasoning (Cohen and Levesque, 1990; Rao and Georgeff, 1991).
    - Programmed agents have a fixed program made by a developer. They make no choice but they can manipulate different types of data and work in synergy between each others.

- *Genericity of agents:* Agents genericity means agents are components able to implement any functions. For rational agents, these functions will be based on the content of their knowledge base but not in the modelling parts of the agent. Concepts and models taking place in the knowledge base define the agent and are independent of the implementation. In our case, the agent is a standard webservice with three generic modules: the communication module, the inference engine and the knowledge base (i.e. memory). The executed internal code of these agents is always the same and only data in memory change when modifying the agent's behaviour. It is also the case for BDI agents. Non-generic agents are built with different specific modules depending of the job they are designed to.
- *Agent Communication Language (ACL):* Declarative communication language used between agents, and between agents and other components to dialog and share concepts and ontologies. In our case, it is EKRL. There are now several XML, RDF, OWL languages for communication, interoperability and interconnection. Some became standards like, FIPA-ACL (Foundation for Intelligent Physical Agents), SWRL (Semantic Web Rule Language), RuleML (Rule Mark-up), KQML (Knowledge Query and Manipulation Language), KIF (Knowledge Interchange Format), etc. Other agents don't use ACL or use their own language.
- *Knowledge Base*: The agent program is contained in (internal) or connected to (external) one or more formal ontologies (memories in our case), meets the best case: distributed (independent to other software part) or the worst case: centralized (software dependent), or is connected to a blackboard. SWRL rules are also a static agent program defined as a database stored in an XML file. Some agents share their own knowledge base, which can be accessed by other software parts in the network; these agents become slow and dependent.
- *Static Base*: Indicates whether the knowledge base (KB) is static or can change in-line. Some KBs can be adapted at run time.
- *Concept Ontology*: This type of ontology is a standard domain ontology containing concept classes, instances, properties, and semantic relationships between them. Most existing ontologies are of this type.
- *Ontology of models*: We defined the ontology of event models as a predicates ontology applied to concepts or other models. Some properties in the concept ontology can do this, but their action field is limited to the class to which they are attached. Our predicates (frames of roles/arguments) are independent of concept classes, reducing the redundancy of a property. Our predicates are exactly like behavioural rules but are perfectly structured in our ontology through semantic relationships between models. Another important advantage is that we reduce search time and increase performance in our inference engine by reducing search space (subtrees of models classes and instances). When models are in regular domain ontology, the tree structure advantage is lost; many relationships appear, leading to an unreadable graph with absolutely no guarantee of closure.
- *Close to NL*: Predicate logic, like propositional logic with semantic relationships, is obviously closer to human language than any other

knowledge representation. Defining our EKRL, this way also enables us to reach a solution to natural language processing (NLP), scientific and linguistic researches.

- *Consistency checking*: Consistency checking is a mechanism for checking semantics and whether rule models are semantically consistent to avoid conflicts. Our ontologies contain only a single reference to a concept or a behavioural meaning; this important point ensures consistency of concepts, models and instances. Due to the formality of the reference model the component models can be checked formally and it can be proven which development relations and composition operations are compatible with each other. Consistency checking is thus intrinsic to the inference engine for an agent and to the design in multiple layers for the system. And it should be required for any other systems or architectures.
- *Model Based behaviours*: Agents or MAS contain behavioural models in their knowledge base that allow them to understand the behaviour of entities present in the environment including them.

The preferred properties of an architecture or agent are shown in bold type in Table 5.9 below. To summarize this comparison, it shows that we have developed a pervasive and multimodal MAS architecture that contains the most generic agents, the same semantic language for all cognitive agent operations (fact storage, event inference, communication), making it easy to model and understand behaviours. Our EKRL is a protocol containing words and values between tags; it is the content of a simplified natural language exchanged between agents that manipulate human concepts. Models of the EKRL language are narratives (agent knows and tells what's happening), performative, or imperative (agent send orders to be executed) and allow fast memory organization and storage of past facts in the ontology of models. In addition, a developer may program the agents directly with EKRL models in their memory. Agents will know what to do when facts come on-line. A final key point is that our architecture reasons on facts in memory, just like humans do, by truly manipulating concepts in natural language. Other agents are implemented only with logical statements. Other architectures are often a mixture of technologies using the same languages to communicate or exchange data with or without required wrappers. These architectures use reasoners for data mining, not for truly extracting meaning; so they are not easily adaptable. Notion of rational agency is not much found in the scientific literature except in a paper of Van der Hoek and Wooldridge (2003) using modal logic and dedicated to BDI agents.

References	Architecture	Pervasive Ubiquitous	Multi I/O Modalities	Platform Independant	Agent Type	Agents Genericity	ACL
INFOSLEUTH : [R.J. Bayardo et al 1997]	Distributed	Yes	No	Yes	Rational	No	KQML, KIF
Teamcore [Tambe et al. 2000]	Distributed	No	No	Yes	Programmed	No	Yes
AMELI [Esteva et al. 2004]	Distributed	Yes	No	Yes	Rational	Yes	FIPA
SWA [Gil Iranzo, 2004]	Distributed	Yes	Yes	Yes	Rational	No	FIPA-RDF
UbisWorld [Heckmann 2005]	Fixed Architecture	Yes	Yes	No	Programmed	No	No
AmbieAgents [Lech & wienhofen 2005]	Distributed	Yes	Yes	No	Programmed	No	Yes
iCare System [Lin & Hsu 2006]	Collaborative	Yes	Yes	Yes	Programmed	No	No
SYNAI agents [Gateau 2007]	Collaborative	Yes	No	No	Programmed	Yes, Moiseinst	No
[Mastrogiovanni 2007]	Collaborative	Yes	Yes	No	Cognitive	No	Yes
MicroPSI [Bach 2008]	Fixed Architecture	Yes	Yes	No	Cognitive	Yes	No
GEORAL [Alain et al. 2009]	Distributed	No	Yes	No	Rational	No	JADE
[Billington et al. 2009]	Fixed Architecture	No	Yes	No	Cognitive	No	No
[Kameas et al. 2009]	Distributed, Adaptable, Reusable	Yes	Yes	Yes	Rational	No	SWRL
ISReal [Kapahnke et al. 2010]	Distributed	Yes	No	No	BDI	Yes	Yes
SAM [Subercaze & Maret 2010]	Distributed, Reusable	Yes	No	Yes	Rational/prog.	Yes	JADE
Dourlens	Distributed, Adaptable, Reusable	Yes	Yes	Yes	Rational	Yes	EKRL

References	Knowledge Base	Static KB	Concept Ontology	Model Ontology	Close to NL	Consistency Checking	Model Based Behaviors
INFOSLEUTH : [R.J. Bayardo et al 1997]	Yes, External database	No	No	No	No	No	No
Teamcore [Tambe et al. 2000]	Yes, Blackboard	Yes	No	No	No	No	No
AMELI [Esteva et al. 2004]	No	No	No, DB	No	No	Yes	No
SWA [Gil Iranzo, 2004]	Yes, External	Yes	Yes, RDF & OWL	No	No	No	No
UbisWorld [Heckmann 2005]	Yes, Blackboard	No	Yes, OWL	No	Yes	yes	Yes
AmbieAgents [Lech & wienhofen 2005]	Yes, several BlackBoard	Yes	Yes	No	No	No	No
iCare System [Lin & Hsu 2006]	Yes, blackboard	No	Yes	Yes, Events	No	No	Yes
SYNAI agents [Gateau 2007]	No	Yes	Yes	No	No	Yes	Yes
[Mastrogiovanni 2007]	Yes, Agent local	Yes	Yes, PDDL	No	Yes	Yes	Yes
MicroPSI [Bach 2008]	Yes, internal STM & LTM	Yes	No	No	No	No	No
GEORAL [Alain et al. 2009]	No	Yes	Yes	No	Yes	No	No
[Billington et al. 2009]	Yes, Whiteboard	No	No	No	Yes	Yes	Yes
[Kameas et al. 2009]	Yes, global and local ontologies	No	Yes, RDF	No	No	No	No
ISReal [Kapahnke et al. 2010]	Yes, local and global ontologies	Yes	Yes	No	Yes	Yes	Yes
SAM [Subercaze & Maret 2010]	Yes, local SWRL	Yes	Yes, OWL DL+SWRL	No	No	Yes	No
Dourlens	Yes, Internal Agent Memory EKRL	No	Yes, OWL	Yes, EKRL	Yes	Yes	Yes

Table 5.9 – Comparison of Multi Agents Systems using semantics

5.7 Conclusion

This chapter presents the implementation of the SAMI framework containing a complete package of software and hardware: the semantic agents, editor of the memory of agents, TCP-IP communication module with its messages queue, inference engine API, ConnectDB API, Third party and robotics platforms, ERKL concentrator managing services and including EKRL wrapper, installation procedure, use procedure and scheduler. All these detailed parts have been realized and are fully operational.

The second part of this chapter was dedicated to the analysis of important performance measures of the architecture and components. In particular, we analyse the networking load, the knowledge base access time, inference time to evaluate the maximum load and events which can be processed by our architecture. We realized this validation on current machines and network. Despite the fact that measures are really excellent, to build ambient intelligence architecture, and in particular with EKRL protocol, it's clear that, in the future, these components will be much more fast and powerful. Happily, our architecture components are not limited to an operating system or a material. Even the EKRL concentrator can be replaced by a more powerful hardware board later, only the embedded I/O ports for hardly connected devices should be taken into consideration. From an internal point of view of knowledge base, we checked Consistency and Robustness of events sent to the system with attention, some of the results are obtained after implementation of uses cases of the next chapter. We check that in case of overload of events by second, consistency may decrease, and in case of non noisy events, robustness can be obviously decreased but stay at a correct and normal level. We also explored the cognitive workload and cognitive load measures of the NASA model, made by Hart and Staveland, they are subjective measures which can be adapted to our architecture for future work. We have studied the complexity of our inference engine algorithm during the storage and retrieval processes using the matching function. This is an algorithm fast in time and linearly dependant in space of *ra* table which drastically reduce space. This is an important result showing a big improvement compared to the curse of dimensionality and all large data management systems, mostly because of the knowledge base designed in an tree structure ontology, the concepts and models indexes aligned in only one database table (the *ra* table) and because of the name of concept pointing on a little sub tree of the ontology. Temporal validation of inference engine was made with formal coloured Petri nets on case a limited to "PutThatHere" scenario and on the general case of all possible scenarios represented by rule models in the ontology of models. We checked completeness, order of arrival, temporal constraint due to precondition of rule model and consistency. All constraints are perfectly well respected due to the conception of the knowledge base and the inference engine algorithm. To avoid too many event repetitions, we proposed to index and manage sequences of rule model preconditions in a specific database table. The time window will be managed by designer at the rule models insertion.

The third part of this chapter is a comparative study of numerous recent platforms, architectures and agents able to process semantically or rationally events in a network. The result is interesting and shows that there are some apparent similarities like, for example, distributed architecture. But, in reality, there are major differences and, in particular, interaction aspects are not implemented and human environment is not understood, the fusion and fission

processes are not encoded to extract meaning of the situation, inference engine is poor often limited to first order logic, temporal aspects are eluded and they focus on a single application. Moreover, our EKRL is much closer than natural language, and is used for communication and ACL, events storage and reasoning. Another point is the symbols grounding to the environment.

We propose as a future work to go further in the validation of all mechanisms presented and implemented in the previous chapter.

CHAPTER 6

USE CASES

« To understand is to be able to do »

André Gide

6.1 Introduction

6.1.1 Objective

After the introduction of the implementation procedure and development environment, our objective is to show the suitability and the fitness of our architecture and platform applied to different assistance applications like indoor daily living activities, healthcare, outdoor crossroad help. We highlight the interaction with multiple modalities and event models in different architecture with the implementation of fusion and fission processes. We bring functioning results. We want the architecture to avoid situation failures, ignored events, and random behaviours. The architecture and programming of agents depend on the conceptor and models inserted in the memory of agents . For failure tolerance, a simple scenario of adaptation can easily be integrated (in the form of query models) to replace a service that has broken down by another that is available.

6.1.2 Implementation Procedure

To build an application with our framework, several steps have to be followed. Procedure (Figure 6.1) is derived from system and software engineering but is close to knowledge engineering. As for any products, first step is the system definition with requirements and objectives clearly expressed in a technical agreement. Second step is the system analysis to refine and complete the specifications corresponding to requirements and objectives, and define the functions of the system and eventually the systems of the system. This work has to be done by a system or software engineer.

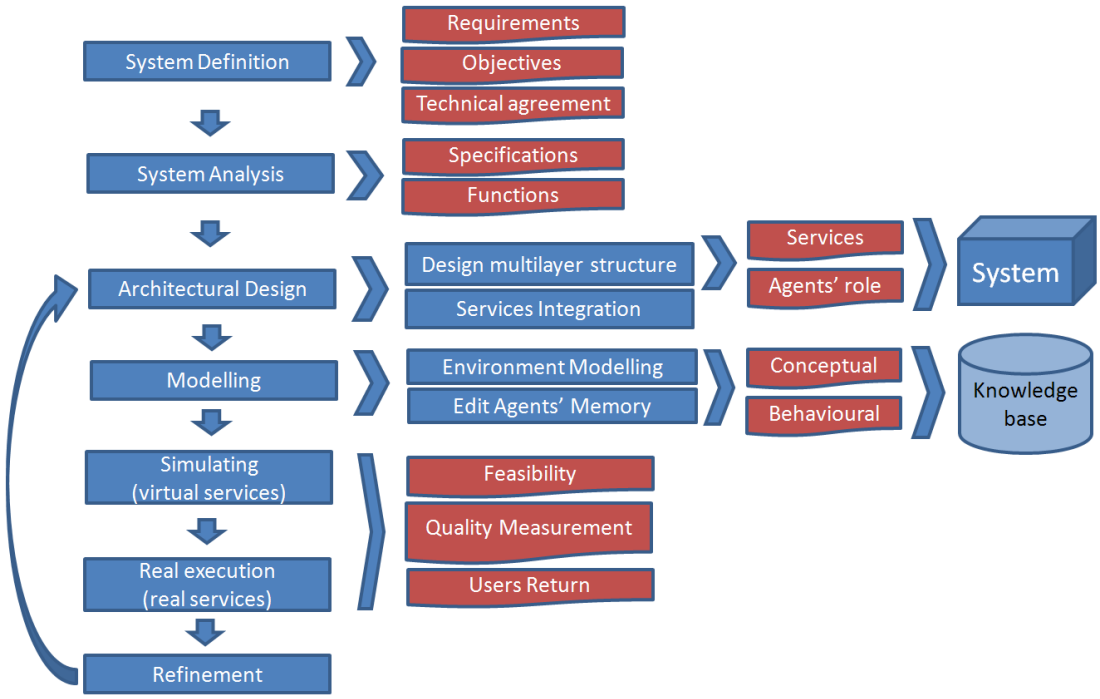


Figure 6.1 –Procedure

Third step is the Architectural design. Our framework helps to design interaction architecture of agents and services connected to and parts of the environment. You design the system with its inputs and outputs (services), and

its memory and logic (agents). Services can be chosen now or can be selected and composed online but at least functions of these services should be conceptualized in the next step to ensure a correct the detection and integration of these functions. We recall that services will be the functional parts of the system and agents will be the reasoning and knowledge parts of the system. Services can be created or reused, and third party web services can be connected. It is the time to size the architecture in terms of number of virtual layers and number of agents by layer. We proposed in last chapters to consider layers of agents like abstraction levels of the meaning of information passing through the agents. And, in addition, to consider the first half layers as the fusion process of the interaction and the other half as the fission process of the interaction (See Figures 4.2 to 4.4). According to the physical processing power and memory load required, you can use a single agent with all necessary knowledge in its memory to a large set of simple agents. Agents will be modified at the next step. The architecture is scalable and replicable.

Fourth step is very important to realize the conceptual and behaviour modelling of the system and the environment where the system will work. This work can be realized by knowledge engineer together with system domain experts. All necessary concepts (entities, words, values, scales) should be entered in the ontology of concepts of the agent's memory using the memory editor. Existing agents can be easily cloned and reused by copying the database of the agent to another. Conceptual knowledge of agents can be easily reused by importing and exporting concepts from the ontology of concepts. Models (event models, rule models, query models and instances) can be created, edited and modified in the ontology of models with the memory editor. Fusion Agents must be used to extract meaning of situation and fission agents to select and generate actions or plans. It is important to keep the situational meaning or global context between the fusion and fission processes. Granularity refers to the degree of resolution or abstraction of the structure and behaviour of the system. It depends on models and concepts of the task or problem to solve. For a compositional system, it is determined by the granularity of systems, which may be more fine-grained than required. Functional parts are realized by services and representation by models in the memory of agents. The granularity of behaviour descriptions is determined by the purpose to achieve. EKRL models allow all possible designs. Representation of the system in a hierarchical structure and at multiple levels of abstraction allows comprehending subsystems at a higher level.

Fifth step is interesting to check your architecture, a service, an agent, a set of services or a set of agents. The website interface of the EKRL concentrator allow creating virtual sensors and actuators to generate events, capture resulting events and then monitor activities of services and agents. Real sensors and actuators can also be connected to the EKRL concentrator to test a part of the architecture. Different quality measurements on these data can be implemented in order to evaluate scenarios, user preferences or comfort, check connections and adaptation, or manually refine of the architecture. Quality of services and quality measurements in general is not part of this thesis more dedicated to knowledge representation, understanding and reasoning with semantic agents but it can easily be implemented by using existing webservices for quality measurements.

Sixth step is the execution of the framework in the real environment like a room or a house (indoor assistance, health care, and safety), a street or an itinerary

(traffic analysis), a city (smart city projects) and so on. Network and web services discovery should be adapted to the application giving the list of available network addresses or services. Composition and selection agents and service can be used. We suggest the designer to use an EKRL concentrator by room or by street.

Seventh step is useful to take account of simulation and execution reports to refine the architectural design at step 3 and the environment modelling at step 4.

This general procedure can be adapted to different domains of engineering to develop intelligent products requiring reasoning and understanding of the environment. Moreover, this procedure can serve to build new cognitive and problem solving architecture by developing multilevel functions of human psychology like memory and learning like mechanisms presented in Chapter 4.

6.1.3 Development Environment

In the applications presented in the next sections, we used several robotics platforms (NAO and Spykee robots), Input modalities (2 MS Kinect) Microsoft Robotics Studio (input/output services and 3D simulation), Aldebaran NaoQi.NET (Nao service), MS Kinect SDK, MS Visual Studio 2010, .NET4 (Multicores agents, Editor of agents' memory), JADE (Multi Agents System), CPNTOOLS, MySQL (Storage in the agent's memory), Roboard, Embedded Linux Debian, Apache 2 and PHP5, gcc, IPTable (Firewall). We also used the EKRL concentrator presented in section 5.2. The test environment is composed of a network with a room containing objects. We project a simulation on the wall of the room. Robots are directly to agents but are not containing agents in their system. We prefer considering robot parts as hardware services.

6.2 Daily Activities

6.2.1 Context

As a proof of concept, in this section we look at two simple scenarios using the architecture of semantic agents and services working together in an agency: a monitoring scenario and an assistance scenario showing fusion and fission agents in action. Activity recognition and monitoring use multiple sensors like solution of (Maurer et al., 2006) on body positions. Robots contain agents interconnected to the network. The objective of the application is to provide house security through a smart monitoring or alarm system, human safety, health care assistance for old or disabled persons, personal comfort with the simplification of human tasks, and companionship for lonely people or children by reading, playing, and conversing.

Jack is a human being alone at home. His house is composed of walls, doors, windows, and intelligent equipment connected in a domotic network. The robots and house contain semantic agents and services as defined in Chapter 3. Robots are able to communicate with all sensors and actuators, such as video cameras and vocal synthesis, and perform face and object recognition. Nao acts as a companion, Spykee as a security guard, and Roomba as a vacuum cleaner for Jack's housekeeping.

Hardware	Services	Event Models
Video Camera Sensor	Object Recognition, Entity Recognition (House, Spykee)	Exist:Object, Exist:Human, Move:Human
Speakers	Audio Emission (Nao)	Behave:Play, Behave:Speak
Mic Sensors	Vocal Recognition, Sound Direction (Nao)	Exist:Entities, Move:Entities, Behave:HumanTalk, Exist:SoundLevel
Neck Actuator	Move Head (Nao)	Behave:Awareness, Behave:Look
Hand	Grasp Object (Nao)	Behave:Grasp
Legs	Walk (Nao)	Move:Walk
Wheels	Roll (Spykee)	Move:Navigate
Human Detector	Human Location (House)	Move:Human
Vacuum cleaner	Clean, Battery (Roomba)	Behave:Clean Room, Behave:Recharge, Exist:Battery Level
UDDI .OWL-S Server	UDDI (input/output service), WSExecute (output)	Exist:AvailableServices, Exist:AvailableAgents, Behave:ExecuteService

Table 6.1 – Events of Services

Table 6.1 summarizes the architectural design and modelling steps of the implementation procedure. It shows relationships between hardware parts (sensors and actuators), service parts (drivers), and event models (representation in memory) used in our scenarios. Memory of agents has a set of event models to recognize or use other agents and services on the network. Many events and concepts relative to house objects and human activities have been entered in the memory of agents.

6.2.2 Monitoring Scenario

To monitor and evaluate daily living activities, our agent's memory contains all the facts (event instances) under the "*Behave:Living*" predicate model. Depending on the period of time required or the granularity of the scenario, the matching process done by the inference engine under a query will send a collection of facts to the agent. These are explicit situations of what has happened.

A list of stored event models and the compliance of the scenario in time allow is to put together a day by day vision of human activities (Figure 6.2). These measurements are taken by monitoring the real environment. It is useful to monitor usual or abnormal contexts. The robot or manager agent for this task will be able to make a decision following the degradation or improvement of the patient's state.

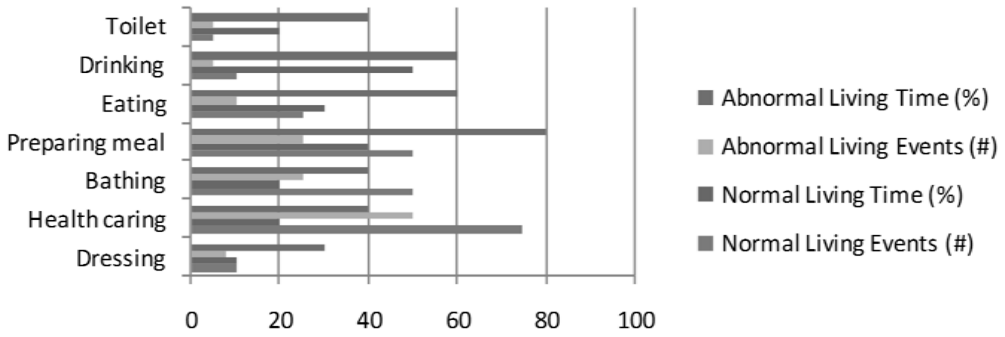


Figure 6.2 – Activities events recognition at home

Behave: Sleep SUBJECT: Jack SENDER: Human Detector DATE1: 25/04/2010 21:09:05 DATE2: 25/04/2010 17:25:14 LOCATION: double bedroom	Exist: Sound Level CONTENT: Very Weak (10 %) DATE1: 25/04/2010 17:25:26 LOCATION: double bedroom

Figure 6.3 – "Behave:Sleep" and "Exist:Sound Level" abnormal facts

Behave: Call
SUBJECT: Jack (Human, 86 years old)
SENDER: Nao
CONTENT: Please, Send help to this address "house address"
RECIPIENT: COORD(Vocal Synthesizer, Phone Service)
DATE1: 25/04/2010 15:26:00
MODIFIER: Emergency

Figure 6.4 – "Behave:Call" order

It is also used as an alarm to trigger an emergency call, checking different activity time periods. In this scenario (Figure 6.3), Jack is sleeping too much (sleep time is abnormal), with low breathing noise (abnormal situation too). The agent will fuse "*Behave:Sleep*" and "*Exist:Sound Level*" predicates into a "*Behave:Call*" predicate. It is a critical situation like this that requires action (Figure 6.4). Lots of monitoring cases can be programmed too. The architecture is also used to anticipate Jack's needs (alarm clock setting, food orders, and so on). All these event models and facts are stored as is in the memory of agents.

6.2.3 Assistance Scenario

Jack moves in the home and Nao helps him avoid obstacles. Nao agents can discover all available agents and services with the two EKRL messages "*Exist:Available Services*" and "*Exist:Available Agents*". They will check all available modalities to interact with Jack but they do not need to know which sensors are sending messages, because each time something is detected by a sensor, an event is sent to all other agents (Figures 6.5 and 6.6).

Move:Walk
SUBJECT: Jack (instance of human being)
POSITION: Between Sofa and TV (x=25,y=10)
DIRECTION: Left (d=45°)
SPEED: Slow (s=1m/s)
SENDER: GestureDetectionVideo1
DATE: 09/01/2010 10:30:10
LOCATION: drawing room (r=3)

Figure 6.5 – "Move:Walk" fact

Exist:Existing Objects
SUBJECT: Table (instance of furnitures)
POSITION: Between Sofa and TV (x=27,y=12)
SENDER: GestureDetectionVideo2
DATE: 09/01/2010 10:30:11
LOCATION: drawing room (r=3)

Figure 6.6 – "Exist:ExistingObjects" fact

Nao agents focus on video camera sensors sending events concerning Jack (path, location, objects and potential obstacles). Nao understands Jack will hit the table. With its reasoner, Nao chooses the best modality to prevent Jack doing so (Figure 6.7).

Behave:Speak
SUBJECT: Obstacle Warning
CONTENT: "Be careful, Jack, you will hit the table!"
SENDER: NaoRobot1 (instance of robots)
DATE: 09/01/2010 10:30:11
LOCATION: drawing room (r=3)

Figure 6.7 – "Behave:Speak" order

Nao send orders to several output modalities (output services) depending on the reasoner rules and by making sure the room is not too noisy (using any other multimodal sources). Jack now knows about the obstacle and can avoid it. Nao continues to be aware of all events happening in the environment.

6.3 Healthcare

6.3.1 Ederly Falling Prevention

In community dwelling (assisted living/nursing homes) elderly people, falls and fall-related injuries are common and appear to be independent determinants of functional decline (Figure 6.8). At least 30% of people over the age of 65 years old fall each year, and this proportion increases to 40% after the age of 75. These resulting functional limitations can be used to significantly predict costs related to physician visits, hospitalizations, mortality, and nursing home admissions. Falls, and even the fear of falling, could also affect health-related quality of life. Because muscle weakness and impaired balance are associated with an increased risk of falls in the elderly an intervention to prevent these conditions would reduce the frequency of falls. At home, lots of services are available, detectors in the doors, coffee maker in the kitchen, drink distributor in the saloon, human detectors in the rooms and corridors. Once connected to the home area network (HAN), our system is able to use all services according to their

availability (Benta et al., 2009). An example of context-aware monitoring is in (Mileo et al., 2010).

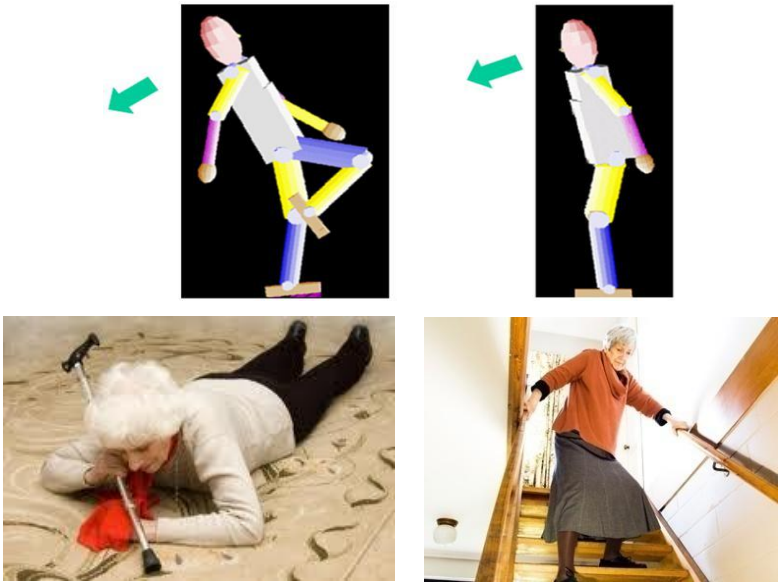


Figure 6.8 – Ederly falls

Management agents of the system discover services and compose a new architecture depending on the application or system objective. All actuators, devices and sensors are fully described in the ontology of concepts of the agents under the "*entities:parts*" class. It is the same for the walls and furniture positions in the house. To know where the human, animal or any entities are, agents may use the following models to query their memory: "*Exist:Existing Entities*" and "*Exist:Locate Entity*". To achieve scheduled tasks, robot agents will read planning in their memory and compose or adapt the architecture with available resources of the system then acts. Triggered tasks are also taken into account by receiving an emergency event from the fusion agents following some priorities. Figures 6.9 and 6.10 represent the situation where someone is falling down in the living room. Management agents delay all current tasks and adapt the architecture for the system (house systems or mobile robot) to hold, retain or catch the one if possible. Else it will alert the rescue from a phone call.

Name: Move: Someone Falls Down
Father: Move:Move Someone
Natural language description: 'James is falling down'
MOVE
SUBJECT: *"fall" (possible moves detected by the camera)*
SENDER: *MoveDetectionVideo5 (instance of sensors)*
SOURCE: *James (instance of human_being)*
MODALITY: COORD(*Behave:Adapt, Move, Behave:Catch Someone)*
TOPIC: *Monitoring*
CONTEXT: *House Activities*
LOCATION: *Living room (instance of Building)*
MODIFIER: *forced (emergency)*
DATE-1: 10/10/2010 10:21:59

Figure 6.9 – People falls down detection implies robot assistance

Name: Behave: Catch Someone	
Father: Behave:Assist	
Natural language description: 'Robot is catching James'	
MOVE	
SOURCE:	*Robot (instance of robots)*
BENEFICIARY:	*James (instance of human_being)*
MODALITY:	COORD(*Left Arm,Left Hand,Right Arm,Right Hand)*
TOPIC:	*Assistance*
MODIFIER:	*forced (emergency)*
DATE-1:	10/10/2010 10:22:01

Figure 6.10 – Robot has helped James

Robot was doing something else, agents receive the "*Move:Someone Falls Down*" event coming from the *MoveDetectionVideo5* service located in the living room (Figure 6.9). As it's a case of emergency and the modality "*Behave:Adapt*" is defined in this event, management agents organize the change by ending events to planning agent and compose architecture to move the robot near the human being called James and to catch him before James hurts the ground. All input modalities stay activated but output modalities are chosen. Fission agents in charge of the coordination of moves control the robot wheels and the rotation actuators to put arms and hands in the right position. During the fall, agents will modify the trajectory of robot arms. To obtain the right position of the body of the human and the speed of fall to calculate the appropriate position of the robot, fission agents can send queries to *MoveDetectionVideo5* service. After the reply; fission agents control the actuators to achieve the goal. The scenario will end when *MoveDetectionVideo5* service detects a secure position of the human being. Lots of scenario can be integrated by the addition of our predicate models: More simple scenarios like to recognize a vocal order from the human and executing it, to send an event to actuators to close the shutters at night or call the police on intrusion detection. Or more complex scenarios like to dialog with the human, to learn gestures or actions by imitating human, and realize complex behaviours.

6.3.2 Sport Activities

Motivation and monitoring of sport activities are necessary to take care of the people health. Exactly as the indoor scenario (Section 6.2.2), we propose to monitor the John during sport activities. Our memory editor allows writing event models and event instances following a hierarchy of different levels of abstraction. Figure 6.11 shows the ontology of models of the memory of an agent. *Behave* and *Move* are two root predicates. We observe *Gym* event model is a subclass of *Practice Sport* event model, and is composed of *Jump* and *Walk* event models which are subclasses of Move.

Models are basic events (*Jump*,*Walk*), composite events or scenarii (*Gym*) and query event (to find events in memory) as shown on Figure 6.12. This example is to show the simplicity of the implementation and how it is easy to manage multilevel activities in the ontology of models.

In addition, it is possible to use *Gym* as a query on the figure. It can also be a rule model to generate *Gym* facts. For this purpose, we could add a more complex precondition on the coordination of *Walk* and *Jump* event models based on *Date1*, *Date2*, a specific instance of *Subject*, and any other arguments in roles of the "*Behave:Gym*" predicate.

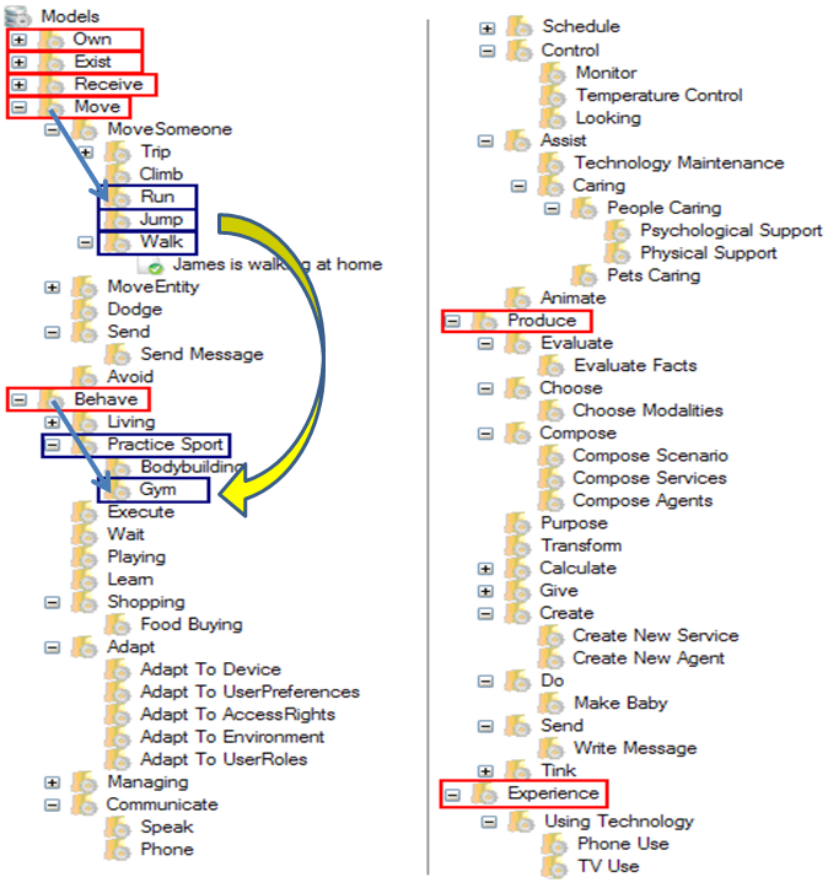


Figure 6.11 – "Behave:Gym" composite model

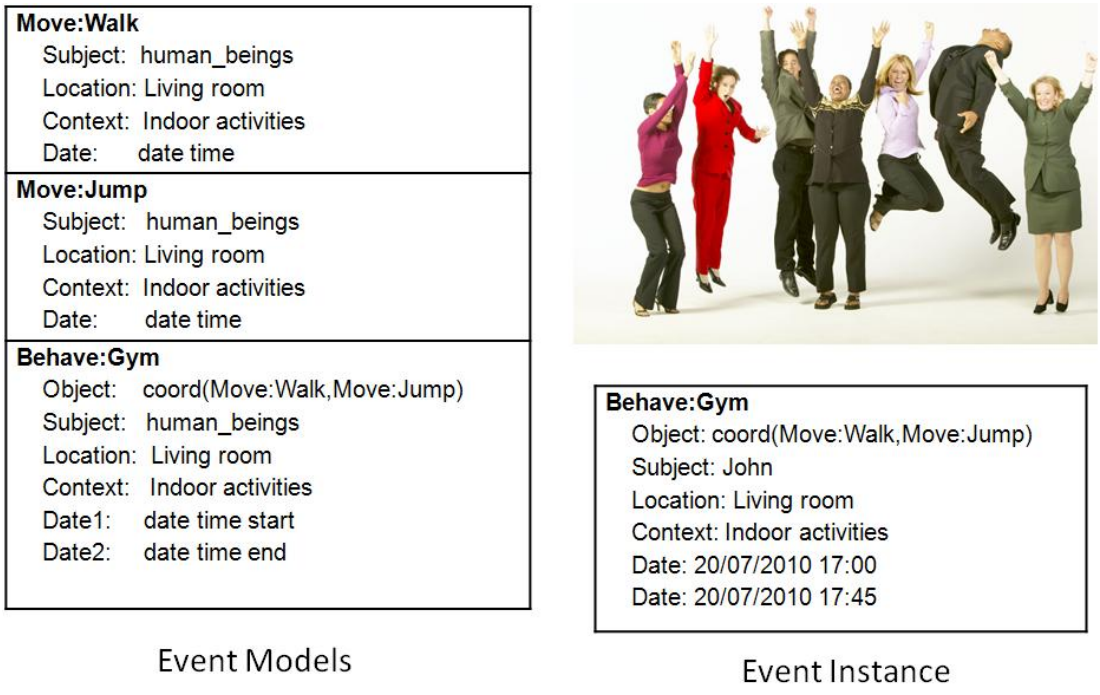


Figure 6.12 – "Behave:Gym" event and fact

6.4 Crossroad Assistance

In this section, we present how our semantic architecture of agents can help the pedestrian to cross a road.

6.4.1 Context

Assistant robot helps people to cross a road by checking the colour of red light, presence of car on the road, position and speed of that car. The objective of our application is to keep a valid user aware of the situation or to protect disabled, blind or old people by making them avoid crossing in situations of danger. We will focus on people crossing the roads and to avoid being a victim of traffic. We use our multimodal architecture to solve this type of problem by applying it to only one crossroad. Obviously, some more actions could be performed, like the robots could assist disabled persons, control the pedestrian signal or car signal (keeping it "red" until arrival to the other side), make an emergency call in the case of someone wants to inform authorities about an accident.

Light color	No car present	Car stopped	Car not stopped	
		Speed null	Speed decreases	Speed increase
Green	**Wait**	**Wait**	**Wait**	**Wait**
Orange	**Wait**	**Wait**	**Wait**	**Wait**
Red	**Walk**	**Walk**	**Wait**	**Wait**

Table 6.2 – Pedestrians are waiting on the sidewalk

Light color	No car present	Car stopped	Car not stopped	
		Speed null	Speed decreases	Speed increase
Any color	**Walk**	**Walk**	**Run if possible**	**Run if possible**

Table 6.3 – Pedestrians are crossing

Tables 6.2 and 6.3 present possible situations and required actions in order to protect pedestrians. Table 6.2 shows the case where pedestrians are waiting on the sidewalk and Table 6.3 shows the case where pedestrians are going across the road. For each situation evaluation of danger (matching of events), robot acts by speaking to humans for example.

6.4.2 Architecture

System architecture contains 3 layers: Hardware, Software Services and Software Agents. Hardware layer consists of sensors and actuators into the robot or from a network around our robot. Our Nao robot's body has 1 head with four mikes and two video cameras, 1 neck, 2 legs, 2 arms and 2 hands for a total of 25 DOF. It can listen, speak, walk and connect a TCP-IP network. Many complementary tasks could be developed later. We added in the environment a

MS Kinect to recognize human gestures and moves on the sidewalk. Software layer is a suitable composition of fusion and fission agents (internal or external mind) and embodied services. Agents will deal with input and output events to finally manage awareness and hardware controllers (Figure 6.13).

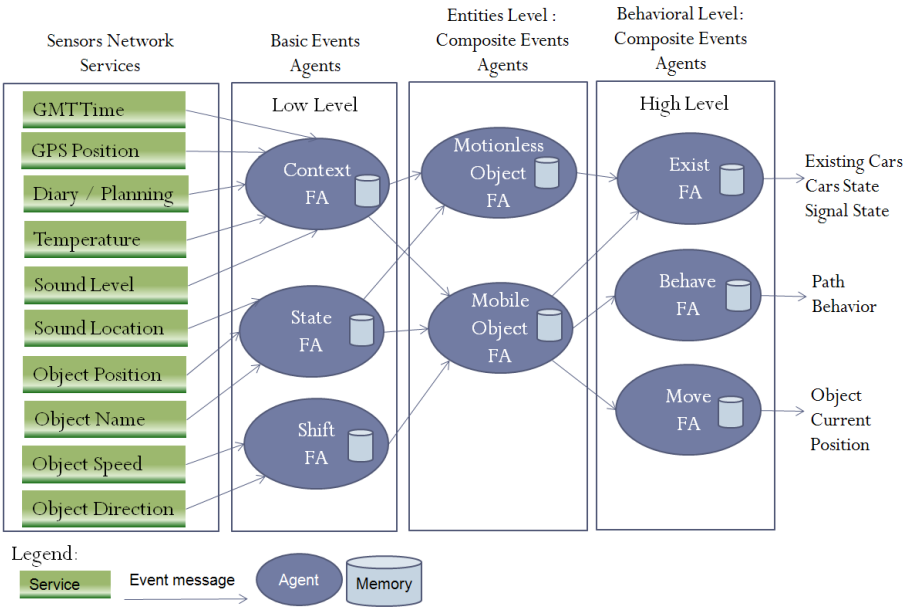


Figure 6.13 – Fusion Agents dedicated to Robot Awareness

Object State agent, in charge of object detection, composes the following event from sensors events. Still Object fusion agent will receive and store this event concerning the signal. These events will be composed by next agents to start evaluate events concerning cars and signal. Figures 6.14, 6.16 and 6.16 are composite events made by fusion agents.

Exist: Signal Color
OBJECT: Pedestrian Signal4
VALUE: Green Color=1
ACCURACY: 100%
DATE: 10/05/2010 10:05
LOCATION: COORD(Hoche Street,5th Avenue)

Figure 6.14 – Composite event "Exist:Signal Color"

Move: MoveObject
OBJECT: FourFourCar1
VALUE: Speed=20 km/h (deceleration)
ACCURACY: 95%
DATE: 10/05/2010 10:05
LOCATION: Hoche Street

Figure 6.15 – Composite event "Move:Object"

Move:Walk
OBJECT: Human
SENDER: MobileObjectFA
DATE: 10/05/2010 10:05
LOCATION: Hoche Street, crossroad4

Figure 6.16 – Composite event "Move:Walk" made by fusion agents

Figure 6.17 shows a composition of fission agents to act on hardware layer. Depending of input events, objective is to prevent user to across a road when danger is present. To achieve this goal, a set of actuators are available. Figure 6.18 presents an example of output orders sent from Control Agent to a robot control service (*LeftArmMotor*) to move the left arm up to the 45° position. Figure 6.19 is an example of order to speak sent to the VoicalSynthesis1 service embedded in a robot or elsewhere (speaker set in a wall or on the red light).

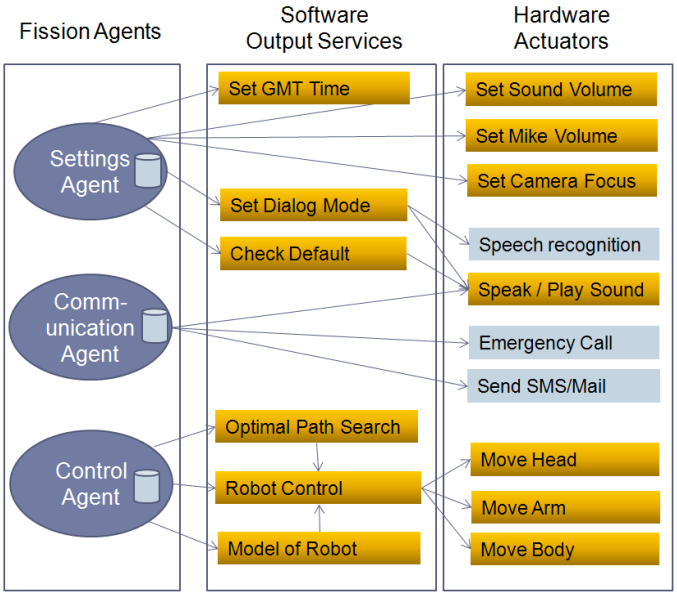


Figure 6.17 – Fission

Move:MoveArm
OBJECT: NaoRobot1
VALUE: 45°
SOURCE: Control Agent
RECIPIENT: LeftArmMotor
DATE: 25/04/2010 11:05:56

Figure 6.18 – "Move:MoveArm" event instance

Behave:Speak
SOURCE: Coord(GreenColor,CarStop)
CONTENT: "You can walk"
SENDER: Communication Agent
RECIPIENT: VocalSynthesis1
DATE: 25/04/2010 11:05:56

Figure 6.19 – "Behave:Speak" event instance

Figure 6.20 – Robotics Studio Simulation (by SimplySim)

Figure 6.21 – Virtual Reality Projection on the wall of the virtual room

Figure 6.22 – Car and Red light Services view in 3D simulation

Car and Red Light are simulated web services into a MS Robotics Studio simulation (Figure 6.20) and projected on a wall in a virtual reality room (Figure 6.21). They send event about their own actions (car moves, car stops, light colour changes), car speed (0 to 50 km/h) and colour of light (green, orange, red) (Figure 6.22).

6.4.3 Simulation

This simulation is done in a virtual reality room with the simulation projected on a wall. Humans (one or two people) and Nao robot are real. A Kinect sends high level events of human gestures and postures to agents. When the humans are close to the wall of video-projection (less than 2 meters from the wall), we consider them on the pedestrian crossing else we consider them on the sidewalk (more than 2 meters from the wall). Cars and Red Light parts are web services which sending events about themselves to agents. The car goes around the block with different speeds so that the arrival is random. Sometimes it stops at the red light, sometimes not, making it unpredictable. When car's position is higher than 50 meters from the crossroad, we consider it is a situation where no car is present. Hardware and software parts of the robot are controlled by services connected to agents and waiting for orders. In case of possible injury, robot will speak or play a sound to warn users by triggering an output event. We are now able to run our experimentations. We tried all situations of Tables 6.2 and 6.3 in real.

6.4.4 Results

Most of knowledge about cars, signal and behavioural scenario were stored in the memory of agents. Evaluation of the situation consists to query this memory. Agents evaluate the meaning of the situation with its past recorded events. Different situations summarized in Tables 6.4 and 6.5 have been simulated with 100 tries. We have also tested these cases with several cars.

Light color	No car present	Car stopped	Car not stopped	
		Speed null	Speed decreases	Speed increase
Any color	**100%**	**100%**	**99%**	**98%**

Table 6.4 – Pedestrians are crossing with one car

Light color	No car present	First car stopped	Car not stopped	
		Speed null	Speed decreases	Speed increase
Any color	**100%**	**100%**	**86%**	**79%**

Table 6.5 – Pedestrians are crossing with several cars

In the case "Pedestrians are waiting on the sidewalk" of table I, results are 100% successful. Other results for the case "Pedestrians are crossing" with one car and

with several cars are presented in tables III and IV. And as it was expected, cars speed determines the dangerousness especially when pedestrians are on the walkway. In this clean environment (no sunlight, no other objects moving) and after a short time of Microsoft Kinect calibration, with sometimes some erroneous gesture recognition events, results are: - at slow speeds 100% of the cases are successful; - at fast speeds with one car, we have 98% of cases are successful; - at fast speeds and several cars, 79% of cases are successful. These problems are due to the short distance between two cars that may occur like in true life when people drive when the orange light is on or don't see people are crossing. If people are not on the pedestrian crossing, it is always possible to avoid danger but once on the pedestrian crossing, in most of the cases, it is impossible to avoid the accident. To the event that vocally allows pedestrians to go cross the road, we added an event model that checks if more than one car comes and if the first car stops at the red light. With this new event, we succeed 100% of any cases.

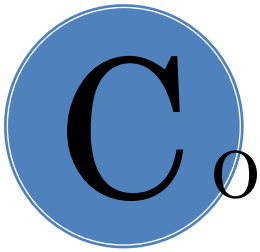
6.5 Conclusion

After an introduction to the implementation procedure to use our platform in development, we presented our architecture with multi levels of abstraction based on semantic agents suitable for interaction in the human environments like home and city. Our architecture brings a software part of the solution and manages multiples input and output modalities for interaction. A fast evaluation of possible scenarios is done by understanding and taking into account of all past events and behaviours. It may be easily adapted to several different tasks and contexts by adding appropriate concepts and models in the memory. We have proved it works well for human assistance in virtual reality simulation (in a dedicated room or at home) but, in real world, we will certainly need more reliable services: better video camera managing sunlight, real-time communicative devices embedded in cars and in red lights. Future work will be the auto-reconfiguration of the architecture depending on other situational contexts and user preferences profiles. Moreover our architecture may be extended to large network and agents can be located to multiple computers.

Many other scenarios can be considered like: - to travel in city taking account of the traffic, of the opening hours of the shops, post, and gym locations; - to travel in train or plane taking account of delay and cancelling; and so on.

We hope this contribution can help the future projects of smart homes (large scale events), smart cities (very large scale events), medical healthcare and sport motivation and various human assistance applications.

CHAPTER 7



CONCLUSION

« Whoever wants to reach perfection wants to walk on the horizon » Paul Carvel

7.1 Contribution

We have modelled and implemented our architecture of semantic agents and services to solve interaction problems in pervasive context. This architecture is embedded in a multi-agent system modelling technique. For this purpose we have modeled the environment using a knowledge representation and communication language (EKRL, Ontology). The obtained semantic environment model is used in two main semantic inference processes: fusion and fission of events at different levels of abstraction. They are considered as two context-aware operations. The overall architecture of fusion and fission is validated under our framework SAMI (agents, services, EKRL concentrator). We have developed this framework in order to design and execute this interaction architecture and to develop different performance analysis on some use cases such as monitoring and assistance in daily activities at home and on the town. We have seen that all behavioural scenarios are fully operational and simple to implement. The agent's memory is very fast and enables these agents to really store, communicate and reason about the environment. Discovery and communication between agents and services is effective. Performance depends on the system and sensor hardware. Good detection depends on sensor accuracy and the fusion agent's ability (event and rule models) to build higher abstraction level events. In our case, it is good enough, as shown in the daily living tasks examples. The decision to execute orders or compose events is invoked by scenario recognition (events list of scenario models). Response time depends on the activity but is adequate because it is much faster than a human's in a simplified context related to the application. This work is a step forward in the design of Multi-agent systems for managing multimodal interaction and Human System Interaction, and for modelling intelligent applications that require understanding and decisional aid in using our learning models.

7.2 Synthesis

This thesis is a subtle mixture of software engineering, architecture design, knowledge representation and reasoning. Distributed semantic agents and services reinforce the interaction managing multiple modalities. Each domain brings a stone to our architecture to solve the interaction issues. Ontology has been used to give conceptualizations of environment knowledge and to make explicit and machine-understandable the meaning of the natural language adopted as a common communication, storage and reasoning language. Semantic Web, with standard web services, exploits ontologies for providing resources with semantically meaningful information.

The main point of this thesis is the understanding of the environment and the extraction, creation and exploitation of the meaning of what is happening. Multimodal fusion solves and provides effective and advanced human-computer interaction by using complementary or redundant modalities. Multimodal fusion helps to provide more informative, exact, complete, reliable interpretation. Dependency between modalities allows reciprocal disambiguation and improves recognition in the interpretation of the scene. Our architecture is able to reason, learn, compose, cooperate, adapt and extend because the meaning is integrated in our semantic agents. Our approach is model-based. The models can be events to store facts, query to retrieve facts and rules to recursively program the agent.

In a very complex environment like the human one and with so much multimodal events to process, it is easy to overload them in terms of memory and processor resources. There are several choices of architecture with such agents but we propose a multilayer architecture to manage multiple levels of abstraction of knowledge. All messages and stored facts are built in EKRL. This representation language permits a disambiguation of the events with the ontology terms and to avoid ontology mapping. We developed an inference engine able to read and generate EKRL messages. The main reason of this new development is the lack of reasoners able to manage high order logic EKRL events. By storing and querying the agent's memory, inference engine extracts meanings of fusion and fission processes. Situation refinement is made by an aggregation function based on multi modal logics inserted into the roles-arguments formula. We highlight EKRL, close to natural language, which can be used as a common language to all interaction operations.

To fulfil the requirements, we design generic components; we connect them together to model the system managing, reinforcing interaction and the behaviours of entities in the environment. We applied the architecture on different interaction mechanisms like modalities selection, dialog, coordination, cooperation and learning. We take care of multiple contexts like goals, situational, dialog, user, execution, location and cooperation constraining the system. The scope of the application can be managed using scales. Next section presents the fulfilled requirements. Qualitative and quantitative analysis is simple.

In a pervasive context, discovery, composition, orchestration and evaluation of web services and agents can be realized autonomously based on EKRL events. We rely on OLW-S, SOAP and REST protocols. We present our EKRL concentrator for the interoperability in ubiquitous network or ambient intelligence. We measure some performance of the agent and the networking architecture. We compare with other semantic agent approaches. To verify the feasibility of our design and as proofs of concept, we presented several use cases interesting in assistance applications to aid the user at home or outside with all implied constraints. We present simulations and real applications realized with Nao robot.

In conclusion, we may add that our study has merged different scientific domains in order to produce a new semantic architecture for the interaction problem. The hierarchical management of information, events, behaviours, activities and contexts allows a continuous processing of the environment without requiring a-priori discretization of the state and action spaces. Our architecture is designed as a control loop allowing building real-time applications. The complexity is distributed on several layers of agents. Number of layers depends on the degrees of abstraction required by the application to design. We provide a powerful development platform where no coding is necessary to program the system. Strict optimality is not crucial but we propose algorithms to ensure optimal policies. Very few architecture globally integrates symbolic information representing the environment to realize all operations required for the interaction.

7.3 Fulfilled Requirements

We present in Table 7.1 our proposed solution to the requirements of Chapter 3. Environment Modelling has been deeply explained in Chapter 3 and proved to be efficient in following chapters. Uncertainty management is at different levels: For events, consistency of the knowledge base, structure of the ontology of models and rule models condition controls the event recognition. For fact contents, in role of predicate frame, concepts and values are verified with the ontology of concepts and scales, a confidence field of the database can be used by our inference engine. Moreover, Hadjiski et al. (2010) proposes to manage uncertain systems by adaptation of fuzzy ontologies; we think some common points like our definition of scales may be complementary. Multiview requirements are realized by the last layer of fusion agents and the first layer of fission agents in the architecture; they continually maintain the global context X in memory. This allows control and monitoring of sub contexts by querying these agents. Modality and reusability are possible by the use of composition and selection mechanisms, and structurally because our components are web services. Communication between components is made by EKRL on the network; other components or services should be connected to a driver embedded in the EKRL concentrator to be able to join the semantic network of the architecture. Integration is similar to modularity and communication requirements in our case. Complexity and heterogeneity requirements are complex which depend on the application and the systems of the environment. Modelling complexity of knowledge, agent inference engine, external components and architecture is possible in ontology and in EKRL. Optimal policies and complex scenarios can also be learned. Heterogeneity management managed by a webservice composition, modalities selection and EKRL concentrator must be used as a wrapper for third party components. Software portability of architecture, components and services on multiple platforms is guaranteed by W3C standards compliance, web services discovery and composition, and the EKRL concentrator. User preferences and profiles can be transmitted to the architecture by the means of input service dedicated to user as we have presented in architecture samples, and the user context can be retrieved in the global context. This context will be taken in consideration in the fission process to choose suitable modalities. Scalability of the architecture can be ensured by discovery of distributed services and agents to increase processing and knowledge storage load, and by EKRL protocol and EKRL concentrator to improve communication, reasoning and add or replace third party services. Performance improvements and efficiency must be measured by specific services to validate the necessity and practical advantages of the growth. Finally, reliability relies on modularity and redundancy of web services components.

Req.	Name	Solution
1	Environment Modelling	Ontology-based formalism embedded in the SAMI framework to build the memory of agents and EKRL
2	Uncertainty	Confidence storage in database and EKRL messages
3	Multiviews	Contextual Management and Awareness
4	Communication	EKRL as an ACL and a common language close to NL. Rule models are used to program agents, manage and synchronize events of a task
5	Integration	Standards respect. Architecture modelling. Modalities selection and composition mechanisms. EKRL concentrator part. Inspection of knowledge base.
6	Modularity & Reusability	Design of generic and easy to connect webservices components. Architecture composition and adaptation mechanisms
7	Complexity & Heterogeneity	Optimal policy management and learning. Modalities selection. Heterogeneity management managed by webservices and EKRL concentrator as a wrapper
8	Portability	W3C Standards compliance. Common development tools. Components Genericity and architecture openness.
9	User preferences	Specific modality and User context management as a part of the situational context
10	Scalability	Utility and performance measurement allows managing several policies in different contexts. Optimization and learning may control the scalability
11	Reliability	Modularity of webservice components allows the discovery, evaluation and composition of redundancy and security to ensure reliability to realize a task

Table 7.1 – Fulfilled requirements

7.4 Perspective

Many optimizations can still be done. For example, any interesting and attainable facts must have a query model; in this case, we could introduce a storage limitation: any new produced events following a rule models application which is not attached to any query models can be sent and cannot be stored. On the contrary, this optimization prevents the learning of new query models which will be not efficient if related facts have been eluded.

In future work, we propose to pay more attention in metrics and measures of interaction mechanisms in an architectural and contextual point of view like adaptation for example. We have not focus on this work as it already exists for any standard web service components attaching OWL-S information to UDDI server to manage quality of services and other quality attributes. But we have proposed and shown how to use such services as input. The work of Alain Abran (2010) on metrics may certainly bring a substantial help. For network security

requirement, we have chosen not to enter in this area due to the existing cryptographic mechanisms like virtual private network, HTTP secured layers, Kerberos and single sign on systems for centralized authentication server, and contracts and manifests management like in the Microsoft Robotics platforms. Other security requirements can be processed like human body protection, limits of and wear on hardware parts, data backup, and so on.

Automatic recording of our simple and complex event models will be interesting like for example in (Kim et al., 2010). And finally, we suggest adapting and creating scales, concepts, values and models by learning. They will tend to improve learning by demonstration and developmental robotics paradigms. This will bring the advantage to naturally program the architecture by imitation without using the memory editor.

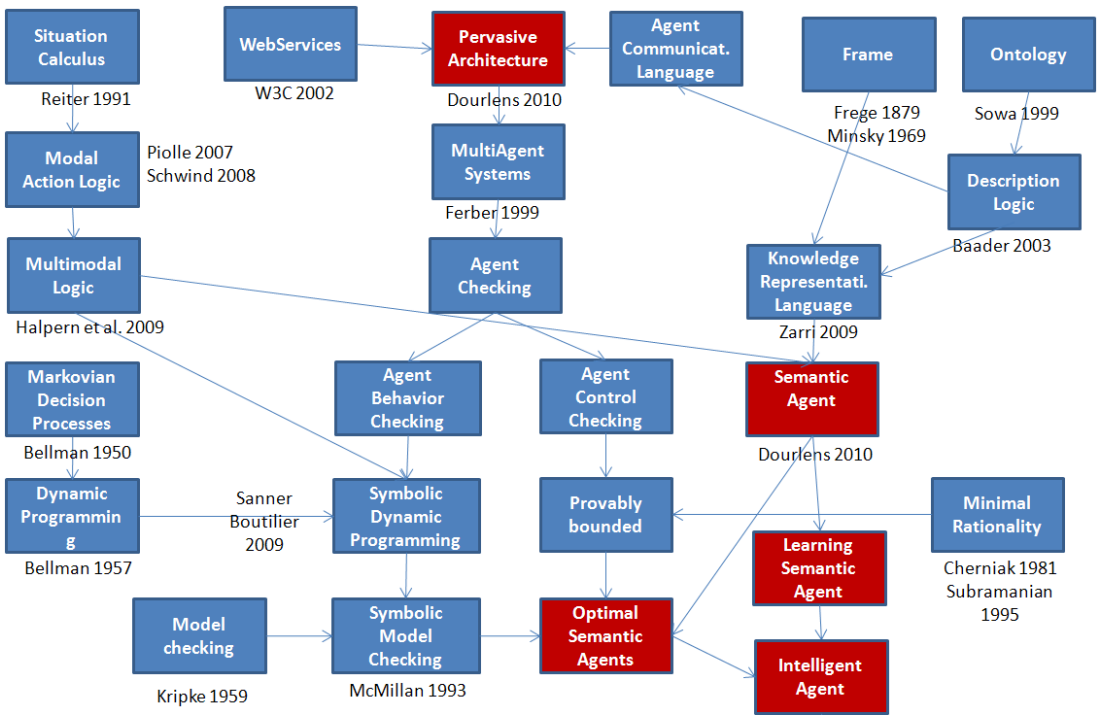


Figure 7.1 – Towards intelligent agent

This work is heading towards more reasoning entities and intelligent agents living in the human environment. This research can evolve to more ambient intelligence to assist the user in its daily tasks like writing a letter from a document template, adapting house to its preference or disabilities, building virtual simulations with vocal commands and gestures, motivating user to move, to walk outside and take care of his health, retrieving information on semantic web. Many applications can be developed with our semantic agents. Figure 7.1 presents in red our contribution to reach this possible goal. Intelligent agent can be defined as an “artificial intelligence” or “artificial life” able to reason on the human environment and find out how to accompany users in the goal of the application to be built by an engineer or a designer.

More personnaly, a future research work is to continue to promote and develop the new domain of *Cognitive & Pervasive Interaction* in ambient intelligence and multi contexts management (Figure 7.2). This new domain will more deeply mix Computational Intelligence, Software engineering, Multimodal Interaction and Knowledge Representation on the basis given in this thesis to better understand, store, learn, act, and interact with humans more naturally.

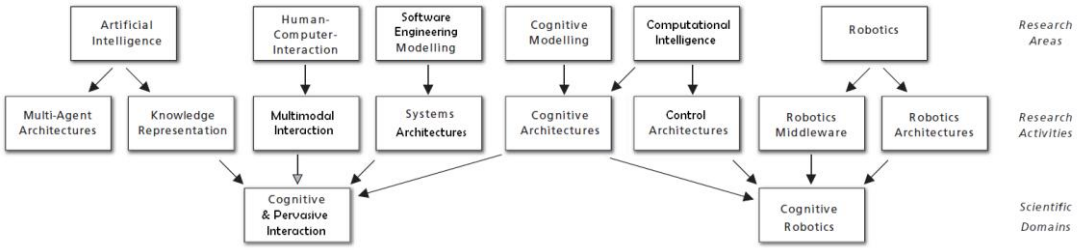


Figure 7.2 – Towards Cognitive & Pervasive Interaction

Other researches may also integrate Developmental Robotics, Cognitive Sciences, Ergonomics paradigms, and simulation experiences (virtual reality and serious games) to learn and evaluate knowledge from observed systems of the real life.

TABLE OF SYMBOLS

Symbol	Name	Description
Ω	Universe	Universe of possible
W	Worlds	World of Universe (global environment $\boldsymbol{W \subset \Omega}$)
ω	World	A world (local environment $\boldsymbol{\omega \subset W}$) or knowledge or Science domain into the world
π	Policy	Policy (a control law, a choice of graph, …)
π^*	Optimal policy	Optimal Policy
Π	Policies	Set of policies functions
X	Situations	Set of possible situations (environment)
S	States	Set of possible states (environment)
A	Actions	Set of actions
x	Situation vector	Vector of a situation
s	State	State of agent
a	Action	Action in environment or into agent
g	Goal	Goal (an agent state)
e	Event	Event or Experiences
L	Language	Knowledge representation language Formal Description Logic Language Alphabet* of symbols (labels) of things
K	Knowledge	Knowledge or Memory of Agent(s)
k	New knowledge	New knowledge
C	Ontology of concepts	Classes of Concept Ontology (included in K) A Formal Classification
P	Ontology of event models	Ontology of classes of event models, Set of predicates (included in K) or Set of possible events or experiences
c	Concept	Concept (Class) label of C – *Atomic concept of C*
ci	Concept instance	Concept Instance (noun,…)
p	Event Model Predicate	Predicate or logic clause of P
pi	Predicate instance	Predicate Instance of $P(= e)$
V	Value function	Value function or set of Values
v	Value	Value of state
Q	Quality function	Quality function or set of Qualities
q	Quality	Quality of a state

MR	Roles	Set of roles from Meta ontology
R	Role	Role of a Predicate/Event
KA	Arguments	Set of arguments
ka	Argument	Argument of Predicates
T	Transitions function	Probability of transition $\boldsymbol{T}: \boldsymbol{S} \times \boldsymbol{A} \rightarrow \prod(\boldsymbol{S})$
τ	Belief function	Belief state transition function
B	Beliefs	Set of belief states over Partially Observable Markov Decision Process (PO MDP)
J	Objective function	Objective function or Optimization criteria
O	Observations	Set of observations
t	Time	Time
α	Learning Coefficient	Learning or Reinforcement (refinement) of knowledge
D	Decisions	Set of decisions (subset of P)
G	Graphs	Set of graphs/Scenarios (state-action graph)
λ	Meta Language	Meta models & Meta concepts
N	Node	Node
M	Meanings Set	Possible meanings/senses of word ci
m	meaning	
I	Invariants	Constraints, Invariants, Forced, Limits
U	Utility function	
r	Reward function	Reward function $\boldsymbol{R}: \boldsymbol{S} \times \boldsymbol{A} \rightarrow \boldsymbol{R}$
H	Hypothesis	Hypothesis space describing policy functions $\boldsymbol{\pi}: \boldsymbol{S} \rightarrow \boldsymbol{A}$
$\widehat{\boldsymbol{Q}}$	Quality estimation	
E	Esperance	
μ	measure	Probability distribution
γ	Discount Factor	
ϕ	Formula	A formal proposition

S'=E(S), E is a list of e (events representing a possible action), *S* a list of e (all events to represent a state).

*An *alphabet* is a finite set of elements which are called *symbols*. For example, {a, b, c, . . . , z} is an alphabet, as is {up, down, left, right}.

REFERENCES

Personal References

Dourlens and Ramdane-cherif	2010a	Dourlens S. and A. Ramdane-Cherif , "Cognitive Memory for Semantic Agents in Robotic Interaction", In *ICCI 2010 - 9th IEEE International Conference on Cognitive Informatics* , 7 July 2010, Beijing, China. Conference article.
Dourlens and Ramdane-cherif	2010b	Dourlens S. and A. Ramdane-Cherif , "Semantic Memory for Pervasive Architecture". In *ICICA 2010 - International Conference on Information Computing and Applications*, Lecture Notes in Computer Science (LNCS), October 15-18, 2010, Tangshan, China. Conference article.
Dourlens and Ramdane-cherif	2011a	Dourlens S. and A. Ramdane-Cherif, "Semantic Architecture for Human Robot Interaction", a chapter in the *Edited book in Studies in Computational Intelligence* by Springer-Verlag. SASFA 'Semantic Agent Systems - Foundations and Applications ', Editors: A. Elci, M. Kone, & M. Orgun. Vol. 344 , 1st Edition., 2011, XVI, 316 p. 104 illus., Hardcover, ISBN: 978-3-642-18307-2. Published book chapter
Dourlens and Ramdane-cherif	2011b	Dourlens S. and A. Ramdane-Cherif , "Cognitive Memory for Semantic Agents in Robotic Interaction", In *Journal of Cognitive Informatics and Natural Intelligence* (IJCINI), Vol.5, Issue 1, April 2011, Canada. Published book chapter.
Dourlens and Ramdane-cherif	2011c	Dourlens, S. and A. Ramdane-Cherif , "Semantic modeling & understanding of environment behaviours", In *SSCI 2011 - IEEE Symposium Series on Computational Intelligence*, Symposium on Intelligent Agents, April 11-15, 2011 - Paris, France. Published conference article.

Literature References

Frege	1879	Frege Gottlob. *Begriffsschrift*. 1879
Russel	1921	Russel, Bertrand. *The analysis of mind*. IndyPublish (January 2, 2002) ISBN-10: 1588277097 ISBN-13: 978-1588277091 - Original copyright 1921
Von Neumann and Morgenstern	1944	Von Neumann, J., and Morgenstern, O. (1944) *Theory of games and economic behavior*. Princeton: Princeton University Pres
Nash	1950	Nash, John. 1928. "Equilibrium Points in n-Person Games". In *Proceedings of the National Academy of Sciences of the United States of America.*1950;36;48-49
Piaget	1950	Piaget, Jean . *Introduction à l'épistémologie génétique.* Volume II. 1950, p. 189-190
Bellman	1957	Bellman, Richard. 1957. *Dynamic Programming*. Princeton University Press, Princeton
Minksy	1965	Minsky Marvin. (1965) *"Matter, Mind and Models"* Proceedings of IFIP Congress 1965, I, May, 1965, pp. 45-49., Spartan Books, Washington D.C.
Piaget	1967	Piaget, Jean . *Logique et connaissance scientifique.* 1967. p. 318
Vendler	1967	Vendler, Z. (1967) "Verbs and times". In *Linguistics in Philosophy*, pages 97-121. Cornell University Press, Ithaca, NY.
Quillian	1968	Quillian, Ross (1968) *Semantic memory*. Ph.D. thesis (1966) at Carnegie Intstitute of Technology. in *Semantic Information Processing*, M. Minsky, ed., MIT Press, 1968, p. 262
Searle	1969	Searle, J. R. (1969) *Speech Acts: An Essay in the Philosophy of Language*. Cambridge: Cambridge University Press.
Bronowski	1970	Bronowski, Jacob. *Creativity: A discussion at the Nobel Conference* organized by Gustavus Adolphus College, St. Peter, Minnesota, 1970
Minksy	1974	Minsky, Marvin (1974) "*A Framework for Representing Knowledge*", MIT-AI Laboratory Memo 306, June, 1974. (Shorter versions in The Psychology of Computer Vision," P. Winston (Ed.), McGraw-Hill 1975, Mind Design," J. Haugeland, Ed., MIT Press, 1981, and "Cognitive Science," Collins, Allan and Edward E. Smith (eds.), Morgan-Kaufmann 1992.)
Piaget	1974a	Piaget, Jean . *Sagesse et illusion de la philosophie.* 1974a. p. 174
Piaget	1974b	Piaget, Jean. *La prise de conscience*. 1974b. Paris: PU pp. 273-273
Piaget	1975	Piaget, Jean. *L'équilibration des structures cognitives : problème central du développement*. 1975.
Newell and Simon	1976	Newell, A., and Simon, H. A. (1976). "Computer science as empirical inquiry: Symbols and search". *Communications of the ACM* (Association for Computing Machinery), 19, 113-126.
Bates	1979	Bates, E.. *The Emergence of Symbols*. Academic Press, 1979. See Chapter 2: Intentions, Conventions, Symbols.

Bolt	1980	Bolt R.: "Put That Here": Voice and gesture at the graphics interface. *Proceedings of the 7th annual conference on Computer Graphics and interactive techniques* (1980)
Posner	1980	Posner, M.I. "Orienting of attention". *Quarterly Journal of Experimental Psychology*, (32):3–25, 1980.
Reid et al.	1981	Reid G., Shingledecker C., Nygren T. & Eggemeier T. (1981) "Development of multidimensional subjective measures of workload", *International conference on Cybernetics & Society*, Atlanta, 403-406
Simon	1982	Simon, H. A. (1982) *Models of bounded rationality, Volume 2*. Cambridge: MIT Press.
Piaget	1983	Piaget, Jean . *Psychogenèse et histoire des sciences.* 1983. p.30 / p. 39-40 / p. 235-237
Tarski and Lindenbaum	1983	Tarski, A. and A. Lindenbaum, 1983. "On the Limitations of the Means of Expression of Deductive Theories", translation of Tarski and Lindenbaum 1935 by J.H. Woodger in Tarski 1983a, pp. 384–392.
Sowa	1984	Sowa J., "Conceptual Structures : Information Processing" in *Mind and Machine, The Systems Programming Series*, Addison-Wesley, 1984
Briggs	1985	Briggs, Rick (1985), "Knowledge Representation in Sanskrit and Artificial Intelligence (PDF)", *AI Magazine* 6 (1)
Fillmore	1985	Fillmore, Charles J. (1985) "Frames and the semantics of understanding". In *Quaderni di Semantica*, Vol. 6.2: 222-254.
Minsky	1985	Minsky, Marvin. (1985). *Society of Mind.* See Chapter 7: Problems and Goals.
Takagi and Sugeno	1985	Takagi, T. and Sugeno, M. (1985) "Fuzzy identification of systems and its application to modelling and control". In *IEEE Transactions on Systems, Man, and Cybernetics* 15, 116–132.
Brooks	1986	Brooks. R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2,14-23
Doyne Farmer et al.	1986	*Evolution, Games and Learning: Models for Adaptation in Machines and Nature*, ed. Doyne Farmer et al. (Amsterdam: North Holland, 1986), 190.
Kowalski and Sergot	1986	Kowalski, N., Marek J. Sergot "A Logic-based Calculus of Events". In *New Generation Comput.* 4(1): 67-95 (1986)
Winograd and Flores	1986	Winograd, T. and F. Flores. (1986). *Understanding computers and cognition: A new foundation for design.*Addison-Wesley. See Chapters 8 and 9, pages 93-124.
Grossberg	1987	Grossberg, S. (1987). Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science* 11: 23-63. http://web.umr.edu/~tauritzd/art/
Laird et al.	1987	Laird.J., Newell, A.. and Rosenbloom, P. S. (1987) "SOAR:An architecture for general intelligence". *Artificial Intelligence Journal*, Vol. 33 (1), 1-64. http://www.soartechnology.com
Lucassen	1987	Lucassen, John M. . *Types and Effects: Towards the Integration of Functional and Imperative Programming*. PhD thesis, MIT, 1987.
Papadimitriou and Tsilsilis	1987	Papadimitriou, C. H. and Tsilsilis, J. N. (1987). "The complexity of Markov decision processes". In *Mathematics of Operations Research*. 12(31), 441-450.
Suchman	1987	Lucy A. Suchman, *Plans and Situated Actions*, Cambridge University Press, Cambridge, UK, 1987.
Tarski and Givant	1987	Tarski, A. and S. Givant, 1987. "A Formalization of Set Theory without Variables". In *RI: American Mathematical Society*. Providence
Brooks	1988	Brooks. R.A. (1988) "Engineering approach to building complete, intelligent beings". *Proc. SPIE the international Society for Optical Engineering*, 1002,618-625.
Hart and Staveland	1988	Hart, S. G., and L. E. Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Human mental workload*, ed. P. A. Hancock, and N. Meshkati, 139–183, Amsterdam: The Netherlands.
Horvitz	1988	Horvitz, E. J. (1988) "Reasoning about beliefs and actions under computational resource constraints". In Levitt, T., Lemmer, J., and Kanal, L. (Eds.) *Uncertainty in Artificial Intelligence* S. Amsterdam: North Holland
Moens and Steedman	1988	Moens, M. and Steedman, M. (1988) "Temporal Ontology and Temporal Reference". In *Computational Linguistics*, 14:15-38.
Gardner and Hatch	1989	Gardner,H., Hatch, T. (1989) "Multiple intelligences Go to School:Educational Implications of the Theory of Multiple Intelligences". In *Educational Researcher*, 18(8),4-9
Hughes	1989	John Hughes. "Why functional programming matters". In *Computer Journal*, 32(2):98–107,

1989. Reprinted n [Tur90], pp. 17–42.

Agre and Chapman	1990	Agre, P. E. and Chapman, D. (1990). "What are plans for?" In *Robotics and Autonomous Systems*, 6:17-34.
Breese and Fehling	1990	Breese, J. S., and Fehling, M. R. (1990) "Control of problem-solving: Principles and architecture". In Shachter, R. D., Levitt, T., Kanal, L., and Lemmer, J. (Eds.) *Uncertainty in Artificial Intelligence* 4- Amsterdam: North Holland.
Cherniak	1990	Christopher Cherniak. *Minimal Rationality.* Publisher: The MIT Press ; 1990-03-14 ; ISBN: 0262530872
Cohen and Levesque	1990	Cohen, P. R. and Levesque, H. J. (1990). "Intention is choice with commitment". In *Artificial Intelligence*, 42:213-261.
Coward	1990	Coward, L. A. (1990). *Pattern Thinking*. New York: Praeger.
Harnad	1990	Harnad, S. (1990) "The symbol grounding problem". In *Physica D*, 42, 335-346.
Lenat	1990	Lenat, Douglas B. 1990. *Building Large Knowledge-Based Systems: Representation and Inference of the CYC Project,* Addison-Wesley.
Mosses	1990	Peter Mosses. "Denotational semantics". In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 575–631. MIT Press/Elsevier, 1990.
Newell	1990	Newell, A. (1990). *Unified Theories of Cognition.* Cambridge, Massachusetts: Harvard University Press
Roscoe and Ellis	1990	Roscoe, A. H., & Ellis, G. A. (1990). *A subjective rating scale for assessing pilot workload in flight (No. TR90019)*. Farnborough, UK: Royal Aeronautical Establishment.
Jensen	1991	Jensen Kurt. Coloured petri nets: a high level language for system design and analysis. In *APN 90: Proceedings on Advances in Petri nets 1990*, pages 342–416, New York, NY, USA, 1991. Springer-Verlag New York, Inc.
Rao and Georgeff	1991	Rao A.S., Georgeff M.P. Modelling rational agents within a BDI-Architecture. In J.Allen, R.Fikes, and E.Sandells Eds, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning.* Morgan Kaufmann Publishers, San Mateo, CA, USA. 1991
Reiter	1991	Reiter R. *The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression.* 1991
Russel and Wefald	1991	Russell, S. J., and Wefald, E. H. (1991) *Do the right thing: Studies in limited rationality*. Cambridge, MA: MIT Press
Jordan	1992	Jordan, C.S. (1992) "*Experimental study of the effects of an instantaneous sef assessment workload recorder on task performance".* Defence Evaluation & Research Agency, Report #DRA/TM(CAD5)/92011
Crowley and Demazeau	1993	Crowley, James L and Demazeau, Yves. "Principles and Techniques for Sensor Data Fusion". *Journal of Signal Processing - Intelligent systems for signal and Image understanding*. Volume 32 Issue 1-2, May 1993. Elsevier North-Holland, Inc. Amsterdam, The Netherlands, The Netherlands
Gruber	1993	Gruber T. R. (1993) "A translation approach to portable ontology specifications".In *Knowledge Acquisition*, 5(2):199–220.
Weiser	1993	Weiser, M. (1993) "Some computer science issues in ubiquitous computing". In *Communications on the ACM*, 36(7):75–84.
Arnborg	1994	Arnborg, S. (1994) *A general purpose msol model checker and optimizer based on boolean function representation*. Technical report, The Royal Institute of Technology, Stockholm, Sweden.
Nau	1994	Nau Dana. *Universal Method Composition Planner (UMCP) was the first provably sound and complete HTN planning system*, 1994. http://www.cs.umd.edu/projects/plus/umcp/manual
Puterman	1994	Putermann. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* 1994
Schilit et al.	1994	Schilit, Bill N., Adams N. and Want R. (1994) "Context-Aware Computing Applications". In *Proceedings of the Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA,* (Decenber 1994). p. 85-90. Santa Cruz, CA, : IEEE Computer Society.
Coutaz et al.	1995	Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J. Young, R.: Four Easy Pieces for Assessing the Usability of Multimodal Interaction: The CARE properties. In: *Proceedings of INTERACT'95*, Lillehammer, Norway, June 1995, pp. 115--120, Chapman & Hall Publications. (1995).
Farmer et al.	1995	Farmer, E.W., Jordan, C.S., Belyavin, A.J., Bunting, A.J., Tattersall, A.J. and Jones D.M.

(1995). Dimensions of operator workload. Defence Evaluation & Research Agency, Report DRA/AS/MMI/CR95098/1

Guarino 1995 Guarino, N. (1995) "Formal ontology, conceptual analysis and knowledge representation". In *Human-Computer Studies*, 43(5/6):625–640.

Russell and Subramanian 1995 Russell, S. and Subramanian, D. (1995) "Provably bounded-optimal agents". In *Journal of AI Research*,2:575–609

Goillau and Kelly 1996 Goillau, PJ. And Kelly, C.J. (1996) "Malvern Capacity Estimate (MACE) - A proposed cognitive measure for complex systems" in *First International Conference on Engineering Psychology and Cognitive Ergonomics*.

O'Reilly 1996 O'Reilly, R.C. (1996), 'Biologically Plausible Error-Driven Learning Using Local Activation Differences: The Generalized Recirculation Algorithm', *Neural Computation*, 8, 895–938.

Zarri 1996 Zarri, gian piero, "NKRL, a Knowledge Representation Language for Narrative Natural Language Processing" in *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*

Anderson and Sutton 1997 Anderson, James and Sutton, Jeffrey (1997) "If we compute faster, do we understand better?" In *Behavior Research Methods*. Springer New York, pp,67-77(29),1.

Bayardo et al. 1997 Bayardo R.J. Jr., W. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk. "InfoSleuth: Agent-Based semantic integration of information in open and dynamic environments". *Proc. ACM SIGMOD Intern. Conf. on Management of Data*, 1997 http://www.mcc.com/projects/infosleuth

Genesereth et al. 1997 Genesereth, M. R.; A. M. Keller, and O. M. Duschka: Infomaster: An Information Integration System. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Tucson, AZ, May 1997

Mitchell 1997 Mitchell, Tom. *Machine Learning*, McGraw Hill, 1997

Smith and Becker 1997 Smith S.F., Becker M.A. (1997) "An ontology for constructing scheduling systems". Working Notes from 1997 *AAAI Spring Symposium on Ontological Engineering*, Stanford, CA, March 1997 (AAAI Press).

Anderson and Lebiere 1998 Anderson, J. R., & Lebiere, C. (Eds.). (1998). *The atomic components of thought*. Mahwah, NJ: Lawrence Erlbaum.

Baker et al. 1998 Baker C. F., Fillmore C. J., and Lowe J. B. 1998. "The Berkeley FrameNet Project". In *Proceedings of ACL/Coling'98*, pages 86-90. http://framenet.icsi.berkeley.edu/

Bratman et al. 1998 Bratman, M. E., Israel, D. J. and Pollack, M. E. (1988), "Plans and resource-bounded practical reasoning". *Computational Intelligence*, 4: 349–355. doi: 10.1111/j.1467-8640.1988.tb00284.x

Doyle 1998 Doyle, John (1998) "Bounded Rationality". in *The MIT Encyclopedia of the Cognitive Sciences* (Rob Wilson and Frank Kiel, editors), Cambridge, Massachusetts: MIT Press

Guarino 1998 Guarino N. (1998) "Formal ontology and information systems", In N.Guarino (Ed.), *Proceedings of the 1st International conference*, Trento, Italy, June 1998, IOS Press Amsterdam, The Nederlands, p. 337, ISBN: 9051993994

Kaelbling et al. 1998 Kaelbling, L.P., Littman, M.L., Cassandra, A.R. (1998). "Planning and acting in partially observable stochastic domains". *Artificial Intelligence Journal* 101: 99–134

Sengers 1998 Sengers Phoebe (1998) "Narrative Intelligence." In *Human Cognition and Social Agent Technology (Advances in Consciousness vol. 19)*, edited by Kerstin Dautenhahn. Amsterdam: John Benjamins

Sutton and Barto 1998 Sutton, Richard S.; Barto, Andrew G. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998. A Bradford Book.

Barsalou 1999 Barsalou, L. (1999). "Perceptual symbol systems". *Behavioural and Brain Sciences*, 22, 577-609.

Coen et al. 1999 Coen, M., Phillips, B., Warshawsky, N., Weisman, L., Peters, S., and Finin, P., 1999. Meeting the Computational Needs of Intelligent Environments: The Metaglue System. In: Nixon, P., Lacey, G., and Dobson, S. (eds.), *1st International Workshop on Managing Interactions in Smart Environments* (MANSE'99), Springer-Verlag, pp. 201–212. http://www.ai.mit.edu/projects/aire/papers.shtml.

Ferber 1999 Ferber, Jacques. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley, 1999. ISBN 0-201-36048-9

Gomez Perez and Benjamins 1999 Gomez Perez A. and Benjamins V.R. (1999) Applications of Ontologies and Problem-Solving Methods. AI-Magazin. Vol 20. N°1. AAAI Press. pp 119-122. ISSN 0738-4602

Pfeifer and 1999 Pfeifer, R., & Scheier, C. (1999). *Understanding Intelligence*. Cambridge, MA: MIT Press.

Scheier

Shedroff 1999 Shedroff, N. (1999) "Information interaction design: A unified field theory of design". In Jacobson, R., editor, *Information Design*, chapter 11, pages 267–292. The MIT Press Cambridge Massachusetts, 1st edition

Sowa 1999 Sowa John F. (1999) *Knowledge Representation : Logical, Philosophical, and Computational Foundations*, Brooks Cole Publishing Co., Pacific Grove, CA, USA, p. 594, ISBN: 0-534-94965-7

Thorisson 1999 Thórisson, K. R. (1999). A Mind Model for Multimodal Communicative Creatures and Humanoids. *International Journal of Applied Artificial Intelligence*, 13(4-5): 449-486

Young and Lewis 1999 Young,Richard M ; Richard L Lewis. "Soar and Human Working Memory." Chapter 7 of: A. Miyake & P. Shah (eds), *Models of Working Memory: Mechanisms of Active Maintenance and Executive Control*, 224-256. Cambridge University Press, 1999.

Dupuy 2000 Jean-Pierre Dupuy. *The Mechanization of the Mind*. Princeton University Press (December 15, 2000)

Fielding 2000 Fielding, R. Roy Thomas. Architectural styles and the design of network-based software architectures. PhD thesis, University of California, Irvine, 2000.

Chen and Kotz 2000 Chen Guanling, Kotz David. (2000) *A Survey of Context-Aware Mobile Computing Research. TR2000-381. Dartmouth: Dartmouth College Computer Science*

Lassila 2000 Lassila, Ora. "The Resource Description Framework", *IEEE Intelligent Systems* 15(6): 67-69 (November/December 2000)

Makatchev 2000 Makatchev Maxim, Tso S. K. (2000) "Human-Robot Interface Using Agents Communicating in an XML-Based Markup Language," *Proceedings of the 2000 IEEE International Workshop on Robot and Human Interactive Communication ROMAN2000*, Osaka, Japan, September 27-29, pp. 270-275

Petrelli et al. 2000 Petrelli, D., Not E., Strapparava C., Stock O. and Zancanaro M. (2000) "Modeling Context is Like Taking Pictures". In *CHI'2000 Workshop on Context Awareness* (April 1-6, 2000). The Hague, Netherlands.

Piaggio et al. 2000 M. Piaggio, A. Sgorbissa, and R. Zaccaria. Pre-emptive versus non pre-emptive real time scheduling in intelligent mobile robotics. *Journal of Exp. and Theoretical Artificial Intelligence*, 12(2), 2000

Reiter 2000 Reiter,Ray "Narratives as Programs" In *Proc. of the Seventh International Conference on Knowledge Representation and Reasoning* (KR2000) pages 99--108. Morgan Kaufmann, 2000

Tambe et al. 2000 Tambe Milind, David V.Pynadath, Nicolas Chauvat, Abhimanyu Das, Gal A.Kaminka. "Adaptive Agent Integration Architectures for Heterogeneous Team Members". *Fourth International Conference on MultiAgent Ssytems Proceedings*. (2000)301-308. Boston, MA. USA.

Boutilier et al. 2001 Boutilier, Craig, Reiter Ray, Price Bob. 2001. "Symbolic Dynamic Programming for First-Order MDPs". Ed. Bernhard Nebel in *Proceedings of the International Joint Conference on Artificial Intelligence* (IJCAI) (August 2001), pp. 690-700.

Dey 2001 Dey, A. K. (2001) "Understanding and using context". *Personal and ubiquitous computing*, vol. 5, p. 4-7.

FIPA 2001 Foundation for Intelligent Physical Agents. *Ontology Service Specification*. XC00086D. 2001/08/10. http://www.fipa.org/specs/fipa00086/XC00086C.html

Maffioletti 2001 Maffioletti S. (2001) "Requirements for an ubiquitous computing infrastructure". *Cultura Narodov Prichernomoria journal*, 3:35–40. Mezhvuzovskii publisher, Simferopol, Ukraine, ISSN 1562-0808

McIlraith et al. 2001 McIlraith,Sheila A., Tran Cao Son, Honglei Zeng. "Semantic Web Services". *IEEE Intelligent Systems* 16(2): 46-53 (2001).

Murdock et al. 2001 Murdock, J.W. and Goel, A.K. (2001). Meta-Case-Based Reasoning: Using Functional Models to Adapt Case-Based Agents. Proceedings of the 4th International Conference on Case-Based Reasoning (ICCBR'01). Vancouver, Canada, July 30 - August 2, 2001.

Reiter 2001 Reiter R. (2001). *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press.

Bourguet 2002 Bourguet, M. L.: A Toolkit for Creating and Testing Multimodal Interface Designs. In: *Companion proceedings* of UIST'02, Paris, Oct. 2002, pp. 29--30 (2002).

Chai 2002 Chai, Joyce. "Semantics-based Representation for Multimodal interpretation in Conversational Systems". *COLING 2002: The 19th International Conference on Computational Linguistics*.

Chella et al. 2002 Chella A, Cossentino M., Pirrone R., Ruisi A. (2002) "Modeling Ontologies for Robotic

Environments". *Proceedings of the 14th international conference on Software engineering and knowledge engineering. Session Knowledge engineering tools and techniques*. Pages: 77 - 80. ISBN:1-58113-556-4

Chen and Kotz	2002	Chen, Guanling; Kotz, David. "Context Aggregation and Dissemination in Ubiquitous Computing Systems". In *4th IEEE Workshop on Mobile Computing Systems and Applications* (WMCSA 2002), 20-21 June 2002, Callicoon, NY, USA.
Cohen et al.	2002	Cohen, P. R., Johnston, M., McGee, D., Oviatt, S., Pittman, J., Smith, I., Chen, L., Clow, J.: QuickSet: multimodal interaction for distributed applications. In: *Proceedings of the Fifth ACM international Conference on Multimedia*, Seattle, USA, pp. 31--40, (1997).
Corbetta and Shulman	2002	Corbetta M. and G. L. Shulman. Control of goal-directed and stimulus-driven attention in the brain. *Nature Reviews Neuroscience*, 3:201–215, 2002.
Foster et al.	2002	Foster, I., C. Kesselman, J. Nick, and S. Tuecke, The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, , Global Grid Forum, 2002.
Hofer et al.	2002	Hofer T., Schwinger W., Pichler M., Leonhartsberger G. et Altmann J. (2002) "Contextawareness on mobile devices - the hydrogen approach". In *the 36th Annual Hawaii International Conference on System Sciences*. p. 292-302. Hawaii, USA.
Krahnstoever et al.	2002	Krahnstoever, N., Kettebekov, S., Yeasin, M. Sharma, R.: A real-time framework for natural multimodal interaction with large screen displays. In: *ICMI'02*, Pittsburgh, USA, Oct. 2002.
Kransted et al.	2002	Kranstedt A., Kopp S., and Wachsmuth I. (2002) "Murml: A multimodal utterance representation markup language for conversational agents". In *Proc. Of the AAMAS. Workshop on "Embodied conversational agents - Let's specify and evaluate them"*.
Tanenbaum and Van Steen	2002	Tanenbaum, Andrew and Van Steen, Maarten . *Distributed Systems: Principles and Paradigm.*Publisher: Prentice Hall; (January 15, 2002), ISBN-10: 0130888931, ISBN-13: 978-0130888938
Tarski	2002	Tarski, A., 2002. "On the Concept of Following Logically", translation of Tarski 1936a by M. Stroinska and D. Hitchcock. *History and Philosophy of Logic*, 23: 155–96.
Barsalou	2003	Barsalou, L.W. (2003). "Abstraction in Perceptual Symbol Systems". *Philosophical Transactions of the Royal Society of London: Biological Sciences, 358, 1177-1187.*
Eberhart	2003	Eberhart Andreas. *Ontology based infrastructure for intelligent applications*. PhD Thesis dissertation. 18.12.2003
Flippo et al.	2003	Flippo, F., Krebs, A. Marsic I.: A Framework for Rapid Development of Multimodal Interfaces. In: Proceedings of ICMI'03, Vancouver, BC, Nov. 5-7, pp. 109--116 (2003).
Fong et al.	2003	Fong, Terrence, Illah Nourbakhsh, Kerstin Dautenhahn, "A survey of socially interactive robots, Robotics and Autonomous Systems", Volume 42, Issues 3-4, 31 March 2003, Pages 143-166, ISSN 0921-8890, DOI: 10.1016/S0921-8890(02)00372-X.
Gibbon et al.	2003	Gibbon, D.,Gut U., Hell B. Looks K. and Trippel A.T.T. (2003) "A computational model of arm gestures in conversation". In *EUROSPEECH-2003*. 2003. pp 813-816
Hangos and Cameron	2003	Hangos, K. and Cameron I. (2003) *Process modeling and model analysis*. Academic Press.
Medjahed et al.	2003	Medjahed, Brahim,Bouguettaya Athman, Elmagarmid Ahmed K. (2003) "Composing Web services on the Semantic Web". *VLDB Journal* 12(4). pp 333-351
Obrst	2003	Obrst, Leo (2003) "Ontologies for semantically Interoperable Systems". *Conference on information ans knowledge Management. Proceedings of the twelfth international conference on Information and knowledge management.* New Orleans. LA, USA. Pp 366-369, ACM Press, New York, NY, USA. ISBN: 1-58113-723-0
Schmidt	2003	Schmidt A. (2003) *Ubiquitous Computing - Computing in Context*. PhD thesis, CSEG, Lancaster University, Lancaster, England, U.K.
Sycara et al.	2003	Sycara, Katia P. , Massimo Paolucci, Anupriya Ankolekar, and Naveen Srinivasan. "Automated discovery, interaction and composition of semantic web services". *Journal Web Semantics*, 1(1):27{46, 2003.
Van der Hoek and Wooldridge	2003	Van der Hoek, W., and Wooldridge, M. 2003. "Towards a logic of rational agency". *L.J. of IGPL* 11(2):135-159
Alonso et al.	2004	Alonso, Katia P. , Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services: Concepts, Architectures and Applications.* Springer-Verlag, 2004.
Asaad and Mubarak	2004	Asaad H., Mubarak S. (2004) "Ontology and Taxonomy Collaborated Framework for Meeting Classification". *17th conference of the International Conference on Pattern Recognition*. ICPR 2004.
Baader et al.	2004	Baader, Franz, Ian Horrocks, Ulrike Sattler. *Description Logics. Handbook on Ontologies.*

2004 pp. 3-28.

Barrouillet et al. 2004 Barrouillet, P., Bernardin, S. and Camos, V. (2004). Time constraints and resource sharing in adults' working memory spans. *Journal of Experimental Psychology: General*, 133, 83-100.

Bouchet et al. 2004 Bouchet, J., Nigay, L., Ganille, T.: ICARE Software Components for Rapidly Developing Multimodal Interfaces. In: Conference Proceedings of ICMI'2004, State College, Pennsylvania, USA, October 2004, ACM Press, pp. 251--258 (2004)

Brezillon et al. 2004 Brezillon, P., Borges M. R., Pino J.A. and Pomerol J.-Ch. (2004) "Context-Awareness in Group Work: Three Case Studies". In *2004 IFIP Int. Conf. on Decision Support Systems (DSS 2004)* (Juillet, 2004). Prato, Italie.

Brickley and Guha 2004 RDF Vocabulary Description Language 1.0: RDF Schema, World Wide Web Consortium, Recommendation REC-rdf-schema-20040210, February 2004.

Cassimatis et al. 2004 Cassimatis, N.L., Trafton, J.G., Bugajska, M.D., & Schultz, A.C. (2004). Integrating cognition, perception and action through mental simulation in robots. *Journal of Robotics and Autonomous Systems* 49 (1-2): 13-23.

Chalmers 2004 Chalmers, M. (2004) "A historical view of context". *Computer supported cooperative work : CSCW*, vol. 13, p. 223-247.

Constantinescu et al. 2004 Constantinescu, Ion ,Boi Faltings, and Walter Binder. "Type based service composition". In *WWW* (Alternate Track Papers & Posters), pages 268-269, 2004

Djenidi 2004 Djenidi H. « Étude des interactions multimodales dans les applications multimédia en vue de proposer des architectures logicielles génériques pour ces applications » PhD (2004)

Esteva et al. 2004 Esteva, Marc, Bruno Rosell, Juan A. Rodríguez-Aguilar, Josep Lluís Arcos. "AMELI: An Agent-Based Middleware for Electronic Institutions". *AAMAS* 2004: 236-243

Ghallab et al. 2004 Ghallab M., Nau Dana, Traverso Paolo. *Automated Planning theory and practice*. Morgan Kaufmann Publishers, May 2004, 663 pages. ISBN 1-55860-856-7

Gil Iranzo 2004 Gil Iranzo, Rosa María , 2004. *Agent Negotiating in a semantic web architecture SWA*. PhD Thesis Universitat Pompeu Fabra, Spain. nov 2004.

Horrocks et al. 2004 Horrocks, I., Patel-Schneider, P., Boley, H. Tabet, S., Grosof, B., and Dean, M. 2004. "SWRL: A semantic web rule language combining OWL and RuleML" *W3C Member submission*, 21.

Kim et al. 2004 Kim,Jong-Hwan; Yong-Duk Kim, Kang-Hee Lee. 2004. "The Third Generation of Robotics: Ubiquitous robot". *2nd International Conference on Autonomous Robots and Agents*. December 13-15, 2004. Palmerston North. New Zealand

Landragin et al. 2004 Landragin Frédéric, Denis A., Ricci A., Romary L. (2004) "Multimodal meaning representation for generic dialogue systems architectures". In *Proc. on Language Resources and Evaluation* (LREC 2004)., 2004, pp. 521-524

Malle 2004 Malle, B.F. *How the mind explains behavior: Folk explanations, meaning, and social interaction.* MIT Press, Cambridge, 2004.

Manola and Miller 2004 Manola,Frank; Miller, Eric. RDF Primer, World Wide Web Consortium, Recommendation REC-rdf-primer-20040210, February 2004.

Martin et al. 2004 Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T. et al. 2004 "OWL-S: Semantic Markup for Web Services, W3C Member Submission". http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/.

McGuinness and Van Harmelen 2004 D L McGuinness, F Van Harmelen (2004) OWL Web Ontology Language Overview. 10 (February). http://www.w3.org/TR/owl-features/

Millikan 2004 Millikan, R.. (2004). *Varieties of Meaning*, MIT Press (excerpt from Introduction).

Oquendo 2004 Oquendo F., "π-ADL: An Architecture Description Language based on the Higher Order Typed π-Calculus for Specifying Dynamic and Mobile Software Architectures," ACM Software Engineering Notes, 2004

Acampora and Loia 2005 Acampora G., Loia V. (2005) "Using FML and Fuzzy Technology in Adaptive Ambient Intelligence Environments". *International Journal of Computational Intelligence Research*. ISSN 0973-1873 Vol.1, No.2 (2005), pp. 171–182. Research India Publications

Albus and Barbera 2005 Albus, J. S., and Barbera, A. J. (2005) "RCS: A cognitive architecture for intelligent multi-agent systems," *Annual Reviews in Control*, Vol. 29, Issue 1 87-99

Bos 2005 Bos Johan (2005) "Towards wide-coverage semantic interpretation". In *Proceedings of Sixth International Workshop on Computational Semantics* IWCS-6. pp 42-53

Deb 2005 Roy Deb, (2005) "Semiotic schemas: A framework for Grounding Language in Action and Perception", *Artificial Intelligence* 167, p. 170–205

Ha et al. 2005 Ha Young-Guk , Sohn Joo-Chan, Cho Young-Jo, Yoon Hyunsoo. "Towards a Ubiquitous

		Robotic Companion: Design and Implementation of Ubiquitous Robotic Service Framework". *ETRI Journal*. Volume 27. Number 6. December 2005
Hawkins	2005	George D, Hawkins J (2009) Towards a Mathematical Theory of Cortical Micro-circuits. PLoS Comput Biol 5(10). "On Intelligence" Jeff Hawkins with Sandra Blakeslee, 2005
Heckmann	2005	Heckmann, Dominik. 2005. *Ubiquitous User Modeling.* PhD thesis, Computer Science Department, Saarland University, Germany
Kushida et al.	2005	Kushida, Kazutaka, Nishimura Yoshitaka, Dohi Hiroshi, Ishizuka Mitsuru, Takeuchi Johane, Tsujino Hiroshi (2005) "A markup language for describing interactive humanoid robot presentations". *AAMAS'05,* July 25-29, Utrecht, Netherlands
Lech and Wienhofen	2005	Lech, Till C., Leendert W. M. Wienhofen. 2005. "AmbieAgents: A Scalable Infrastructure for Mobile and Context-Aware Information Services" In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems* (2005), pp. 625-631.
Oquendo	2005	Oquendo F., "Tutorial on ArchWare ADL – Version 2 (π-ADL Tutorial)," ArchWare European RTD Project IST-2001-32360, 2005
Razzaque et al.	2005	Razzaque, M. A., Dobson S. and Nixon P. (2005) "Categorization and modeling of quality in contextinformation". In *the IJCAI 2005 Workshop on AI and Autonomic Communications* (August 2005). Edinburgh, Scotland.
Samsonovitch and De Jong	2005	Samsonovich, A. V. and De Jong, K. A. (2005). Designing a self-aware neuromorphic hybrid. In K.R. Thorisson, H. Vilhjalmsson, and S. Marsela (Eds.). *AAAI-05 Workshop on Modular Construction of Human-Like Intelligence*: AAAI Technical Report, volume WS-05-0
Sansonnet et al.	2005	Sansonnet J-P., Martin J-C., Leguern K.A (2005)" Software Engineering Approach Combining Rational and Conversational Agents for the Design of Assistance Applications", T. Panayiotopoulos et al. (Eds.): IVA 2005, *LNAI 3661,* pp 111 - 119
Singh	2005	Singh Push (2005) *EM-ONE: An Architecture for Reflective Commonsense Thinking.* PhD Thesis. MIT 2005
Sirin et al.	2005	Sirin, E., Parsia, B., Hendler, J.: Template-based composition of semantic Web services. *AAAI-05 Fall Symposium on Agents and the Semantic Web*, Arlington-Virginia (USA) (2005).
Steedman and Baldridge	2005	Steedman Mark and Baldridge Jason (2005) "Combinatory Categorial Grammar to appear" in: R. Borsley and K. Borjars (eds.) *Non-Transformational Syntax,* Blackwell
Stuckenschmidt and Van Harmelen	2005	Stuckenschmidt, Heiner, Van Harmelen, Frank. *Information sharing on the semantic web*. Springer. 2005
Weyns et al.	2005	Weyns, D. ,H. V. D. Parunak, F. Michel, T. Holvoet, and J. Ferber. "Environments for Multiagent Systems, State-of-the-Art and Research Challenges". In D. Weyns, H. V. D. Parunak, and F. Michel, editors, *Environment for Multi–Agent Systems'04,* volume 3374 of LNAI, pages 1–47. Springer, 2005.
Zhuge	2005	Zhuge, Hai. 2005. Semantic grid: scientific issues, infrastructure, and methodology. Commun. ACM 48, 4 (April 2005), 117-119. DOI=10.1145/1053291.1053325
Blasch et al.	2006	Blasch Erik, Kadar Ivan, Salerno John, Kokar M.M., Das Subrata, Powell G.M., Orkill D.D., Ruspini E.H. (2006) "Issues and Challenges in Situation Assessment (Level 2 Fusion)". In *Journal Of Advances In Information Fusion.* Vol. 1, No. 2. December 2006
Franklin and Patterson	2006	Franklin, S. and F. G. Patterson, Jr. (2006). The Lida Architecture: Adding New Modes of Learning to an Intelligent, Autonomous, Software Agent. Integrated Design and Process Technology, IDPT-2006, San Diego, CA, Society for Design and Process Science.
Heckmann	2006	Heckmann Dominikus (2006) *Ubiquitous User Modeling.* PhD Thesis. ISBN 3-89838-297-4
Könik and Laird	2006	Könik, Tolga; Laird, John. 2006. "Learning goal hierarchies from structured observations and expert annotations". In *Machine Learning*, Volume 64, Numbers 1-3, September 2006 , pp. 263-287(25). Springer
Kopecky	2006	Kopecky, Karen A. "Value Function Iteration". In *Lectures notes Computational methods for macroeconomics* Fall 2006
Kwak et al.	2006	Kwak Jun-young ; Yoon Ji Young; Shinn Richard H. (2006) "An Intelligent Robot Architecture based on Robot Mark-up Languages". *IEEE International Conference on Engineering of Intelligent Systems*, Page(s):1 - 6. ISBN: 1-4244-0456-8
Lécué and Léger	2006	Freddy Lécué, Alain Léger. "A Formal Model for Semantic Web Service Composition". *International Semantic Web Conference* 2006: 385-398.
Lin and Hsu	2006	Lin, C. & Hsu, J. (2006)." IPARS: Intelligent Portable Activity Recognition System Via Everyday Objects, Human Movements, and Activity Duration". *AAAI Workshop on Modeling Others from Observations.* Pp.44-52

Macal and North	2006	Macal C.M., North M.J. (2006) "Tutorial On Agent-Based Modeling And Simulation Part2: How To Model with Agents". *IEEE Proceedings of the 2006 Winter Simulation Conference*. L.F.Perrone,F.P. Wieland,J.Liu,B.G.Lawson,D.M.Nicol,R.M.Fujimoto Eds.
Maurer et al.	2006	Maurer U., Smailagic A., Siewiorek D. and Deisher M.. "Activity recognition and monitoring using multiple sensors on different body positions". *Proceedings of the International Workshop on Wearable and Implantable Body Sensor Network*s, pp. 99-102. 2006.
Panton et al.	2006	Panton, Kathy, Cynthia Matuszek, Douglas Lenat, Dave Schneider, Michael Witbrock, Nick Siegel, and Blake Shepard. "Common Sense Reasoning – From Cyc to Intelligent Assistant". In Y. Cai and J. Abascal (Eds.): *Ambient Intelligence in Everyday Life*, LNAI 3864, pp. 1 – 31, 2006, Springer-Verlag Berlin Heidelberg 2006
Yu et al.	2006	Yu, Z., Zhou, Z., Yu, Z., Zhang, D., & Chin, C. (2006). An OSGI-based infrastructure for context-aware multimedia services. *Communications Magazine*, IEEE, 44(10), 136–142.
Bellifemine et al.	2007	Bellifemine, Fabio Luigi; Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE*. Wiley & Sons, 2007.
Bertoli et al.	2007	Piergiorgio Bertoli, Jörg Hoffmann, Freddy Lécué, Marco Pistore. "Integrating Discovery and Automated Composition: from Semantic Requirements to Executable Code". *ICWS 2007*: 815-822.
Daghan et al.	2007	Daghan Lemi Acay, Philippe Pasquier, Liz Sonenberg,"Extrospection: Agents Reasoning About the Environment", 3*rd IET International Conference on Intelligent Environments* (IE'07), University of Ulm, Germany, September 2007
Euzenat and Shvaiko	2007	Euzenat, Jérôme,Pavel Shvaiko. 2007. *Ontology Matching*. Springer-Verlag, Berlin Heidelberg (DE), 2007. ISBN: 3-540-49611-4; ISBN13: 978-3-540-49611-3
Gateau	2007	Gateau, Benjamin. Modélisation et Supervision d'Institutions Muli-Agents. PhD Thesis. CRP Henri Tudor. Luxembourg. 2007
Goodrich and Schultz	2007	Goodrich, M. A., and A. C. Schultz, "Human-Robot Interaction: A Survey," *Foundations and Trends in Human-Computer Interaction*, vol. 1, no. 3, pp. 203–275, 2007.
Kurup and Chandrasekaran	2007	Unmesh Kurup & B. Chandrasekaran, "Modeling Memories of Large-scale Space Using a Bimodal Cognitive Architecture." in *Proceedings of the International Conference on Cognitive Modeling* July 27-29, 2007, Ann Arbor, MI. (CD-ROM, 6 pages.)
Landragin	2007	Landragin Frédéric (2007) "Physical, semantic and pragmatics levels for multimodal fusion and fission". In: *Seventh International Workshop on Computational Semantics* (IWCS-7), Tilburg, The Netherlands, 2007, pp. 346-350.
Mastrogiovanni et al.	2007	Mastrogiovanni, Fulvio, Antonio Sgorbissa and Renato Zaccaria. 2007. "A Distributed Architecture for Symbolic Data Fusion". *IJCAI* 2007
Miraoui and Chakib	2007	Moeiz, Miraoui, and Chakib Tadj (2007) "A service oriented definition of context for pervasive computing". In *The 16th International Conference on Computing* (November, 2007). Mexico city, Mexico: IEEE Computer Society
Park et al.	2007	Park, Ji hwan, Song Tae Houn, Jung Soon Mook; Jeon Jae Wook (2007) "XML based robot description language". ICCAS apos;07. *International Conference on Control, Automation and Systems*, 2007. 17-20 Oct. 2007 Page(s):2477 - 2482
Pirrò and Talia	2007	Pirrò, Giuseppe, Domenico Talia. "UFOme: A User Friendly Ontology Mapping Environment" In *Proceedings of SWAP'2007*.
Sonntag et al.	2007	Sonntag Daniel, Engel Ralf, Herzog Gerd, Pfalzgraf Alexander, Pfleger Norbert, Romanelli Massimo, Reithinger Norbert." SmartWeb Handheld - Multimodal interaction with Ontological knowledge Bases and Semantic Web Services". In: *Artifical Intelligence for Human Computing: ICMI 2006 and IJCAI 2007 International Workshops* (2007) , p. 272-295.
Sun	2007	R. Sun, The importance of cognitive architectures: An analysis based on CLARION. *Journal of Experimental and Theoretical Artificial Intelligence*, Vol.19, No.2, p p.159-193. 2007 http://www.cogsci.rpi.edu/~rsun/clarion.html
Vaculin and Sycara	2007	Vaculín Roman, Sycara Katia (2007) "Specifying and Monitoring Composite Events for SemanticWeb Services". In *ECOWS '07, Fifth European Conference on Web Services*, 2007. Halle, Germany. ISBN: 978-0-7695-3044-4
Wang	2007	Wang Pei. From NARS to a thinking machine, *Advance of Artificial General Intelligence*, Pages 75-93, IOS Press, Amsterdam, 2007
Bach	2008	Bach, Joscha. 2008. "Seven Principles of Synthetic Intelligence". *Proceeding of the 2008 conference on Artificial General Intelligence 2008: Proceedings of the First AGI Conference* IOS Press Amsterdam, The Netherlands, The Netherlands, MicroPsi http://www.informatik.hu-berlin.de/~bach/artificial-emotion/
Bedini et al.	2008	Bedini, Ivan; Benjamin Nguyen, Georges Gardarin. "Janus: Automatic Ontology Builder from

		XSD Files". Poster Session of the *16th International Conference on Knowledge Engineering and Knowledge Management Knowledge Patterns* (EKAW), Acitrezza, Catania, Italy, September 2008.
Billard et al.	2008	Billard, A., Calinon, S., Dillman, R., Schaal, S., 2008. "Robot programming by demonstration". In: Siciliano, B., Khatib, O. (Eds.), *Handbook of Robotics.* Springer-Verlag, New York, NY, USA, Ch. 59.
Briola et al.	2008	Briola, D., A. Locoro, and V. Mascardi. "Ontology Agents in FIPA-compliant Platforms:a Survey and a New Proposal". In M. Baldoni, M. Cossentino, F. De Paoli,and V. Seidita, editors, *Proc. of WOA 2008: Dagli oggetti agli agenti, Evoluzione de ll'agent development: metodologie, tool, piattaforme e linguaggi.* Seneca Edizioni, 2008.
Coradeshi and Loutfi	2008	Coradeschi Silvia and Loutfi Amy (2008) "A review of Past and Future Trends in Perceptual Anchoring". In P.Fritze, editor, *Tools in Artificial intelligence*, I-Tech Education and Publishing. Vienna.
Cuenca Grau et al.	2008	Cuenca Grau, Bernardo; Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider, and Ulrike Sattler "OWL 2: The next step for OWL". *Journal of Web Semantics*, 6(4):309-322, November 2008, DOI: 10.1016/j.websem.2008.05.001
Gaitanis et al.	2008	Gaitanis Kosta, Gemo Monica, Vybornova Olga, Ruiz Diego, Moncarey Ronald and Macq Benoît (2008) "Cooperative Team Behavior Recognition for Multimodal Fusion". *IEEE Transactions on Multimedia*
Guiliani and Knoll	2008	Giuliani Manuel and Knoll Alois (2008) "MultiML - A General Purpose Representation Language for Multimodal Human Utterances".*ICMI'08*. Chania, Crete, Greece.
Karray et al.	2008	Fakhreddine Karray, Milad Alemzadeh, Jamil Abou Saleh and Mo Nours Arab. "Human-Computer Interaction An Overview on State of the Art". In *International Journal On Smart Sensing and Intelligent Systems,* Vol. 1, N0. 1, March 2008. http://www.docstoc.com/docs/45178338/Human-Computer-Interaction--An-Overview
Newbold et al.	2008	Newbold N., Vrusias B. and Gillam L. (2008) "Lexical Ontology Extraction Using Terminology Analysis: Automating Video Annotation", *The 6th Int. Conf. On Language resources & Evaluation (LREC)*, Morocco.
ODM	2008	*Ontology Definition Metamodel*. Object modeling Group. September 2008
Qi et al.	2008	Qi, Yu, Liu Xumin, Bouguettaya Athman, Medjahed Brahim (2008) "Deploying and managing Web services: issues, solutions, and directions". *The VLDB Journal.* Springer Berlin / Heidelberg. Volume 17, Number 3 / mai 2008
Ramos et al.	2008	Ramos, Carlos; Augusto, Juan Carlos; Shapiro, Daniel. "Ambient Intelligence—the Next Step for Artificial Intelligence" in *Intelligent Systems*, IEEE. Volume: 23 Issue:2 page(s): 15 - 18. march,21 2008
Serrano et al.	2008	Serrano, M., Nigay, L., Lawson, J.-Y.L., Ramsay, A., Murray-Smith, R., Denef. S.: The OpenInterface framework: a tool for multimodal interaction. In: *Adjunct Proceedings of CHI'2008* (April 5-10, Florence, Italy), ACM Press, pp. 3501--3506 (2008)
Sharp	2008	Sharp. "Implementing Decision Trees and Forests on a GPU". *ECCV 2008.*
Shvaiko and Euzenat	2008	Shvaiko, Pavel; Jérôme Euzenat. "Ten Challenges for Ontology Matching". OTM Conferences (2)1164-1182, 2008
Steinberg	2008	Steinberg A.N. (2008) "Foundations of Situation and Threat Assessment". In *Handbook of Multisensor Data Fusion, Theory and Practice, Second Edition. The Electrical Engineering And Applied Signal Processing Series* Edited by Alexander Poularikas. p438-496. ISBN: 978-1-4200-5308-1
Thomaz and Breazeal	2008	Thomaz,Andrea L. and Cynthia Breazeal. "Teachable Robots: Understanding Human Teaching Behavior to Build More Effective Robot Learners" in *Artificial intelligence* 2008, vol. 172, no6-7, pp. 716-737 Elsevier, Oxford, ROYAUME-UNI (1970) (Revue)
Alain et al.	2009	Alain, Pierre, Nelly Barbot, Vincent Barreaud, Laurent Blin, Olivier Boëffard, Laure Charonnat, Ali Choumane, Arnaud Delhay, Sébastien Le Maguer, Damien Lolive, Thierry Moudenc, and Gaëlle Vidal (2009). "A multi-agent platform for multimodal pervasive applications". In: *Proceedings of the Conference on the Networked and Electronic Media* (Nem), Saint Malo, France.
Allan	2009	Allan, Rob. "Survey of Agent-Based Modelling and Simulation Tools". In *Engineering.* Volume: 501, Issue: October, Pages: 57–72. June 3, 2009
Arrabales et al.	2009	Arrabales, R. Ledezma, A. and Sanchis, A. "CERA-CRANIUM: A Test Bed for Machine Consciousness Research". *International Workshop on Machine Consciousness*. Hong Kong. June 2009.
Benta et al.	2009	Kuderna-Iulian Benta, Amalia Hoszu, Lucia Vacariu, Octavian Cret. "Agent based smart house platform with affective control", paper 16, *EATIS'09,* June 3-5, 2009, Prague, CZ

Bessam and Kimour	2009	Bessam, Ammar; Mohamed Tahar Kimour. "Multi-view Meta-Modeling of Software Architecture Behavior." in *Journal of Software*, vol.4, N°5, July 2009.
Bettini et al.	2009	Bettini,Claudio;Oliver Brdiczka, Karen Henricksen, Jadwiga Indulska, Daniela Nicklas,Anand Ranganathan, Daniele Riboni. "A survey of context modelling and reasoning techniques". In *Pervasive and Mobile Computing* (2009), doi:10.1016/j.pmcj.2009.06.002
Besana et al.	2009	Besana, Paolo; Vivek Patkar, Adam Barker, David Robertson, David Glasspool "Sharing Choreographies in OpenKnowledge: A Novel Approach to Interoperability". In *Journal of Software* 4(8): 833-842 (2009) Brisbane, Australia
Billington et al.	2009	Billington,David, Vladimir Estivill-Castro, René Hexel, and Andrew Rock. "Architecture for hybrid robotic behavior". HAIS '09 *Proceedings of the 4th International Conference on Hybrid Artificial Intelligence Systems.* ISBN: 978-3-642-02318-7
Calvanese et al.	2009	Calvanese D., J.Carrol, Giacomo GD, J.Hendler, I.Berman, B. Parsia, P.F. Patel-Schneider, A.Ruttenberg, U.Sattler and M.Schneider. *OWL2 Web ontology Languages Profiles.* 2009
Dinarelli et al.	2009	Dinarelli M., Quarteroni S., Tonelli S., Moschitti A., and Riccardi G. "Annotating spoken dialogs: from speech segments to dialogacts and frame semantics", in *Proceedings of SRSL'09*, 2009.
Dumas et al.	2009	Bruno Dumas, Denis Lalanne, and Sharon Oviatt. 2009. Multimodal Interfaces: A Survey of Principles, Models and Frameworks. In *Human Machine Interaction*, Denis Lalanne and Jürg Kohlas (Eds.). Lecture Notes In Computer Science, Vol. 5440. Springer-Verlag, Berlin, Heidelberg 3-26. DOI=10.1007/978-3-642-00437-7_1 http://dx.doi.org/10.1007/978-3-642-00437-7_1
Evertsz et al.	2009	Evertsz, R., Pedrotti, M., Busetta, P., Acar, H., & Ritter, F. E. (2009). Populating VBS2 with realistic virtual actors. In *Proceedings of the 18th Conference on Behavior Representation in Modeling and Simulation*. 1-8. 09-BRIMS-04.
Gemrot et al.	2009	Gemrot, J., Kadlec, R., Bida, M., Burkert, O., Pibil, R., Havlicek, J., Zemcak, L., Simlovic, J., Vansa, R., Stolba, M., Plch, T., Brom C.: Pogamut 3 Can Assist Developers in Building AI (Not Only) for Their Videogame Agents. In: *Agents for Games and Simulations*, LNCS 5920, Springer(external link), 2009, pp. 1-15
Gluck and Pew	2009	Gluck,Kevin A. & Richard W. Pew. *Modeling Human Behavior With Integrated Cognitive Architectures. Comparison, Evaluation, and Validation.* LAWRENCE ERLBAUM ASSOCIATES, PUBLISHERS London. ISBN: 0-8058-5047-3
Goertzel	2009	Goertzel, Ben (2009). OpenCogPrime: A Cognitive Synergy Based Architecture for Embodied General Intelligence, *Proceedings of ICCI-2009*. See also http://opencog.orr
Goumopoulos and Kameas	2009	Goumopoulos,Christos, Achilles Kameas "A Service-Oriented Architecture Combining Agents and Ontologies Towards Pervasive Adaptation" in *5th International Conference on Intelligent Environments* (IE'09), Series: Ambient Intelligence and Smart Environments IOS Press, Spain, 2009
Gupta et al.	2009	Gupta Abhinav, Kembhavi Aniruddha, Davis L.S. (2009) "Observing Human-Object Interactions: Using Spatial and Functional Compatibility for Recognition". *IEEE Transactions On Pattern Analysis And Machine Intelligence,* Vol. 31, No.10, pp. 1775-1789, Oct 2009
Johnston	2009	Johnston Michael. "Building Multimodal Applications with EMMA". *ICMI-MLMI'09.* November 2-4, 2009, Cambridge, MA, USA. Copyright 2009 ACM 978-1-60558-772-1/09/11
Johnston et al.	2009	Johnston Michael, Baggia Paolo, Burnett Daniel C., Carter Jerry, Dahl Deborah A., MacCobb Gerry and Ragget Dave (2009) "EMMA: Extensible MultiModal Annotation markup language". *W3C Recommendation* 10 February 2009
Kameas et al.	2009	Kameas,Achilles, Christos Goumopoulos, Tobias Heinroth, Wolfgang Minker, Apostolos Meliones, Dimitris Economou. Yacine Bellik, Gaëtan Pruvost. "A Pervasive System Architecture that supports Adaptation using Agents and Ontologies". ispan, pp.148-153, 2009 10th *International Symposium on Pervasive Systems, Algorithms, and Networks*, 2009.
Kieras	2009	Kieras, D. (2009). The persistent visual store as the locus of fixation memory in visual search tasks. In A. Howes, D. Peebles, R. Cooper (Eds.), *9th International Conference on Cognitive Modeling* – ICCM2009, Manchester, UK. http://www.umich.edu/~bcalab/epic.html
Lallee et al.	2009	Lallee S, Warneken F, Dominey PF (2009) "Learning to Collaborate by Observation" in *International Conference on Epigenetic Robotics*
Nobre et al.	2009	Nobre, Farley-Simon, Andrew M. Tobias, and David S. Walker. "Rationale for Cognitive Machines". *In Organizational and Technological Implications of Cognitive Machines: Designing Future Information Management Systems*. 2009 DOI: 10.4018/978-1-60566-302-9.ch004,ISBN13: 9781605663029,ISBN10: 1605663026
Pareiras et al.	2009	Pareiras F.Silva, T.Walter, S.Staab, C.Saathoff, T.Franz, APIs agogo: Automatic Generation of Ontology APIs, In *Proceedings of the 2rd IEEE International Conference on Semantic Computing*, Santa Clara, 2009

Sabouret et al.	2009	Sabouret, Nicolas and Benazzouz, Yazid and Chikhaoui, Belkacem (2009) "Dynamic Service Composition in Ambient Intelligence Environment". In *IEEE International Conference on Services Computing*, pp. 411--418.
Sanner and Boutilier	2009	Sanner,Scott , and Craig Boutilier. 2009. "Practical solution techniques for first-order MDPs". *Artificial Intelligence* 173, 5-6 (April 2009), 748-788
Shapiro and Bona	2009	Stuart C. Shapiro and Jonathan P. Bona, The GLAIR Cognitive Architecture. In Alexei Samsonovich, Ed., *Biologically Inspired Cognitive Architectures-II: Papers from the AAAI Fall Symposium*, Technical Report FS-09-01, AAAI Press, Menlo Park, CA, 2009, 141-152.
Soylu et al.	2009	Soylu, Ahmet; Patrick De Causmaecker, Piet Desmet. "Context and Adaptivity in Pervasive Computing Environments: Links with Software Engineering and Ontological Engineering". In *Journal of Software* 4(8): 992-1013 (2009) Brisbane, Australia
Sun	2009	R. Sun, Theoretical status of computational cognitive modeling . *Cognitive Systems Research*, Vol.10, No.2, pp.124-140. 2009.
Sterling and Taveter	2009	Sterling, Leon;Kuldar Taveter. *The Art of Agent-Oriented Modeling*. MIT Press 2009
Wooldridge	2009	Michael Wooldridge. *An Introduction to MultiAgent Systems - Second Edition*. Eds John Wiley & Sons. ISBN-10:0470519460 - ISBN-13:978-0470519462
Zarri	2009a	Zarri Gian Piero (2009) "Representation and Processing of Complex Events, Association for the Advancement of Artificial Intelligence" *AAAI Spring Symposium*.
Zarri	2009b	Zarri Gian Piero (2009) *Representation and Management of Narrative Information, Theoretical Principles and Implementation*. Series: Advanced Information and Knowledge Processing 2009, X, 302 p. 55 illus., Hardcover. ISBN: 978-1-84800-077-3 Springer-Verlag London
Abran	2010	Abran Alain (2010) "Software Metrics and Software Metrology". Wiley and sons, inc. Indiannapolis, USA. ISBN: 978-0-470--59720--0
Artikis et al.	2010	Artikis, Alexander, Georgios Paliouras, François Portet nad Anastasios Skarlatidis. "Logic-Based Representation, Reasoning and Machine Learning for Event Recognition" in DEBS'10, July 12-15, 2010, Cambridge, UK.
Augusto et al.	2010	Juan Carlos Augusto, Hideyuki Nakashima, Hamid Aghajan. "Ambient Intelligence and Smart Environments:A State of the Art". *Handbook of Ambient Intelligence and Smart Environments.* Page 3-31
Ben Amor	2010	Ben Amor, Heni. *Imitation Learning of Motor Skills for Synthetic Humanoids*. PhD Thesis Dissertation 12.11.2010
Bothell	2010	Dan Bothell. *ACT-R Reference Manual.* http:/ /act-r.psy.cmu.edu/actr6/reference - manual . pdf (visited on 07/27/2010).
David et al.	2010	Jérôme David, Jérôme Euzenat, François Scharffe, Cássia Trojahn dos Santos. 2010. "The Alignment API 4.0". *Semantic Web*. Publisher IOS Press
Epstein and Petrovic	2010	Epstein, S. L. and S. Petrovic. 2010 Learning Expertise with Bounded Rationality and Self-awareness. In *Autonomous Search.* Y. Hamadi, E. M., F. Saubion, editors. Springer.
Gobet and Lane	2010	Gobet, F., & Lane, P. C. R. (2010). The CHREST architecture of cognition: The role of perception in general intelligence. In *The Third Conference on Artificial General Intelligence*. Lugano, Switzerland. http://www.psyc.nott.ac.uk/research/credit/projects/CHREST
Grigori et al.	2010	Grigori, Daniela, Juan Carlos Corales, Mokrane Bouzeghoub and Ahmed Gater. "Ranking BPEL Processes for Service Discovery" In IEEE Transactions on Services Computing, pages 640-641 (ICWS) February, 17 2010
Hadjiski et al.	2010	Mincho Hadjiski, Vassil Sgurev, Venelina Boishina. "Intelligent control of uncertain complex systems by adaptation of fuzzy ontologies". In *Intelligent systems: from theory to practice.Studies in Computational Intelligence* Eds: Mincho Hadjiski, Vassil Sgurev, Janusz Kacprzyk. 2010 Springer Verlag
Hoet and Sabouret	2010	Hoet, S., N. Sabouret. "Apprentissage par renforcement d'actes de communication dans un système multi-agents", In *Revue d'Intelligence Artificielle* (RIA), vol. 24:2, pp. 159-188, 2010.
Kapahnke et al.	2010	Kapahnke,Patrick, Pascal Liedtke, Stefan Nesbigall, Stefan Warwas, and Matthias Klusch. "ISReal: An Open Platform for Semantic-Based 3D Simulations in the 3D Internet". In *9th International Semantic Web Conference* (ISWC 2010) Shanghai, China.
Kim et al.	2010	Kim Eunju, Helal Sumi and Cook Diane. "Human Activity Recognition and Pattern Discovery". *IEEE Pervasive Computing*, 2010
Mendel et al.	2010	Mendel, Jerry M., Lofti A. Zadeh, Enric Trillas, Ronald Yager, Jonathan Lawry, Hani Hagras, Sergio Guadarrama. What Computing with Words Means to Me. In *IEEE Computational Intelligence Magazine*. Feb. 2010

Mileo et al.	2010	Mileo Alessandra, Merico Davide and Bisiani Roberto "Support for context-aware monitoring in home healthcare". *Journal of Ambient Intelligence and Smart Environments* 2 pp 49–66 49 DOI 10.3233/AIS-2010-0052 IOS Press (2010)
OCL	2010	*Object Constraint Language formal v2.2*, Object modeling Group, http://www.omg.org/spec/OCL/2.2, February 2010
Pacuit and Roy	2010	Pacuit,Eric and Olivier Roy. "Individual and Collective Agency". In *European Summer School in Logic, Language and Information 2010 (ESSLLI'10)*
Pirrò and Talia	2010	Pirrò, Giuseppe and Domenico Talia. 2010. UFOme: An ontology mapping system with strategy prediction capabilities. In *Data Knowledge Engineering* 69, 5 (May 2010), 444-471. DOI=10.1016/j.datak.2009.12.002 http://dx.doi.org/10.1016/j.datak.2009.12.002
Russel and Norvig	2010	Russell,Stuart, & Norvig Peter, *Artificial Intelligence: A Modern Approach (3rd Edition)* Prentice Hall; 3 edition (December 11, 2009). ISBN-10: 0136042597 ISBN-13: 978-0136042594
Sakarkar and Upadhye	2010	Sakarkar, Gopal, and Sachin Upadhye. "A Survey of Software Agent and Ontology". *International Journal of Computer Applications* 1(7):18–23, February 2010. Published By Foundation of Computer Science.
Sanner and Kersting	2010	Sanner, Scott, Kristian Kersting. "Symbolic Dynamic Programming for First-order POMDPs" in AAAI 2010
Subercaze and Maret	2010	Subercaze Julien, Pierre Maret. 2010, "Semantic Agent Model for SWRL rule-based agents" in ICAART 2010, INSTICC Press 2010 ed. Valencia, Spain. pp. 244-248. *Proceedings of the International Conference on Agents and Artificial Intelligence*, Volume 2 - Agents . ISBN 978-989-674-022-1.
Tecuci et al.	2010	Tecuci G., Schum D.A., Boicu M., Marcu D., Hamilton B., Intelligence Analysis as Agent-Assisted Discovery of Evidence, Hypotheses and Arguments. In: Phillips-Wren, G., Jain, L.C., Nakamatsu, K., Howlett, R.J. (eds.) *Advances in Intelligent Decision Technologies*, SIST 4, pp. 1-10. Springer-Verlag, Berlin Heidelberg, 2010.
Vashist and Loeb	2010	Akshay Vashist, Shoshana Loeb (2010), Attention Focusing Model for Nexting Based on Learning and Reasoning. in *BICA 2010*. (Submitted).
Becerra and Kremer	2011	Becerra, Gabriel, Rob Kremer. "Ambient intelligent environments and environmental decisions via agent-based systems". *Journal of Ambient Intellingence Human Computer*. DOI 10.1007/s12652-011-0056-0, 12 june 2011
Diamos et al.	2011	Gregory Diamos, Benjamin Ashbaugh, Subramaniam Maiyuran, Andrew Kerr, Haicheng Wu and Sudhakar Yalamanchili: *"SIMD Re-convergence at Thread Frontiers"*. In *44th International Symposium on Microarchitecture* (MICRO 44), 2011
Dindo et al.	2011	H. Dindo, A. Chella, G. La Tona, M. Vitali, E. Nivel, and K. R. Thórisson. "Learning problem solving skills from demonstration: An architectural approach", in *Proceeding of the Fourth Conference on Articial General Intelligence* (AGI-11), Google, Mountain View, California, USA, August 3-7, 2011
Farabet et al.	2011	Farabet, C. , B. Martini, B. Corda, P. Akselrod, E. Culurciello and Y. LeCun, "NeuFlow: A Runtime Reconfigurable Dataflow Processor for Vision", in *Proceeding of the Fifth IEEE Workshop on Embedded Computer Vision* (ECV'11 @ CVPR'11), IEEE, Colorado Springs, 2011.
Gravier et al.	2011	Gravier C., Y.-G. Billet, J. Subercaze, B. Jailly, J. Fayolle, P. Maret, J. Lardon, "Reasoning with Context: Autonomic system for A-Box revisions in Semantic Context-Aware Systems". Submitted to *Third International Workshop on Automated Reasoning about Context and Ontology Evolution*. (2011)
Kaelbling and Lozano-Perez	2011	Leslie Pack Kaelbling and Tomas Lozano-Pérez. "Hierarchical Task and Motion Planning in the Now". IEEE *Conference on Robotics and Automation (ICRA)*, May 2011
Legendre	2011	Legendre, Marie-Françoise. *Piaget et l'épistémologie*. Fondation Jean Piaget. 2011
Oltramari and Lebiere	2011	Alessandro Oltramari, Christian Lebiere. "Extending Cognitive Architectures with Semantic Resources" in Proc. of the Fourth Conference on Arti cial General Intelligence (AGI-11), Google, Mountain View, California, USA, August 3-7, 2011
Oudeyer	2011	Oudeyer, P-Y. (2011) Developmental Robotics, Encyclopedia of the Sciences of Learning, N.M. Seel ed., Springer Reference Series, Springer.
Reckman et al.	2011	Reckman,Hilke,Jeff Orkin, and Deb Roy. "Extracting aspects of determiner meaning from dialogue in a virtual world environment". *IWCS2011*
Rohrer	2011	B Rohrer, "An implemented architecture for feature creation and general reinforcement learning," Workshop on Self-Programming in AGI Systems, *Fourth International Conference on Artificial General Intelligence*, Mountain View, CA, Aug 3-6, 2011. (1MB pdf) http://www.sandia.gov/rohrer/doc/Rohrer11ImplementedArchitectureFeature.pdf

Sabri et al.	2011a	Sabri, Lyazid, Chibani, Abdelghani, Amirat, Yacine and Zarri, Gian Piero, "Semantic and architectural approach for Spatio-Temporal Reasoning in Ambient assisted Living", in: *International Joint Conference on Artificial Intelligence* (IJCAI 2011), Barcelona, Spain, pages 77-84, 2011
Sabri et al.	2011b	Sabri, Lyazid, Chibani, Abdelghani, Amirat, Yacine and Zarri, Gian Piero, "Semantic reasoning framework to supervise and manage contexts and objects in pervasive computing environments", in: *Proceedings of the International Conference on Advanced Information Networking and Applications* (AINA 2011), Biopolis, Singapor, pages 47-52, 2011
Samsonovich	2011	http://bicasociety.org/cogarch/architectures.xls
Subercaze and Maret	2011	Subercaze Julien , Pierre Maret. "Virtual Knowledge Communities for Semantic Agents". In *Proceeding of the Web Intelligence, Mining and Semantics Conference.* Norway (2011)
Thorisson and Helgason	2011	Thórisson, K. R. & H. P. Helgason (2011). *Review of Cognitive Architectures, With Suggestions for Future Directions*.

GLOSSARY

Action

Realization of a will or an intention. Exercise capacity to act. (Dictionary) Something that an agent does by sending an order to a service. The most primitive unit of activity. Primitive actions are executed by services (web services with SOAP or REST). Compound actions are defined by a sequence or parallel sequences of actions of lower level in a graphical flow.

Activity

An activity uniquely defined a set of rules, i.e., a set of models that relates to a distributed activity.

Agent

A piece of software that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators. An agent is an object that contains methods to communicate with environment and other agents. In our case, agent is a part of a global semantic architecture.

Agent-based approach

Generalises object-oriented software engineering.

Agent-oriented software engineering methodology

A kind of software engineering methodology that uses the notion of agent or actor in all stages of its software engineering process

Agent's behaviour

It is described by the agent function that maps any given percept sequence to an action or a new produced event.

Agent function

The agent function is implemented by an agent program. This agent function is an abstract mathematical description; the agent program is a concrete implementation of the agent function running on the agent architecture.

Aggregation

During matching of inference, aggregation allows filling roles of composite predicate with the combination of operations on concepts or models in the arguments of these roles.

Anchorage

Memory associated to physical sense/perception

Artificial Intelligence

An observation of intelligence (conceptual reasoning or behaviour) performed by a non natural system.

Attribute

A characterization of a physical agent or physical object in one or more quality dimensions.

Autonomous agent

A kind of agent that creates and pursues its own job as opposed to functioning under the control of another agent.

Behaviour

A behaviour is a reified piece of activity in which a system engages in itself or in the environment. A system behaves in various ways. A behaviour is a model of event or an instance of event in our agent.

Beliefs

The facts and rule models possessed by an agent.

Communication

A kind of physical action where an agent or a service sends a message to another agent or another service.

Component

In modelling, a component is an object or an agent modelling following an UML design. Components have several levels of abstraction from a global component (composed) to a single component. An UML diagram depicts how a software system is split up into components and shows the dependencies among these components. Our Memory component can be a single component or integrated in an agent component.

Composition of services

Service composition can be defined as the process of combining existing services to form new services". With the property of loose coupling of services, service-oriented programming allows to easily perform such compositions. In the world of web services, there are two approaches to design compositions: orchestrations and choreography.

Choreography describes the flow of messages exchanged by a Web service when interacting with other services. It is therefore in a decentralized way of managing a Web service composition as each actor is responsible for part of the workflow. In particular, the standard Web Services Choreography Interface (WSCI) to specify the behavior of Web service from the rest of the composition.

Orchestration comprises a central element responsible for the composition as a whole. The process then becomes the sum of its sub-processes and the entity responsible for the composition, called "orchestrator" only manages message exchanges. Many studies have focused on execution languages of business processes, including WSFL from IBM and XLANG from Microsoft. These efforts were then merged into a common specification named BPEL4WS9 (called BPEL for short). BPEL is now an essential standard for the specification and implementation of service compositions. This is particular to this language that has turned the efforts of the industry by developing multiple implementation technologies of BPEL (Microsoft BizTalk Server, IBM BPWS4J, Oracle BPEL Server, etc.).

Computational Intelligence

Computerized mechanisms, methods or paradigms of the artificial intelligence like fuzzy logic (vague reasoning), artificial neural networks (machine learning), computing with words (Zadeh 2010) and so on.

Concept

A kind of knowledge item used for mentally representing an entity. In a knowledge representation language or an ontology representation system, a concept is a system of general ideas targeting a domain of knowledge or philosophy in a well defined context. Concepts are words that bring meaning to a sentence. Concepts are used to reason and permit to categorize any items, spatial and temporal perception of the world of experience at different scales, from more global to more specialized. Concepts can also be seen as picture or metaphors in our brain. Concept can be abstract (mental) or concrete (physical). A concept in an ontology of domain is like an object with properties and attributes.

Concept Instance

Concept instance are all possible names of things owning all common points matching the concept.

Context

Contexts are propagated from agents to agents. They help the understanding by limiting event/action to a possible meaning. A scientific domain or a class of concepts can be also used as a context.

Coordination

Coordinations are characterized by inferences, expressed or implied, that the subject sees or uses as if they are imposed on him, with all intermediaries between the subjective obvious and logical necessity. The criterion of necessary or required pseudo-inferences is that this is not simply inductive generalizations (...) but to build new relationships: for example, anticipating that the shock of a ball A against a ball B will always be followed by a motion of B will not be called "coordination", while this term applies to the assumption of a transmission such as the "momentum" of A past to B, because a transmission of movement is never observable in itself. (Piaget 1975)

Culture

"Culture consists in the shared patterns of behavior and associated meanings that people learn and participate in within the groups to which they belong" (Whitten and Hunter, 1976)

In the context of this research, a more cognitive definition of culture is desired, with the focus on behavioral rules, knowledge, norms and meanings. A culture is a configuration them which are shared and transmitted by members of a particular group and therefore understandable and acceptable to its members.

Deduction

Find a specific rule. The deduction proceeds from the conception that the means are more important than the end (conclusion). Incorrect deduction often comes from a missing relationship in the assertion like, for example, membership between two concepts used in the assertion.

Denotation

It refers to what the sign refers and can be found in the dictionary or encyclopedia.

Bertrand Russell (1872-1970) is by the importance of his philosophical work and his political commitment, one of the greatest intellectuals of the twentieth century. By "language denotation" he means an expression similar to any of the following expressions: a man, any man, every man, all men, the present king of France, ... Russell will make a distinction between *direct knowledge* and *knowledge about*, to distinguish the things we have mental representations, things that we attain only through language denotation. Moreover, it often happens

that we know that a certain phrase denotes unambiguously, although we have no direct knowledge of what it denotes: this is what happens in the case of center of mass. In perception, we have direct knowledge of the objects of perception, and we have in mind a direct knowledge of objects of a more abstract sense, but we do not have necessarily knowledge about these objects that denote the expressions composed of words which we know the meaning through direct knowledge. Here is an interesting example: there seems no reason to believe that we know directly the minds of others as they are not perceived directly, too, what we know about them, we get by means of denotation. *Think* always seems to take its starting point in a direct knowledge, but we have thought about many things which we have no direct knowledge. Everything, nothing and something are assumed to have no meaning in isolation, but a meaning is assigned to each proposal in which they appear. The principle of the theory of denotation that it wishes to defend is the following: the denoting expressions never meaningless in themselves, but each proposal in the verbal expression which they have found a way. The difficulties are related to the denotation, in his opinion, the result of poor analysis of propositions whose verbal expressions contain denoting expressions. What speaks in favor of this theory, it is the difficulties inevitably encounter when you consider that the denoting expressions represent the genuine components of proposals in the verbal expression which they appear. Of course, there is the theory of Frege which avoids precisely this violation of the law of contradiction. He distinguishes, in a denoting expression, two elements that we can call *meaning* and *denotation*. One of the benefits between meaning and denotation is that this distinction permits to show why it is often useful to assert identity. So to recap: you can choose the first solution assuming, as Meinong, objects that do not subsist, and denying that they obey the law of contradiction. But it is required, wherever possible, avoid it. Another way to defend the same solution is pursued by Frege providing, by means of a definition, a purely conventional denotation for cases where there would otherwise be none. But, however, the relationship between meaning and denotation is not a purely linguistic relationship; a logical relationship is necessarily involved, which we express by saying that the meaning denotes the denotation. From another point of view, the correct way to speak is to say that some meanings have denotations. This theory of denotation has an interesting consequence: when we have no immediate direct knowledge of something, but only one definition by means of denoting expressions, the proposals where this thing is introduced by a denoting expression do not really count this thing among their constituents, but contain instead the constituents expressed by different words in the denoting expression. Also in each proposal that we can understand (that is to say, not just those of the truth or falsity of which we can judge, but all the ones we can think of), all components are real entities which we have direct knowledge. Now we do not know such things as the material (in the sense that the material was in physics) or the minds of others, than by language indicating, that is to say that we do not know not directly, but only as what has such and such properties. We do not know directly who say the proposals that we know these things have to be true, because we cannot apprehend the actual entities in question. Also in each proposal that we can understand (not just those of the truth or falsity of which we can judge, but all the ones we can think of), all components are real entities which we have direct knowledge. But we do not know such things as the material (in the material meaning in physics) or the minds of others, than by denoting expressions, that we do not know not directly, but only as what has such and such properties. We do not know directly who say the proposals that we know these things have to be true, because we cannot apprehend the actual entities in question.

Entity

Any concepts observable, perceivable, conceivable or mentally manipulable.

Environment

A first class abstraction that provides the surrounding conditions for agents to exist and that mediates both the interaction among agents, services and systems and the access to resources.

Epistemology derived from logic

Piaget distinguishes six major varieties of epistemology derived from logic, as they are based on interpretations of transcendental (1-3) or natural (4-6) the logic and whether they focus on experimental data (1 and 4), on already established structures (2 and 5) or constructive interactions (3 and 6). This classification allows it to locate the constructivist point of view, which is his own, compared to other common epistemological:

1. The Platonic conception of logic is an example of transcendental interpretation focused on objective data about the discovered through the ability to "design".
2. The apriorism is a transcendental interpretation focuses on the structures of the subject.
3. The transcendental phenomenology invokes an interaction between subject and object and uses an intuitive "pure" essence.
4. The empiricist interpretation of logic is to reduce the logic to read data from experience, that it is internal or external.
5. The nominalist interpretation of logical positivism reduces the logic of linguistic structures or conventions.
6. Finally, the interactionist and constructivist interpretation of Piaget is the axiomatization of the logic of the activities and operational structures of the subject.

(Legendre 2011)

Event

An event expression (a formal logical frame) defines when a rule should be applied. It is a predicate that is true when it appears (i.e. it is recognized) or it must be done (i.e recognition of an event of higher level by fusion or an action to be executed as a postcondition of a rule). An event expression is a model.

Event Instance

An event instance is a happened event, action or scenario (composite event) matching an event model. We may also call them “facts”.

Event Model

A class of models representing an event. An event model or action model is a predicate in a defined state, situation or context.

Fact

We can consider the "fact" - a property of an action or of any event - as observed from the time it is "interpreted", that is wearing a relative meaning on a broader context, while a single observable (for any equation already that gives a meaning) can remain entirely local in space and even time. One fact is in turn always the product of the composition between a share contributed by the objects and another built by the subject. The intervention of the latter is so important that it can even be a distortion or even a discharge of observance, which then distorts made according to the interpretation. (Piaget, 1983)

A "fact" presupposes interpretations implied from the position of the problem and from the finding, but it is a scientific fact that if it leads to (...) an explicit interpretation that ensures understanding. (Piaget, 1974a)

To achieve the same reading of the facts, the subject must be in possession of schemes to assimilate them, not even in the sense of assimilation explanatory, but a simple recognition as a given. (...) To connect a data collected at the following (and in this relationship is essentially the same as inductive generalization), it must be in possession of deductive models (...) the first work of the mind, to make given the passage of the law (and thus to achieve induction), is therefore to build new patterns represent (and sometimes first perceptual), may allow the same data record, which remains impossible without them but to build new schemes, it is to link the data with each successive, and therefore, first, to save them properly. (Piaget, 1950)

(...) the subject is not called as pure experience in any cognitive development (...) any observable is always interpreted and any "fact" necessarily implies an interaction between subject and object in question. These results in all areas of knowledge, exactly as though mistaken, have an inferential aspect and for each cognitive area (...) the contribution of the subject is beyond dispute (...). (The) "facts" (...) are naturally achieved by approximations (trial and error), and even with the obligation to abandon any property which appeared constitutive, but the maintenance would have contradicted the new one. (...) The "facts", despite their obvious links with external sources (immediate or mediate), cannot be independent of endogenous structures through which the subject interprets a mixture of continuous findings and inferences (the first preceding the second or vice versa). (Piaget, 1983)

Fission

Decomposition of a high level scenario or global context in events of lower level of abstraction or execution orders by combination of concepts and model of events. And selection of output components.

Fluent

Fluents are (logical) propositions whose value is subject to change over time.

Frame Problem

The *frame problem* as an issue in Cognitive Architectures can be described as the task of any agent acting in a dynamic environment to keep its model of the world and its knowledge in general in synchrony with the world. In the case of detecting changes and asserting them perceptually, the problem is logically trivial. However, the effect of these changes on derivational knowledge and on the state of goals and strategies is non-trivial and not understood.

Functional Architecture

A *Functional Architecture* represents the constraints of a hypothesis or model of the network of functional areas in the brain that makes different modules or components interact.

System Architecture is a representation of a system in which there is a mapping of functionality onto hardware and software components, a mapping of the software architecture onto the hardware architecture, and human interaction with these components (Carnegie Mellon Software Engineering Institute. Software engineering glossary. October 2006).

Fusion

Composition of events of higher level of abstraction by aggregation or combination of concepts and model of events. And selection of input components.

Generalization

A taxonomic formal relation between more general and more specific types of entities.

Generic component

A generic component is a primitive or basic component which we specialize to create more specific component.

Goal

A 'goal' is a token which represents at a high level something which the agent is trying to achieve. Generally speaking, a goal is represented as a state of the world which the agent would like to see happen: for example, that the car is parked without denting anything.

Induction

Find a general rule from observation (more often a probabilistic way). However, induction does not prove the certainty of the rule because an immediate counterexample will break it.

Integration Architecture

An *integration architecture* deals with the structural composition of software components into a system instance. It provides design elements which bind domain functionality provided by software components to artefacts of the integration architecture, thereby exposing their services to other components. It provides design patterns for the composition of design elements and establishes guidelines for the selection among design alternatives. It provides functionality for physical distribution, communication, synchronization and coordination between design elements and functionality for data access within the architecture.

Interaction

Reciprocal action, effect, or influence of systems (dictionary)

Lexeme

Smallest unit of information that is meaningful.

Meaning

Interpretation of a word, a symbol or an expression. It can be "giving the possible senses or "the concept that a signifier denotes". In fact, this definition is not clear at all because a cognitive process consists to find relationships with words well known/defined of the applicant.

"*In the case of the simpler kind of propositions, namely those that I call "atomic" propositions, where there is only one word expressing a relation, the objective which would verify our proposition, assuming that the word "not" is absent, is obtained by replacing each word by what it means, the word meaning a relation being replaced by this relation among the meanings of the other words.*" (Russel, 1921)

We define meaning as a function realizing:

- fusion (composition) from lower level of input data to higher level of abstraction of information, or
- fission (decomposition) from higher level to lower level of action or event. This function is executed by the inference engine of the agent.

Meaning often results of a need of clarification or equivalence.

Meta Level

More abstract level. A meta level contains the vocabulary and the semantics necessary to explain the level just below. Each meta level subsumes the below level and then explains the understanding (description/decomposition of a level meaning by the low levels in a relation of generalization (down-up) or specialization (top-down)). An equivalence relationship may also permit to explain a class of concept by analogy if the equivalent class is first clearly understood (thus well described by a subsumption relationship). In brief, it's a denotation.

Modal

Modals — words that express modalities — qualify a statement.

Modal logic

It is a type of formal logic primarily developed in the 1960s that extends classical propositional and predicate logic to include operators expressing modality.

Model

A kind of knowledge item used for mentally representing a behaviour. structure that gives meaning to the sentences of a formal language is called a model (Structure of mathematical logic: In universal algebra and in model theory, a structure consists of a set along with a collection of finitary functions and relations which are defined on it) for the language. If a model for a language moreover satisfies a particular sentence or theory (set of sentences), it is called a model of the sentence or theory.

Modifier

A modifier specifies the event or fact. It is used by inference engine to activate or not a rule (i.e. a model). It can be temporal, spatial, negative, modal, and so on.

Motivation

It conducts to use a tool or to do something (act).

Multiagent system

A kind of system where several, perhaps all, of the connected entities are agents.

Multimodal Interaction

Interaction of several input and output modalities (like human senses: vision, audition, touching, smelling, gustativeness, any sensors and any actuators). Different types of interaction exist like communication, dialog, feedback or physical contact. Multimodal interaction refers to modalities management mechanisms like composition, selection, orchestration, conversion, fusion and fission. Multimodal interaction can also refer to interoperation on several modalities.

Multimodal Logic

Applied to NLP, it can be seen as an application of a preposition in a NL sentence. Most frequent prepositions types are: rank (in front of, behind, after), location (in, into, at, to, under), time (before, after, at, since), cause (for, because, seeing that), manner (with, without, within, according to), goal (in order to, for, to), separation (without, except, unless, apart from, but). Rank can be seen as a location in a list of objects. We call them modifiers too because they modify the interpretation of the sentence and then must be taken into the inference engine. Other modifiers like emotions, feelings and so on are defined in the meta ontology. Nuances or concept types like matter, quantity, price, weight, distance, and so on are specific roles also defined in the meta ontology.

Negation

Rejection, prohibition, self-prohibition, disappearance, unfulfilled expectation, truth functional denial.

Object

In programming, an object is a language mechanism for binding data (also called attributes) with methods (also called properties) that operate on that data. Generally, an object is an instance of a class or array. In physical environment, an object can be mobile or not but it has no communication skills and no thinking abilities. On the contrary, it can have different functions depending of use done by an agent

Observable

We consider, in fact, as observed anything that can be saved by a simple statement of fact (or empirical: a singular event, a relationship repeatable, momentary or even a regular covariation) allowing to speak of functional dependence or law. In this broad sense, a relationship or a regular function of two observables is still observable. (...) By cons, we reserve the term *inferential coordination* to unrecognized connections but deduced by operating composition (and not by simple extensional generalization) and exceeding the scope of observable, especially as introducing necessity relationships, for example, coordination based on transitivity (...). (Piaget, 1974b)

Ontology

A structure of knowledge of the problem domain. A domain ontology is a taxonomy added to relationships like subsumption, equivalence and specialization. The memory of our agents comprises 3 ontologies: a meta ontology to build the two other ontologies, an ontology of concepts (common model of domain ontology) and an ontology of event models (frame-based). For an inference engine optimization purpose, they are tree-structured.

Percept

Agent's perceptual input at any given instant. In our case, a fact or an event.

Percept sequence

History of everything the agent has ever perceived.

Predicate Logic

In mathematical logic, predicate logic is the generic term for symbolic formal systems like first-order logic, second-order logic, many-sorted logic or infinitary logic. This formal system is distinguished from other systems in that its formulae contain variables which can be quantified. Two common quantifiers are the existential ∃ ("there exists") and universal ∀ ("for all") quantifiers. The variables could be elements in the universe under discussion, or perhaps relations or functions over that universe. For instance, an existential quantifier over a function symbol would be interpreted as modifier "there is a function".

Predicate calculus symbols may represent either variables, constants, functions or predicates. Constants name specific objects or properties in the domain of discourse. Thus George, tree, tall and blue are examples of well formed constant symbols. The constants $\top$ (true) and $\bot$ (false) are sometimes included. Variable symbols are used to designate general classes or objects or properties in the domain of discourse. Functions denote a mapping of one or more elements in a set (called the *domain* of the function) into a unique element of another set(the *range* of the function). Elements of the domain and range are objects in the world of discourse. Every function symbol has an associated *arity*, indicating the number of elements in the domain mapped onto each element of range. A *function expression* is a function symbol followed by its arguments. The arguments are elements from the domain of the function; the number of arguments is equal to the arity of the function. The arguments are enclosed in parentheses and separated by commas. e.g.: f(X,Y), father(james), price(fruit) are all well-formed function expressions. Predicate logics may be viewed syntactically as Chomsky grammars. As such, predicate logics (as well as modal logics and mixed modal predicate logics) may be viewed as context-sensitive, or more typically as context-free, grammars. As each one of the four Chomsky-type grammars have equivalent automata, these logics can be viewed as automata just as well.

Process

A complex event that consists of two or more possibly parallel occurrences of events.

Protocol

A kind of interaction model that represents an interaction pattern between agents of two or moreagent types along with the aspects of the agents' behavior.

Quality

A kind of moment that inheres in exactly one endurant and can be represented in several quality dimensions.

Quality Attribute

A characterization of the quality of performing an activity with respect to some quality goal.

Query Model

A class of models representing a query (a question).

Rational agent

One that applies rational behavior using logic. Or one that maximizes its performances according to some performance measure and then should do the right thing (Russel & Norvig, 2010)

RDF (Resource Description Framework)

RDF is a knowledge representation language dedicated to the annotation of resources within the Semantic Web. Currently, many documents and contents are annotated via RDF due to its simple data model and its formal semantics. The W3C published a specification of RDF's data model and XML syntax as a Recommendation in 1999. Work then began on a new version that was published as a set of related specifications in 2004.

Reality that logic formalizes

" (…) the reality that formalizes the logic is no longer to be found in internal or external experiences considered statically, but in an interesting historical process of building all levels of development, from the most elementary to higher forms of reflective abstraction involved in the history of logic itself. While psychology analyze conditions due to the formation of these structures and their way of operating plan for the coordination of actions to that of reflective abstraction, logic can be formalized as a system of abstract operations and not as experienced or as used in the subject's experience, which eliminates any psychologism" (Piaget, 1967).

Role

An item or a formula of a predicate frame.

Rule

A relationship, a formula or a clause between a premise and a conclusion, or a precondition and a postcondition. In our work, a rule is modelled by a n-ary predicate in first order logic, second order logic and higher order logic (recursive).

Rule Model

A class of models representing a rule.

Scale

In this thesis, we defined **Scale** as a kind of valued concepts that is variable according to the context of the sentence. They are the list of linguistic variables of fuzzy logic membership function in which the limits of the function can change depending on context.

Scenario

A specification of a purposeful sequence of activities or actions produced by the agents involved and realized by services involved.

Semantic agent

Rational agent that contains a knowledge base and an inference engine to reason on knowledge. In our architectural design, an agent is a basic component and web service of the architecture.

Semantics

It gives a meaning, and does not define if a logical statement or propositional statement is true or not, the latter (truthful) is a consequence of the former (meaning at a different level of abstraction), most often at a lower level (the lowest level contains atoms and primitives).

Example: A number or a word is one or a list of characters, a character is a written or printed sign, a sign is a drawn symbol (in this context of number).

Service

A service is a component of the architecture to manage or control inputs or outputs. In our case they are web services. Input Service is a sensor service sending events to agents. Output Service is a actuator service receiveing events from agents.

Network services are software, web services or hardware controllers working in a network. Physical service are services which control hardware parts. Virtual service are services emulated by a software embedded in the EKRL concentrator which can receive or produce events like if the service were real.

Simple Object Access Protocol

SOAP est le protocole d'échange de messages permettant d'interagir avec les services Web. Ces messages sont délivrés sous la forme de documents XML qui sont structurés par : une entête (pouvant être vide) contenant les informations non fonctionnelles liées au message (par exemple la sécurité), ainsi qu'un corps contenant les informations fonctionnelles (données et opérations du domaine d'application).

SOAP est générique et extensible puisque les autres spécifications sont définies par dessus en définissant par exemple de nouveaux entêtes (sécurité, garantie de livraison). Ce protocole n'est pas lié à un type de protocole de transport de données particulier, mais il est fréquent qu'il soit associé à HTTP ou SMTP.

Situated Action

Intelligence should be understood in terms of interaction between an agent and environment, rather than in terms of the manipulations of an agent of a hostile world. For [Agre and Chapman, 1990], the challenge is to understand how routine behavior can arise and adapt to a changing environment, rather than how the system can anticipate and plan for every possible contingency. Given this interaction-oriented outlook, the separation of perception, planning, and execution no longer make sense. All parts of the agent should be integrated and tightly coupled with sensing and action. For Agre and Chapman, the parts of the agent are based on simple routines in which the agent should engage when placed in a particular environment. These routines are decomposed into actions, with the rationale for each action analyzed. The rationale for actions is then reduced to conditions in the environment that the agent can sense. An agent, then, consists of physical actions that are cued by sensed conditions. (Suchman, 1987)

Software Architecture

A *Software Architecture* is defined [. . .] as the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution (IEEE Std 1471-2000). The definition of the structures of the system that is implemented in software, composed of architectural elements and the relations between the elements.

Software Engineering Methodology

A kind of software engineering process for the organized production ofsoftware using a collection of predefined techniques and notational convention.

Software Engineering Process

A kind of process involving activities, roles, and resources that produces intended software of some kind.

SPARQL

SPARQL is the query language for querying RDF knowledge bases. SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. SPARQL also supports extensible value testing and constraining queries by source RDF graph. The results of SPARQL queries can be results sets or RDF graphs. In 2008, the W3C published the latest specification of SPARQL as a Recommendation.

Speech Acts (Illocutionary acts)

Searle (1975) set up the following classification of illocutionary speech acts:

- assertives = speech acts that commit a speaker to the truth of the expressed proposition
- directives = speech acts that are to cause the hearer to take a particular action, e.g. requests, commands and advice
- commissives = speech acts that commit a speaker to some future action, e.g. promises and oaths
- expressives = speech acts that express on the speaker's attitudes and emotions towards the proposition, e.g. congratulations, excuses and thanks
- declarations = speech acts that change the reality in accord with the proposition of the declaration, e.g. baptisms, pronouncing someone guilty or pronouncing someone husband and wife

System

A set of entities or components connected together to make a complex whole or perform a complex function.

System of Systems

A *system of systems* is a set or arrangement of systems that results when independent and useful systems are integrated into a large system that delivers unique capabilities. (Carnegie Mellon, Software Engineering Institute, CMMI glossary, October 2010).

Task

A kind of activity where the logic goal to be achieved by the activity has been defined explicitlybefore the activity is started and where the actions performed by an agent are defined in terms of plans.

UML

Unified Modelling Language is a standardized general-purpose modelling language in the field of software engineering.

Universal Description, Discovery and Integration

UDDI est une spécification et un service de registre permettant de publier et de découvrir des services Web. Un registre UDDI est un registre basé sur XML qui contient des informations à propos d'entités d'affaire fournissant des services Web ainsi que des métadonnées concernant ces services (informations techniques ou légales). En outre, UDDI spécifie plusieurs API pour interagir avec le registre, demander ou publier un service.

Utility

Utility is a way of accounting for how desirable a particular state of the agent in the environment or the environment is and can therefore be used as a performance measure like principle of maximum expected utility.

Web Service

A web service is a service processed by an agent on a TCP/IP communication network. In addition, web services can have a web interface to list available functions and also to use it. In our definition, a web service can be a robot, a human or any sensors or actuators in the environment.

Web Service Description Language

WSDL is the XML-based language for describing Web services. It can generate structured documents in two parts: an abstract part (the event) that describes the functional interface of the service in terms of operations and messages, as well as some practical (the how) that contains the details of protocols to use and the physical address of operations. In particular, a port type denotes a collection of operations, a connection is a combination between a typical port and a transport protocol and data format, a port defines the physical address of a connection, and a service is a collection of ports

INDEX