University of Porto
Faculty of Engineering
Department of Electrical and Computer Engineering

Thesis presented for the fulfilment of the requirements necessary to complete the degree of PhD in Electrical and Computer Engineering

AFRANCI: A Multi-Layer Architecture for Cognitive Agents

by

Francisco Antonio Fernandes Reinaldo

Supervisor: Professor Rui Camacho
Co-supervisor: Professor Luís Paulo Reis

To my parents Alcebíades Reinaldo and Emília Fernandes Reinaldo

Resumo

O desenvolvimento de agentes autónomos e inteligentes que são capazes de cumprir tarefas e sobreviver em ambientes pouco favoráveis, dinâmicos e imprevisíveis, é uma tarefa extremamente complexa. Inteligência, neste trabalho, é entendida como a efectiva negociação de comportamentos entre as camadas internas do agente, permitindo respostas correctas ao ambiente onde opera.

Esta tese propõe que a inspiração pela análise de arquitecturas baseadas em comportamentos e cognição e o resultado da sua junção, seja a fonte de mecanismos e soluções que permitam projectar e implementar novas arquiteturas de agentes com elevado grau de autonomia.

Assim, esta tese apresenta um modelo de arquitectura híbrida, que é composta por modelos simbólicos e não-simbólicos, para modelizar os aspectos comportamentais e inteligentes de um agente. Esta metodologia oferece uma arquitectura heterogénea com diferentes camadas de abstracção. Sua estrutura multi-camadas é interconectada. Cada camanda suporta diversos tipos de comportamento como os estereotipados, reactivos, instintivos, deliberativos e de consciência. A camada "consciência" está localizada no metanível cognitivo da arquitectura. Ela contém uma maquinaria para desenvolver o papel de monitorar os estados internos e sensores e as experiências adquiridas e então sugerir comportamentos inteligentes no momento adequado. O conflito interno de comportamentos que são gerados pelas camadas simples e complexas, comuns em arquitecturas de agentes, são resolvidos pela própria heterogeneidade intrínseca das camadas integradas e pela utilização de seus vários módulos de controle, prioritizando o trabalho em simultâneo.

A arquitectura proposta oferece uma estrutura hierárquica expansível e um fluxo de informação bidireccional íntegro que alimenta os diferentes níveis de inteligência para promover a correcta emergência de comportamentos pela tomada de decisões autónomas. A estrutura hierárquica é o resultado de uma série de classes de comportamentos e de características observadas em outras arquitecturas. Em cada camada ou entre camadas, heterogéneos módulos de controle colaboram entre si no processo de controle comportamental e cognitivo do agente,

deixando-o apto a agir em um dado ambiente, consoante suas crenças e intenções. Assim, procura-se comprovar a hipótese de que os diferentes níveis hierárquicos de inteligência possam fornecer interpretação de informação e novas acções que guiem o agente para diferentes espaços de soluções pelos diferentes estados internos.

Para implementar esta estrutura multi-camada, no desenvolvimento de agentes autónomos, construiu-se AFRANCI. AFRANCI é uma ferramenta que modeliza e automatiza *frameworks*. Construída nos fundamentos da PiramidNet, arquitectura biologicamente inspirada, AFRANCI herda algoritmos da abordagem conexionista (não-simbólica) e acrescenta outros algoritmos da abordagem simbólica. A ferramenta AFRANCI permite que utilizadores comuns integrem os diferentes algoritmos de Aprendizagem Computacional (Machine Learning) e projectem um Agente Cognitivo robusto. O sistema produzido pelo utilizador através de AFRANCI é expansível e possibilita que outros algoritmos sejam adicionados sem a necessidade de recompilar toda a estrutura já desenvolvida. A automatização da construção dos agentes utiliza apenas três passos: a especificação da estrutura, o treino dos módulos e a geração do código fonte final. Melhoras significativas foram alcançadas com o desenvolvimento da estrutura de alguns algoritmos de Aprendizagem Computacional através do uso de *wrappers*, contribuindo para o processo de investigação e agilizando a tomada de decisão dos agentes de forma correcta.

Abstract

The development of Autonomous and Intelligent agents, capable of accomplishing goals and survive in complex, dynamic and unpredictable environments is a highly complex task. In this context, intelligence, is regarded as an adaptive and a fast behaviour that agents use to survive in the environments.

In this thesis we propose that the analysis and combination of computational models involving both behaviour and cognition will lead to improved agents with high degree of autonomy and utility. Following that proposal we developed a hybrid methodology (symbolic and non-symbolic) to model and standardise the agent's behavioural and intelligent aspects. This methodology enables the development of an architecture with different levels of abstraction. The multi-layer structure is interconnected. Each layer implements several types of behaviours that include stereotyped, reactive, instinctive, deliberative and conscient behaviours. The "conscient" layer it is located at the cognitive meta-level of the architecture. This layer is powerful enough to be able to monitor the inner states and sensors and acquired past experiences and then suggest intelligent behaviours at the adequate circumstances. Internal conflict behaviours generated in the lower layers, so common in agents architectures, are resolved by control modules working simultaneously.

The abstract architecture is implemented in an hierarchic and expandable structure, and a bidirectional information flow integrates and feeds the different intelligence levels in order to promote the right behaviour emerging by autonomous decision making. The proposed hierarchic structure is the result of a series of behaviour classes and characteristics observed in other architectures. In each layer or between them, heterogeneous control modules implement Machine Learning algorithms that interchange messages to cooperate and control the agent behaviour and conscience, making it capable to act in an environment in accordance with its believes, desires and intentions. Thus, the thesis corroborates the hypothesis that different hierarchic intelligent levels offer the interpretation of

information and trigger new behavioural actions that guide the agent to different space of solutions by different inner states.

To be able to easily implement the proposed multi-layer agent structure we developed the AFRANCI tool. AFRANCI is a tool capable of modelling and automate frameworks. AFRANCI is based on a previously proposed approach, PiramidNet, uses connectionist algorithms and integrates them with the symbolic algorithms. This tool allows any common user to integrate different Machine Learning algorithms and develop a robust Cognitive Agent. The system produced by the user is extensible and allows the addition of other algorithms without recompiling the whole of the structure already developed. The automation of the agent construction process is based on three steps: the specification of the structure; the modules training and finally; the code generation. Significant improvements were found with the development of wrappers that automatically tune the Machine Learning algorithms.

Résumé

La mise en place d'agents autonomes et intelligents capables d'accomplir des tâches et de survivre au sein d' environnements plutôt hostiles, dynamiques et imprévisibles, est extrêmement complexe. Dans ce cas spécifique, l'intelligence est envisagée comme négociation effective du comportement entre les couches internes de l'agent, permettant ainsi d'avoir des réponses correctes du milieu dans lequel elle opère. La proposition de cette thèse est de soutenir que l'inspiration émanant de l'analyse d'architectures elles-mêmes fondées sur les comportements et la cognition bien comme le résultat de leur jonction, soit source de mécanismes et solutions capable de projeter et mettre en oeuvre de nouvelles architectures d'agents à haut degré d'autonomie. Cette thèse présente ainsi un modèle d'architecture hybride composée de modèles symboliques et non symboliques, ayant pour but de moderniser les aspects comportamentaux et intelligents de l'agent. Cette méthodologie offre une architecture hétérogène munie de différentes couches d'abstraction. Sa structure multi-couches est interconnectée. Chaque couche donne support a des comportements, quíls soient stéréotypés, réactifs, instinctifs, délibératifs ou de conscience. La couche "conscience" est située au méta-niveau cognitif de l'architecture. Dotée de toute une machinerie qui contrôle et les états internes et les capteurs et les expériences acquises, elle suggère au moment opportun, des comportements intelligents. Le conflit interne des comportements produits par les couches simples et complexes et qui sont communs dans les architectures d'agents, est réglé par la propre hétérogénéite intrinsèque des couches intégrées ainsi que par l'utilisation de la plurarilité des modules de contrôle. Le travail (en) simultané est mis en exergue . L'architecture proposée offre une structure hiérarchique expansible et un flux d'information bidirectionnel complet qui alimente les différents niveaux d'intelligence. Par ce biais la prise de décisions autonomes déclenchera l'urgence correcte de comportements. La structure hiérarchique , est le résultat d'une série de classes de comportements et de caractéristiques déjà observées à l'intérieur d'autres architectures. Dans chaque couche ou entre couches, des modules de contrôle hétérogènes collaborent entre eux au processus de contrôle comportemental et cognitif de l'agent

alors apte à agir dans un environnement donné, au gré de ses croyances et intentions. Tentative qui prouverait l'hypothèse que les différents niveaux hiérarchiques d'intelligence pourraient fournir une interprétation de l'information ainsi que de nouvelles actions guidant l'agent vers différents espaces de solutions au moyen de différents états internes. AFRANCI a vu le jour afin d' implanter cette structure multi-couche dans le développement d'agents autonomes. AFRANCI est un outil qui modèlise et automatise des frameworks. Conçu sur la base du piramidnet, AFRANCI hérite des algorithmes de l'approche connexioniste (non symbolique) en addition à d' autres algorithmes de l'approche symbolique. AFRANCI permet aux utilisateurs communs d'intégrer les différents algorithmes d'apprentissage de Machine et de projeter un système robuste d'apprentissage multistratégique de l'ordinateur. Avec AFRANCI, l'architecture produite par l'usager peut être expansible et permet l'ajout d'autres algorithmes sans avoir a recompiler toute la structure déjà existante. Seulement trois etapes sont nécessaires a l'automatisation de la construction des agents: la spécificité de la structure, l'entraînement des modules et l'engendrement du code source final. Dû a l'utilisation de wrappers, des gains significatifs dans le développement de la structure de quelques algorithmes d'apprentissage de Machine ont été atteints, ce qui a contribué au processus d'investigation et a pu accélérer de façon correcte la prise de décision des agents.

Contents

List of Figures

List of Tables

Acknowledgements

A great deal of effort goes into writing a thesis. I first of all wish to thank God, who makes this opportunity real, unique and possible. After that, my primary debt, of course, is to Prof. Eugénio Oliveira at LIACC for his support during the time I was "living" in his research centre at FEUP, and mainly to Prof. Luís Paulo Reis and Prof. Rui Camacho that opened their arms for receiving a foreign student came from South America.

I owe so much to many people. I want to express my gratitude to Prof. Luís Paulo Reis and Prof. Rui Camacho for their support and advising during the time I was writing the thesis. Both advisors and their numerous comments, suggestions, corrections and hints have substantially improved the quality of the text. I am greatly in debit to Rui Camacho for his careful reading of all chapters of the manuscript; through his constant encouragement also kept me on the right track. I am grateful to Andreia Malucelli, Ana Paula Gonçalves and Ricardo Moraes for advice, encouragement and comments. There is insufficient space here to list all those who have made significant contributions; I thank them all. The impetuous of this thesis came from the relationship among you all. Among them I specially wish to acknowledge Leornardo and his wife (both in memoriam) for their patience and advice. Thanks to be a great guy! I want to express my gratitude to Fatinha, Ildo and wife, José and wife for their affectuous and good humour.

I would like to thank my families and friends for the support given during the preparation of this thesis, specially for my lovely mom and daddy for all support over these years. I would also like to take this opportunity to thank Adilson de Sousa, Antonio Machado Filho, Ana Mariante, Ângela Reis, Demétrio Renó, Eva Machnikova, Francisco Vasques, Frederic Hustinx, Jamilson, Maria Tereza, Nuno Lau, Norton Oliveira, Kátia Ramos, Ricardo Oliveira, Tatiane Rocha, and many other friends for their friendship and patience. Hearty thanks for similar reasons go to Cartoninha1 and family, Brasup, Bezerra of Menezes's Centre, and SASUP.

This acknowledge is for everyone direct or indirectly involved in this thesis. My apologies if I missed someone. Thank you all.

Last, but not least, I want to express my gratitude to the FCT and Unileste-MG for supporting my work and the LIACC and LIC for providing a most pleasant environment in which to complete my work on this thesis. I gratefully recognise all of my colleagues at LIAC and LIC who helped me anytime I need.

Chapter 1

Introduction

One of the main hallmarks of Artificial Intelligence (AI) is the heterogeneous abilities of an agent to act independently in the environment, such as to simulate perceptual and motor processes, and cognitive faculties. According to this idea, AI emerged from the experimental approach of a sequence of behaviours - scripted behaviours [214]. Whether the scripted behaviour produces a known answer to the situation, there is no learning by the agent in that moment, but an automatic thinking.

The first scientific stretches of thinking machines theory begun in 1940 with a progressive movement labelled Cybernetics [246]. The group of cyberneticists believed that the whole psychologic activity could be translated into mathematic models. The mathematic models would be used to simulate the brain functions by electric circuits in computational entities.

After cybernetic, AI notion emerged like a symbolic computational model, and mind was defined as a chain of mental representations made of symbols. The symbolic model uses symbols in an abstract form to represent knowledge at a deliberative level. Since then, computational entities can simulate natural processes of human intelligence. Knowledge is commonly acquired by relations between sensations of world and inner rules. Consequently, the act of knowing, refers to the emergent development of knowledge by information processing of "mental" functions such as inference, decision-making, planning and learning - common properties of an abstract mind on symbolic and sub-symbolic layers of a modular and hierarchic architecture.

In the symbolic model, the inner construction of world is based on representations of situations lived or repetitions of analogue situations. The representations are the result of acquired symbols by the sensory systems in contact with the

world. The construction of every representation is determined by cognitive knowledge. Cognitive knowledge evidences new ways to act based on inner interference or external noises.

Conversely to the symbolic model, the sub-symbolic model, also called connectionist or "the new robotic approach", does not use knowledge like representations. In this new approach, the notion of mind is not refereed as result of algorithms process, such as logic inferences. The main idea is not to set specific symbols, but deal with a global system state where complex interactions patterns emerge from it, without having a specific component storing it.

On the one hand, the classic approach is centred in the deliberative level to make decisions, and plans in a specific top-down strategy. On the other hand, the new robotic approach is looking for simple behaviours that do not require previous knowledge representations to achieve complex outcomes, like reacting in a typical bottom-up strategy.

To outline a reliable agent mind that produces fast responses, we believe to combine heterogeneous Machine Learning algorithms of two different cognitive lines in a single architecture. Thus, a hybrid layered architecture is modelled from combination of the best characteristics of some heterogeneous classic architectures.

1.1 Problem Description

The human beings are constantly in contact with the environment that surrounds them. They have skills to perceive the world, interpret symbols by analogies, react when necessary, learn the meaning of things by inclusion or exclusion principles, and interact back with the world. These innate human beings characteristics, encoded in the genes and biologically inherited from parents, have augmented the innate knowledge, and preserved the species up to now.

Looking at the Nature, we can see a diversity of other species that developed different innate abilities (skills) of survival and intelligence along the centuries. Typically, the skills are responsible to recognise information, use and manipulate them, judge them as adequate in face of a situation, formulate them again or tune their "thought" appropriately.

Species that use embedding knowledge to interact with the environment have several advantages. First, the access to the accurate and restructured knowledge, and further retrieval happens instantaneously (reflexivity, reactivity). Second, no current learning is necessary; consequently only interpretation of signals in real-time is need. Embedded knowledge means no memorisation of the world, which avoids storing an outdated or misleading knowledge of the environment in

a long-term memory. Third, the agent behaviour reflects in real-time the state of the world. Fourth, it requires neither reasoning nor training. This bottom-up approach is possible only in inferior levels. As Brooks observed, "The world is its own best model." [31]. On the other hand, species that relying on knowledge in the world have benefits in many ways. First, they use their cognitive process to monitor complex and dynamic conditions and affect the environment in a goal-directed way. Second, the cognitive process allows the opportunity of an organism to demonstrate emergent behaviour in face of some situation by applying knowledge (proactive decisions) for a specific purpose (for instance, develop a plan) if appropriated. This top-down approach is only possible in superior levels.

The possibility of simulating an intelligent agent mind made of patterns rather than particles - the common structure of an intelligent system (like a system of functions and controls) - was the main point that motivated us to develop this thesis. On this basis, we drive our research to combine autonomous decisions and pre-defined actions (skills) in an unique hybrid and robust agent architecture. Therefore, the agent can adapt its behaviour appropriately by external and internal stimuli, in this sense, our agent is said to behave "intelligently".

1.2 Symbolic and non-Symbolic Features

In symbolic (that is, rule-based) approach, symbols are used in an abstract level to extract implicit information from simple situations, represent models of environments, and trigger a chain of production rules for reaching a goal. If the goal was not reached, a new strategy will be obtained again by developing plans - several cycles of inner combinations of actions - until find the most promising answer. A symbolic goal-oriented behaviour uses long-term changes to reason about what actions to do next. The Symbolic systems store long-term changes in the form of production rules, commonly referred as universal subgoaling that are arranged in a single central unit, to simulate the human cognition. Thus, it is presumable that the agent behaviour system must be motivated by goal-oriented states and, consequently, learning occurs in the process.

Unfortunately, the symbolic approach fails in some aspects, such as: a) the classic symbolic approach has difficulties in dealing with noise and failure; b) it requires either representing some goals implicitly or forcing unrelated goals into a single hierarchy; c) it uses world representation and knowledge previously obtained to trigger an effective action; d) the representation of an object can scale-up as the size of the knowledge base increases; e) the development of plans represent a combinatory explosion of paths to be followed, and in the same time the problem grows in complexity; f) it is not useful the idea of frozen a dynamic environment in memory in order to find the best answer. Once the logic mechanisms do not

prevent invariant changes, so the system works with outdated values; g) it is very difficult to implement symbolic rationalisation (cognition) that allows agents to perform complex analysis of sensorial data quickly and generalised; h) update of long-term memories; i) the use of a single goal hierarchy affects the symbolic learning because produces over generalisation of chunks, and makes them expensive chunks; j) some catastrophic collapses can occur in long simulations, where a large set of good rules are lost; k) symbolic models are limited to maintain long simulations updated; l) these lost rules are acquired again, but the price of a huge instability of the hierarchy defaults; m) training is a learning way that occurs via division of production rules; n) the size of long-term memory in Symbolic system needs to be extended; o) it depends entirely on the world state to obtain the next "decision", and sometimes the current world state is not sufficient to provide the necessary "decision" about what to do next.

In this sense, it is perceived that a non-Symbolic approach, if rightly combined with a Symbolic approach, like simpler pieces of entities working together in different hierarchic levels, they can obtain a robust decision system. Thus, the following features of a non-Symbolic system comes to complete with or substitute some Symbolic systems characteristics. For instance, in non-Symbolic, the minimisation of the scalability problem happens by the use of biologic plausibility; conversely, automatic learning and imprecise general answers are a real trouble when logic is required, but a Symbolic system is able to solve it. Other non-Symbolic features are: a) there is no shared global memory; b) it is sufficiently robust in the presence of noise; c) its knowledge is stored in the form synaptic weights; d) the absence of an inner classic symbolic environment model generates low level behaviours, such as the reactive one; e) it deals with short-term changes; f) it acts freely in accordance with stimulus-response from changes in the environment; g) it reaches rationality from the result of interaction between reactivity and environment; h) it is robust in unpredictable environmental situations; i) there is no symbolic rationalisation (cognition) that allows agents to perform complex analysis of sensorial data; j) the process occurs in parallel, and actions can be performed without having to wait for such symbolic complexity; k) the association between raw data and action is pre-formed in the system, such as intrinsic rules; l) reasoning is usually represented as the adjustment of weights on the network's nodes; such models of reasoning are sometimes described as non-symbolic; m) training is a learning way that occurs via simple adjusts of weights (plastic knowledge).

Therefore, this work uses both the symbolic and the connectionist paradigms to develop AFRANCI architecture. The idea of joining both research areas appeared in McCulloch's work, which had a strong biologic inspiration and was made in conformity with Pitts's work using mathematical concepts. In the same direction, in order to construct hybrid systems, we combine the characteristics

of adaptability, robustness and uniformity that are offered by neural networks with representation, inference and universality, native characteristics of symbolic Artificial Intelligence. Thus, we analyse (a) several ML algorithms that makes relation with the proposed skills, (b) different architectural patterns in order to know what are the most robust and flexible at the same time, and (c) classic agent architectures with the purpose of collecting the most usable skills to our architecture.

1.3 Motivations

Our motivation is devoted on the research and development of behaviours in agents such as they origins, how the emerge and how they inner can command agent parts in order to be autonomous in unknown and unpredictable environments. Additionally, our motivation is to offer flexibility in the development of agents "soul". As the general architecture is concerned, we may customise a Cognitive Architecture and train it to solve a particular problem, such as the RoboCup Rescue Domain, CyberRat Domain or standalone experiments.

Other main motivations of this work are as follow:

- Development of an agent architecture that will be used as pattern for designing agents;
- Contribution in the improvement of straightforward agent learning techniques that are based on similarity of behavioural human being architecture;
- Providing opportunities for students to perform significant test bed in classroom through which they can learn many of the train techniques of multiagent system development;
- Comprising the following domains: social, technical and cognitive. Social environment is significant because it focuses on human search and rescue mission planning. Technical research is important because it solves problems of strategies and tactics for a fast and optimised mission. Cognitive science is considered because it studies models of information and representation, capacity of human memory and biologic behaviour to be applied in an agent.

1.4 Research Objectives

The main goal of this thesis is to develop and architecture for autonomous agent with cognitive insights, and to research implementation issues associated with its development. Thus, this thesis has the following objectives:

- Investigate and identify various relevant agent issues through the theoretical studies about architecture and meta-architecture styles, and symbolic and connectionist approaches. These include investigate how many capabilities should the agent have, understand what control processes are capable of such activity, comprehend whether the arrangement of control modules and data flow interferes on manifestation of intelligence, identify which cognitive strategies would be used to produce reasoning in agent, and discuss motivations for combining the most promising features of each style by using heterogeneous Machine Learning algorithms of both heterogeneous approaches in a robust framework;

- Develop an agent architecture that hosts heterogeneous architectures of Symbolic and non-Symbolic approaches in different organisational levels. The proposed architecture must synchronise the communication among levels for working together in order to reach the agent goals autonomously. We present a sample framework that directly supports reasoning, methods of prevision and abstraction obtained from many well-known architectures and representation from sensors to actuators;

- Develop AFRANCI Tool, a tool designed for easy agent architecture implementation. The tool uses Machine Learning algorithms for automatisation of tasks on the development of behaviour - cognition modules composed by symbolic-connectionist approaches in the framework;

- Implement a computational agent with bidirectional route of data with lower layer sending sensory inputs to the upper layer in order to solve a problem by specialised architecture slices; consequently, the upper layer sending data back to the lower layer to perform actions by its actuators;

- Evaluate the performance of agent architecture, cross-comparing the results along the simulations.

1.5 Outline of the Thesis

The individual chapters are briefly described in the next paragraphs and further elucidate the specific goals of the research.

Chapter 1 outlines the thesis and presents a general introduction to the problem area by establishing the research problem, the research aims, and contributions.

Chapter 2 presents the state-of-the-art and the popular foundations of the term learning, followed by the analysis of the most usable Machine Learning (ML) algorithms, such as Decision Trees, Rule Induction, Artificial Neural Networks, and Genetic Algorithms. This chapter molds the initial ML specification for our agent architecture. We also take the first steps towards elucidating the reflective, reactive, deliberative and cognitive activities to the control modules by describing their functional attributes, and proposing a fourth-layered model that will explore the structural and dimensional attributes of the mechanisms.

Chapter 3 shows the state-of-the-art of typical architectural approaches used to assemble computational entities. We start with a brief overview of related work on such architectural approaches [83, 219, 33, 17], and identify the most useful architecture features to build our agent architecture. Additionally, the chapter provides a brief overview of existing deliberative and behaviour-based agent architectures [102, 28, 204, 155] followed by their implementations. We also clarify a set of terms and concepts to lead the reader capable of understanding of skills that can be used towards build our intelligent agent entity. We then highlight the most interesting skills to meet the basic requirements of intelligent autonomy for agents.

Chapter 4 describes the AFRANCI tool as an intuitive and visual resource to develop Multi-Strategy Learning systems for autonomous agents. AFRANCI tool provides a set of heterogeneous ML induced modules and resources to train, and test them.

Chapter 5 presents the design of the cognitively-inspired agent architecture connecting hybrid states - integrating the different research strands explored from chapters 1 to 4. We also describe how the different concern-processing competence levels of our four-layered architecture can act, and we identify the different processes active in the emergence of decision states.

Chapter 6 reports the implementation of our agent design, and an analysis of similar designs in the design-space. We also present an analysis and evaluation of our design, and address some of the architectural requirements needing it.

Chapter 7 summarises the contributions of this research to the field of understanding concern-processing in intelligent and autonomous agents, and points to new directions in which the research can be taken in the future.

Chapter 2

Machine Learning algorithms

Machine Learning algorithms comprises a set of techniques for acquisition and integration of new knowledge by study or training with intention to achieve a particular goal. This capacity of learning from observation focuses on continuous self-improvement. Using Machine Learning, models can predict how a system behaves or "think" under certain circumstances, such as to survive and respond in the world. In this chapter, we focus on the techniques that are directly relevant for the thesis work. The Machine Learning algorithms reviewed in this chapter are Artificial Neural Networks, Rule Induction, Decision Trees, and Genetic Algorithms. These algorithms will be used to fill in the modules at each level of the agent architecture as described in the following chapters.

2.1 Introduction

Machine Learning algorithms (ML) are computational tools capable of optimising the performance criterion of a model using example data or past experience, and a desired output or action [5].

Inductive learners are left to discover - or induce - rules from their experience, that is, a general rule is derived from a specific case and then applied in all cases. So, Inductive learning takes examples and generalises rather than starting with existing knowledge. Actually, there is a contrast between Deductive and Inductive learning methods. While induction follows from particular to general, deduction follows from general to specific instances. However, there is a scope of error in the inductive method, but supervised learning techniques address that problem as we describe next.

Supervised Learning (SL) is a subset of Machine Learning techniques that monitors or maps the off-line mode of both inputs and outputs. In SL, an instructor "supervises" the training and test cycles of the algorithm. The feedback can be closer or not from the expected result. Commonly, SL deals with classification and regression problems. Learning by classification uses discrete values that respond to instance cases of the model, that is, assign examples to predefined classes. On the other hand, learning by regression the goal is to predict a numerical value.

The SL algorithms we choose to construct our agent architecture are Artificial Neural Networks, Rule Induction, Decisions Trees, and Genetic Algorithm. These SL algorithms offer the most important features for an agent sense the environment and deliberate actions to the environment. The features of ML algorithms are: a) estimate fundamental rules successfully; b) maximise correct data from classification and regression; c) derive correct choices from observation data; d) detect patterns in a data set, and produce decision rules made easy for humans beings - except for Artificial Neural Networks; e) find a solution by recursive division of problem in subproblems; f) take advantage of incremental training already completed by encoding all past training examples as negative examples for a hypothetical learning task.

2.2 Artificial Neural Networks

In the recent past, the metaphor of mind, prominent since the 1960's, has been reset by the *brain metaphor* [121] in which learning and cognition take place via simplified models of the connectionist approach (interconnected network of neurones). So, we define mind as a set of abstract computations. The syntactic proprieties of symbols are controllable and a programmer establishes the semantic. In the same line, cognition is represented by an act of knowing or knowledge, which models describe or explain certain behaviours emerged in terms of information flow or functions of brain. Cognition also represents a process or method fired by an inner state or situation of the environment, so understood by perceptive brain mechanisms. In this sense, metacognition is an abstract cognitive method that creates cognition, and information-processing systems (computational entities) can simulate cognition.

The connectionist approach attempts to understand how interconnected network of neurone-like-units work, can learn and remember facts. The approach stresses the capability of learning, recognising patterns and discovering representations. Representation may be seen as the development of a similarity model of the world, based on background knowledge and trails made by memory to find analogue states.

In this fashion, the *brain metaphor* suggests that intelligent systems can be built by adding and connecting in parallel a large number of simple processing units (neurones). Thus, the brain-style parallel computation announces new directions to develop a network composed of collective simple neurones working together. This kind of network is termed Artificial Neural Networks (ANN).

An ANN is a mathematic model of information processing that attempts to mimic functions of the brain. Therefore, complex behaviours and robustness of ANN emerge from collective workings neurones. Neurones are individual processes that perform only simple operations. Its individual autonomy gives ANN the ability to achieve a better performance. Although there is no explicit knowledge on how the brain works, it is well known that learning from examples, decision-making, and recognising patterns are considerable powerful characteristics of this flexible system.

ANN history started when the first neurone was encoded like a binary circuit by McCulloch and Pitts in 1943 [148]. In the 1943, they observed the potential waves of a neurone membrane among many neurones. It was believed that a rather complicated computer program could be encoded like a brain structure and produce the same brain outputs. The McCulloch-Pitts's artificial neurone model fires an impulse only if its threshold value was exceeded. Because of threshold has a predetermined limit in that epoch, the neurone model was not able to learn.

Rosenblatt published the basis of his work on the perceptron theory in 1958 [207]. In his work, Rosenblatt solved the limitations of McCulloch and Pitts's model. Rosenblatt developed the first neurone able to learn [22]. The Rosenblatt's perceptron is in Figure 2.1. As opposed to McCulloch and Pitts's model, the general neurone model has null retard. Additionally, transfer function output was adjusted to support not only binary responses, but also to assume continuous values.

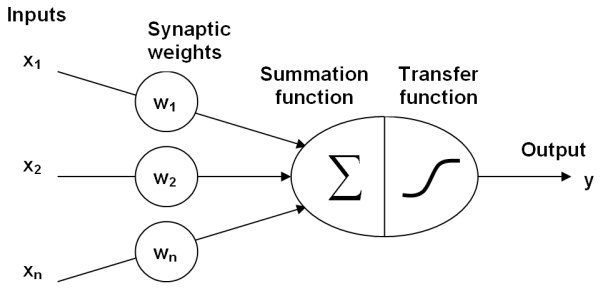

Figure 2.1: Functional representation of an artificial neurone.

In order to produce a desired output, the perceptron, also called neurone, receives a predefined range of values from the environment or from the output of other neurones. The values are combined by a function (x) to produce an effec-

tive activation value. After that, a transfer function receives the activate value to produce the desired neurone output (or signal). The most common transfer functions are: Identify function, Binary Step, Binary Sigmoid, and Bipolar Sigmoid [65], respectively the Figures 2.2, 2.3, 2.4 and 2.5.

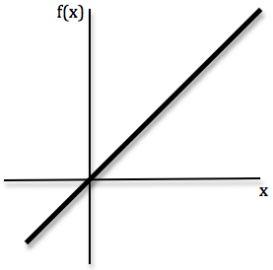

Figure 2.2: Identify function.

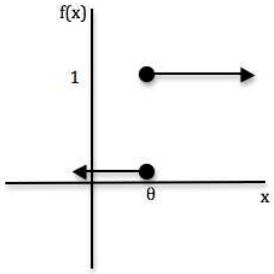

Figure 2.3: Binary Step function.

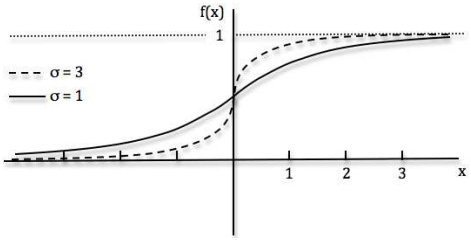

Figure 2.4: Binary Sigmoid function.

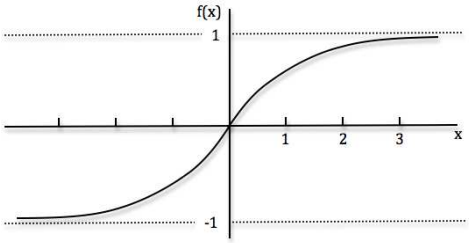

Figure 2.5: Bipolar Sigmoid function.

Networks that have only one adaptive layer are bounded to recognise only linearly separable patterns [157]. This limitation was overcome with the introduction of a intermediary or hidden layer. So, a network that has one or more hidden layers called Multilayered Layer Perceptrons (MLP) or Multilayered Feedforward Artificial Neural Network (FF), and its learning algorithm is called Backward error Propagation (or Backpropagation) [244, 208].

A MLP permits more complex, nonlinear relationships of input data with output results. MLP can learn continuous mapping with an arbitrary accuracy. The MLP network is assembled by perceptrons interconnecting other perceptrons by unidirectional channels (axons), and they are structured in at least three layers as exemplified in Figure 2.6.

Figure 2.6 presents ANN diagram. The circles (n_1 to n_6) are neurones arranged in a network. Neurones are interconnected by axons-dendrites (synapses). Neurones that receive stimuli from the environment are titled input neurones, and they are set at the input layer (n_1 and n_2). Neurones acting in the environment are called output neurones, and they constitute the output layer (neuron n_6). The neurones between the input and the output layer are considered to be

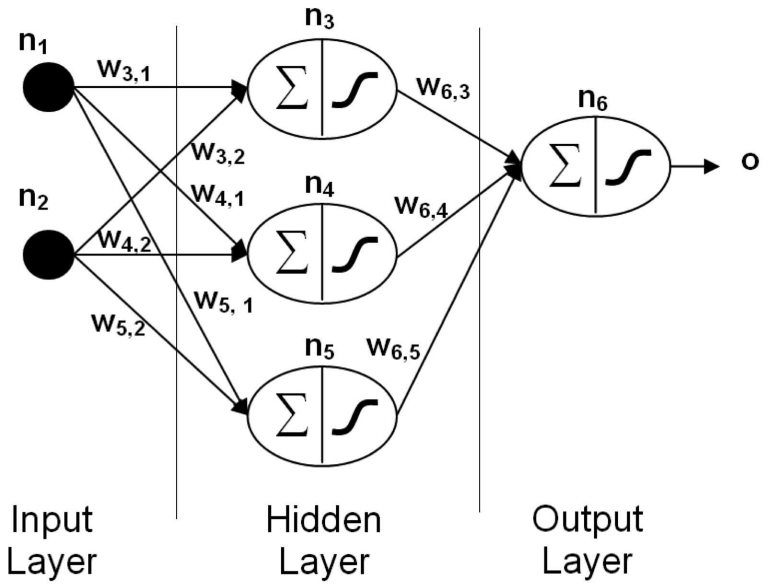

Figure 2.6: A model of MLP architecture.

"hidden" neurones, and so they are located at one or more intermediate layers or "hidden" layers (neurones n_3, n_4 and n_5). Hidden layers allow recognition of non-linear associations between input and output patterns (vectors) because they can form more complex decision regions (rather than just hyperplanes). A useful example of non-linear associations were described in [81] and are presented in Figure 2.7.

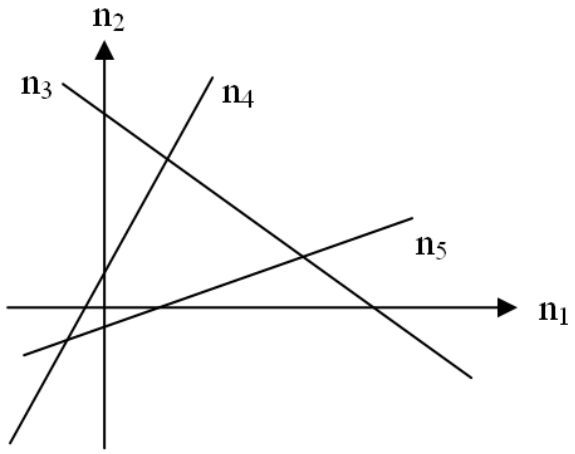

Figure 2.7: Arbitrary decision regions modelled by FF.

In ANN diagram above described, each neurone processes simultaneous inputs neurones of the previous layer (neighbouring neurones (short-term mem-

ory)). Each simultaneous input has its own signal (synaptic weight) received from one or many arrows (axons or connections). Signals can be adjusted by weights. The weights distributed on the connections are pieces of information stored in synapses that have been acquired to solve a problem. More specifically, the weights represent the knowledge (long-term memory) that enable ANN to store and recall patterns, classify patterns, perform general mapping from input patterns to output patterns, or find solutions to constrained optimisation problems [65]. In the first layer, each node creates a hyperplane. In the second layer, each node combines hyperplanes to create convex decision regions. Finally, in the third layer, each node combines convex regions to form concave regions [81]. By storing the information in weights, ANN can convert data patterns into behaviours. It is well noting that synaptic weights determine the behaviour of the network.

In Figure 2.6, arrows give a forward direction of activation, from the input to the output layer. The weights labelling the arrows are the outcome (or synaptic strengths) between neurones and previous weights. Synaptic strengths can excite or inhibit neurones situated in the next layer. The excitatory or inhibitory connection is indicated by the plus and minus signs, respectively. Synaptic weights can be interpreted like a matrix of real numbers or integer values in a graph. It is worth noting that, except the input neurones, every neurone uses a transfer function to get an activation value.

As the brain learns from experience, and ANN also uses that method. Learning is implemented by adjusting the synaptic weights between layers in order to optimise the network performance when a pattern is presented. The learning phase of ANN uses a set of vector pairs to learn from experience. Vector pairs are schemers of assimilation formed by background knowledge to embody a behaviour script in training cycle. During each network training cycle, synaptic connection weights are adjustable to be closer to the pattern presented; consequently minimise the difference between the desired and actual network outputs, and to optimise the network performance when a pattern is presented. The ANN synaptic connection weights represent a set of well-defined "rules" acquired along training cycle or learning phase. For each example showed to the input of the network, the correspondent correct output value will also be presented. Thus, a network learns when we say the network has plasticity.

As mentioned before, Backward error Propagation paradigm, or simply Backpropagation learning algorithm (BP) is a well-known SL method for training cycle of FF. As BP the name suggests, the error generated by output layer will be feedback to the hidden layers and to the input layer in order to update the weights. Commonly, the weight correction is applied to entire axons of the network to reduce the error rate. Activation levels are necessary to determine the values used as the basis for weight adjustments.

The basis of BP is the Delta Rule [245] function in its generalised form. Delta Rule adjusts proportionally the weights to the output in order to minimise the Mean-Squared Error (MSE). The more vectors from the training set are applied to the network or the more iterative cycles are fired, the faster the error rate is reduced. In each training cycle, the error signal modifies the network weights in direction to the minimal error. The process of training cycle is then repeated until the MSE of the output reaches an acceptable value.

Conclusions

Artificial Neural Networks are a subject of common interest to Psychologists, Neurophysiologists, Scientists, Engineers and other researchers. ANN respond by stimuli and ANN do not require a formalised algorithm to achieve a goal. A sub-optimal result is commonly necessary to solve problems instead of "optimal" results because a final result is unclear in some cases.

The connectionist learning technique adopts the brain-style information processing. Depending on the ANN topology, strings of facts can be processed to support noisy or imprecise data. Additionally, connectionist models deal with "unseen" patterns and generalise them from the training set.

However, ANN have inspiration on biologic networks of neurones, there is minimal similarity between a biologic and an artificial neurone, and the network topology. Nevertheless, the minimal similarity is sufficiently robust in the presence of noise and failure. So, noise and failure are just the tasks that the classic symbolic approach has difficulty in dealing with.

2.3 Rule Induction Algorithms

A Rule Induction algorithm is a symbolic Machine Learning method used to induce rules from the example cases. A rule is a kind of implication with an antecedent part and a consequent part, and an example case is a state that represents a description of a problem situation in a given moment.

Rule induction transforms the process of constructing a new rule into a search over the space. In such space, a goal state is any acceptable rule. Using induction to solve a problem, a state corresponds to a candidate rule and operators correspond to generalisation and specialisation operations that transform a candidate rule into another. So, the rule is fashioned like a basic generalisation or specialisation operation (set of sequential beliefs or conditions) accepted as true. It is also described as well-formed formula, rules are commonly used to determine predefined categories and to construct the basis of reasoning or to dictate the behaviour actions. Inductive reasoning, by its nature, is more open-ended and exploratory,

especially at the beginning. In its natural form, an inductive reasoning process from specific observations and measures, starts to detect patterns and regularities, formulates some hypotheses that can be explored, and finally end up developing some general conclusions or theories that defines the production conditions.

CN2 [41, 38] is a typical rule-based induction algorithm that accepts examples as input, together with relevant information (the background knowledge), and induces a model that "explains" the examples given in the background knowledge. The inductive reasoning process creates useful broader concepts obtained by bottom-up inference from specific observations from the whole environment, based on a necessarily limited number of observations; informally, it is called a "bottom up" approach.

The CN2 Induction algorithm was developed at the Turing Institute as part of the Machine Learning Toolbox project. The algorithm was designed to be an efficient induction tool of simple decision rules for problems involving large data sets where there might be noise[1] [39, 38]. The latest version of CN2 offers statistic methods similar to tree pruning in `IF...THEN...` rules generated from a set of samples [41]. The CN2 algorithm had previously been developed for UNIX systems, but new improve it to run on Microsoft Windows system [197].

Basically, the CN2 algorithm works as follows:

Observations Two data files are loaded to construct and test decision rules along with statements of the data types of each attribute and the examples of the algorithm;

Discovery of a relation between them The algorithm uses a concept description language, the rule is assembled in the form of:
`IF <complex> THEN <class>`.
A `<complex>` is specialised by either adding a new conjunctive term or removing a disjunctive element in one of its selectors. The learning algorithm works interactively, and for each new iteration the algorithm searches for a `<complex>` that predicts a large number of samples in a unique `<class>`, and few in other classes;

Generalisation The system searches for the `<complex>` by performing a pruned global-to-local search. When the `<complex>` is evaluated as good, the samples predicted are removed from the training set, and the rule `IF <complex> THEN predict <class>` is added to the end of the rule list. The last rule, in the CN2 list, is a "default rule" that classifies all the new samples based on the most frequent `<class>`. This process repeats until the satisfactory `<complex>` no longer exist or there are no more examples to "explain".

[1] The noise represents errors due to transcription or due to an insufficient description language.

The choice to apply the operator is no longer restricted on seed positive and negative examples of a specific class; rather is determined by an user-defined heuristic function. The user-defined heuristic function evaluates the effectiveness of each `<complex>` with respect to the given candidate rule on its classification performance on the whole training set. The obtained rules can cover overlapping data regions, that is, an instance can satisfy the antecedents of several rules. This new feature makes the algorithm more tolerant to noise, rather just predictive and reliable [41].

The learning process of the first version of CN2 generated a set of rules in an ordered fashion, and Entropy function was used to evaluate the quality of a `<complex>` [41]. The lower the entropy, the better the `<complex>` is. For instance, the quality of the `<complex>` is evaluated by determining if a new `<complex>` should be reset by the most improved `<complex>` found, and which `<complex>` should be discarded if the maximum size is exceeded. The Entropy evaluation function is given by:

$$Entropy = -\sum_{i=1}^{n} p_i \log(p_i)$$

where n is the number of classes represented in the training data, and p_i is the probability of the i^{th} class, in the set of samples covered by the complex.

The evaluation function of the learning process introduces the pruning or the "search stopping" mechanism. This mechanism is a heuristic measure of the recently generated `<complex>`. This is achieved by the classification of the `<complex>` that is based on the distance between the resulting class distribution and the default one. The generation rule process only continues if the result of the measure gets above a user-defined threshold [123]. The Likelihood Radio Statistic (LRS) evaluation function [111] is given by:

$$LRS = 2\sum_{i=1}^{n} f_i \log(f_i/e_i)$$

where n is the number of classes represented in the training data, f_i is the number of examples belonging to the i^{th} class, and e_i is the total number of examples belonging to the class that is scaled to the coverage of the complex, such as $\sum_{i=1}^{n} f_i$.

Some improvements on the second version of CN2 [38] provided the generation of unordered rules in which the quality of the complexes can be evaluated by Laplace Error Estimate function. The advantage of this new resource is its comprehensibility. This function evaluates the total coverage of the `<complex>`, rather than its performance in individual classes. The Laplace Error Estimate (LEE) is given by:

$$LEE = 1 - \frac{(f_i + 1)}{(\sum_{j=1}^{n} f_i + n)}$$

where n and f_i are defined as in the equation above. Empirical results [38] have showed that the Laplacian Error Estimative results is substantially higher accuracy than the Entropy evaluation function, especially in noisy domains.

Conclusions

The goodness of Rule Induction algorithms are on advantageous generalisations about the whole environment, based on a consequence of a limited amount of observations. As such, it is an important method to inductively build knowledge-based systems.

Using Rule Induction algorithms, we may reach wrong conclusions in cases where observations are previously faulty or are wrongly written down from an inaccurate sample. Indeed, Rule Induction systems use simple propositional-like logic representations to increase knowledge based on experience and to generate a set of unordered as well as ordered rules, thus helping the comprehensibility of the induced rule set.

Currently, CN2 analyses the variables and the preferences of each result, and discards the manual process that makes a slow search based on the number of variables and the need to make a comparison between them.

2.4 Decision Trees

The decision theory structures decision processes in situations where a choice must be made among several alternatives. Decision process plays an important role in Cognitive Science, Artificial Intelligence, and general behavioural studies. Decision process focuses on the descriptive analysis of risk, doubt and conflict, as well as its reward. However, the decision process will not be presented here in detail, we will go over the main points of this approach.

Decision Tree algorithms (DT) construct a tree-based model for a data set of objects. Objects are described by the values for a fixed set of attributes. One of the attributes is special and is called the class. The root and the internal nodes encode a test on an attribute and have one branch for each possible outcome. A leaf assigns a class value for the object that reach that leaf.

The objects are classified from training sets (seen instances) and test sets (unseen instances). This method determines a class of object by classification or regression rules with attribute values in a tree-based format. Objects can be referred to a collection of attributes, and each attribute is represented by an object feature. If there is at least an object in a set of mutually exclusive classes, and if a class of any object of the training set is known, a training set should not contain

objects with identical values to each attribute that there are in different classes. In this case, there is a conflict, and the attributes are considered unsuited to the training set of induction task. To solve this, experts can manually introduce some additional attributes.

J.R. Quinlan has popularised Decision Trees models [182] with the C4.5 tool. The Quinlan's C4.5 aims at determining the most promising strategies to achieve a goal by using graphs or decisions models and their possible results. C4.5 is a tool that builds tree models automatically from given data set of objects. Two main parts compose the Quinlan's tool: C4.5-DT [182] is a Decision Tree generator, and C4.5-rules that produces `IF...THEN...` rules based on Decision Trees generated by C4.5 -DT [180]. The tool is found for free in WEKA library respectively labelled as J48 tool [250].

Decision Trees algorithms perform a greedy top-down approach. DT starts at the top with a first decision at the root node and follows assertions down up to reach achieve the most promising decision. Analysing the training data, DT will develop branches of internal decision nodes (features) and external terminal leaves (categories). Each non-leaf node in the tree specifies a logic test of some attribute of an instance. The decision node splits its set of possible answers into subsets that correspond to different test results. Each branch descending from that decision node carries a particular test result subset to another decision node, and each decision node is connected to a set of possible answers. Usually, these branches are exclusive, that is, non-overlapping. For example, each internal node **m** implements a function test $f_m(x)$ with discrete outcomes to label the branches [5]. These sequence of steps are repeated until a satisfactory condition of the leaf node is achieved, otherwise the satisfactory condition can be on the next decision node (recursion). Finally, the DT is evaluated by a test set. Figure 2.8 was adapted from [5] to present a data set and a Decision Tree graph.

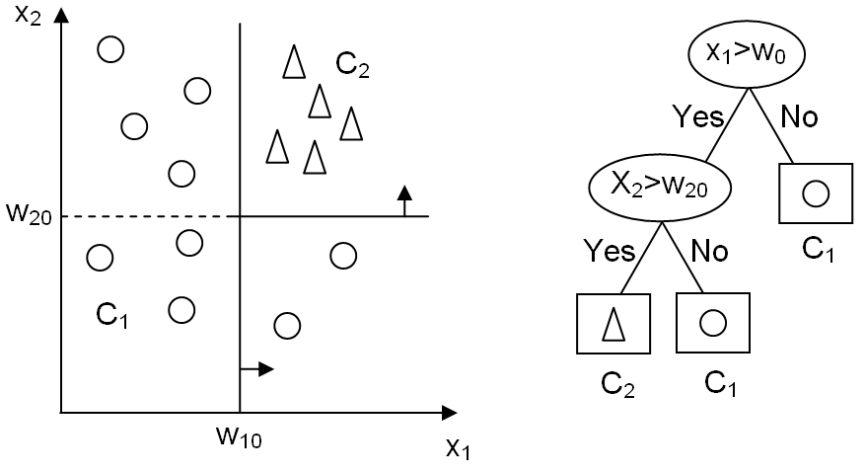

Figure 2.8: The data set (left-hand side) and its Decision Tree (right-hand side).

Figure 2.8 represents oval nodes as decision nodes, and rectangles as the leaf nodes. The data set was classified in two classes by the rectangular nodes.

Another well known DT tool is CART. CART, or Classification and Regression Tree, is considered nonparametric tree models. CART establishes a relation between a vector of predictor variables $\mathbf{x}$ and a single outcome variable $\mathbf{y}$. The variables in CART can be a mixture of categorical, interval, and continuous variables. The nonparametric technique selects the most important variables to determine the satisfactory outcome in which it can be explained from a large number of those variables and their possible interactions. The outcome variable is categorical, the right approach to solve the problem is the Classification Tree. On the other hand, if the outcome variable is continuous, a Regression Tree is the best choice. Apart from being able to perform both Classification and Regression, CART improves over C4.5 by using sophisticated tests in the internal nodes. CART may use a linear combination of attributes in the test in each internal node.

Regression Trees predict continuous dependent variables for problem solving. In this method, each leaf node is a linear model with an equation to achieve the previous value of the set of samples (training set). The unknown regression function is valued by a local regression. As any common regression technique, this method obtains a subset average defined by explicative variables (covariables). The training samples are started where explicative variables and outcome variables are known. According to Ethem [5], Regression Trees are constructed in almost the same manner like Classification Trees, except that the impurity measure is appropriate for regression.

Pre-pruning is a specific method for interrupting the growing of the tree if a non-reliable division is detected. Unfortunately, it is difficult to get a sub-optimal

tree [27]. Conversely, pruning is another method to interrupt the growing of trees by cutting off nodes. The pruning is a satisfactory technique to remove unnecessary subset tests, and to replace them with leaves or branches. Trees with low complexity have several advantages over others because they can classify correctly a large number of objects out of the training set [180]. It minimises the low reliability of error rate used to select the division into construction of complex trees (over-fitting). Cut-off nodes may elevate the error rate to the training set, and do not achieve any over-fitting.

Conclusions

Differently from statistical models, Decision Trees offer an overview of the risks and rewards associated with each possible course of action by a set of legible choices. Tree models provide a highly effective framework for common users or experts can lay out decision options (a tree-based graphic model), and get an investigate conclusion generated by those options chosen (discriminant-based).

Decision Trees became popular with the C4.5 tool. The tool classifies both categorical and numerical data, as long as the output attribute is categorical, but multiple output attributes are not allowed. The DT strengths are (a) quite simple graph mode, (b) and decisions encoded like `IF...THEN...` rules, (c) a white box model used to explain the result provided by the model, which can easily be replicated by a simple mathematical operation. Conversely, Artificial Neural Networks are considered to be black boxes because the explanations of the results can be excessively complex for any user.

Decision Trees algorithms also have an important weakness, the instability. Minimal variations in the training data produces very different attribute selections at each choice point in the tree. Consequently, all descendent subtrees will suffer the variation effect as well.

Tree models are not based on a probabilistic approach, so there is no probability levels nor confidence intervals associated with predictions derived of CART to classify a new data set. The confidence intervals represented by the accuracy of the results that were obtained by a given model or tree was based purely on its historical accuracy - how well it has predicted the desired response in other, similar circumstances. Moreover, trees created from numeric data sets can be extremely complicated to understand since attribute splits for numeric data sets represent binary values.

2.5 Genetic Algorithms

Inspired by concepts from Genomics and Charles Darwin's [47] Natural Selection theory, Genetic Algorithms (GA) are a powerful sort of search algorithm. GA implement a stochastic and parallel hill-climbing search strategy and are one of the techniques of a larger research area called Evolutive Computing. GAs were initially proposed by J. Holland [96]. The goal of GA is to study computational methods that simulate the theory of evolution [87] and use that model of Nature to find appropriate solutions to problems.

In GA, the candidate solutions are encoded in structures called chromosomes. A chromosome is a sequence of bits that encode the values of a pre-specified set of attributes or problem variables as can be seen in Figure 2.14. The GA evolve a set of chromosomes until an accepted solution (the most promising chromosome) is found. During this evolutive cycle (generation) new chromosomes (candidate solutions) are generated (population) by the application of *genetic operators*.

Each chromosomes is evaluated by a user specified fitness function. During the evolutive cycle the number of chromosomes in the population is kept constant. Since new chromosomes are generated at each cycle the chromosomes with lower fitness value are discarded (effect of the Natural Selection theory). Based on a suitability of a given organism to its environment (static fitness), and following certain iterations (reproductive cycles), GA converge to a generation of promising candidates, that is, solutions to the problem. GENESIS [90] was one of the first desktop commercial software with GA built-in, but today GAlib[158] and other research groups freely distribute the implementations of GA running on several systems and platforms.

A candidate solution is a chromosome composed of linear chains of small units named genes. The chromosome is made of an alphabet of binary digits, integers or real values to represent in the gene each independent feature (allele), as shown in Figure 2.9. Each gene has a fixed place, named *locus*, in the chromosome.

A genotype is a collection of genes and alleles to create a candidate, and a phenotype is a collection of the features of this candidate. The adaptation of each candidate is directly related to the phenotype. The traditional GA method [87] uses a binary alphabet, a fixed-length bit string chromosome and a population with a fixed size.

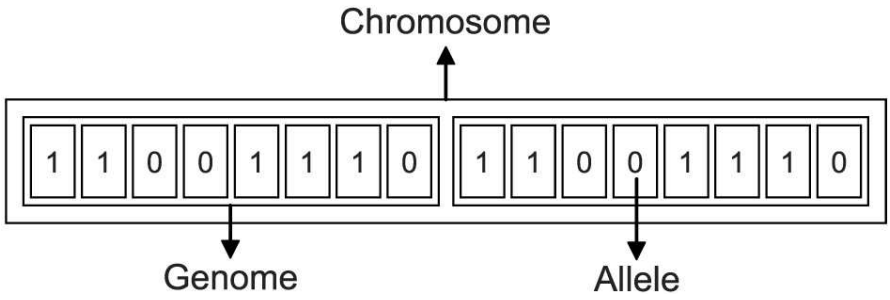

Figure 2.9: A simple artificial chromosome.

The Holland's studies [96] propose some step-by-step to run GA. Figure 2.10 shows the pseudo-code of a classic GA:

Algorithm: GA

Input: Pool of possible candidates
Output: The best candidate
begin
    Initialisation:
        Create a population of random and valid candidates
    Main:
        while (**not** final_condition) **do**
        begin
            Evaluate the fitness of each offspring
            Select the best-ranking candidates to be recombined
            Recombine the parents
            Mutate the offspring
            Replace the worst candidates by new offspring
        end
        Return the most promising candidate of the last population
end

Figure 2.10: The GA computational search method.

To be able to implement GA we need the following items:

- **Solution Candidates Encoding**: We first have to represent the problem candidate solutions as chromosomes (a linear chain of binary digits);
- **Initialisation**: We then have to set up the initial population by randomly generating a population of chromosomes (or candidates). Generally, the size of population obeys to a heuristic rule [232];

- **Evaluation Process**: Another ingredient concerns the fitness statistic score evaluation of each possible solution (chromosome). Before the fitness score, the objective score of objective function is obtained at first. The objective score is the value of error rate obtained in the test phase of that current candidate. A linear scaling function is the most used to evaluate the candidate [87, 158]. The error rate extracted from the current candidate is encoded to a proportional non negative transformed rating fitness or fitness score;

- **Selection Process**: A third ingredient concerns the selection of the most evolved candidates to reproduction. This implies more chances at spreading the candidate features in the next generation, that is, to preserve the "knowledge" of that candidate. The most used selection mechanisms are:

  - Roulette Wheel: This method selects the most evolved candidate to reproduction based on the highest fitness score relative to the remaining part of the population. Figure 2.11 [87] presents five eligible chromosomes. The roulette wheel will rotate to randomly choose the chromosome. To this example, the chromosome number five was selected. The portion of the roulette wheel of each candidate is given by:

$$Portion(x_i) = \frac{f(x_i)}{\sum_{i=1}^{N} f(x_i)}$$

    where x_i is the candidate with a $f(x_i)$ probability area to be selected.

  - Elitism: This process ensures the survival of the most promising candidate to the next generation, preserving it to participate in the next recombination process [108].

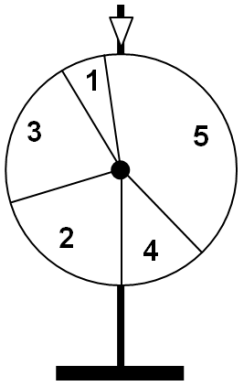

Figure 2.11: The selection method Roulette Wheel.

- **Genetic Operators**: An ingredient of capital importance is the set of genetic operators that implement the generation of new chromosomes by the

combination of existing ones. We may use a sexual crossover with one-cut point crossover technique, where the selection point is random. After that, the parents change the right side, generating two offspring and preserving the same previous population size. In sequence, the mutation operator is used that "disturbs" the chromosomes, simulating interference from the environment.

- Crossover: This operator shares information between chromosomes, offering a global heuristic to be exploited. Commonly, a single point crossover is chosen to obtain two offspring from two chromosome parents, and preserve the population size. The single point crossover [96] randomly cut the chromosome parents and change the portion of the right side between the candidates, automatically creating two offspring (new chromosomes) (see Figure 2.12). In general, the crossover operator combines two candidates with a high fitness score to create new offspring, so that the nasty offspring will be eliminated in the next generation. The crossover operator is not typically relevant to all candidates in the evolution process.

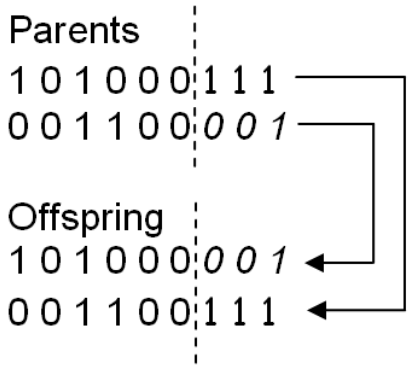

Figure 2.12: Recombination with a single point crossover.

- Mutation: The mutation operator is another method used to avoid the convergence of population to a minimum local. The mutation operator consists of changing the genetic material of chromosomes randomly selected (see Figure 2.13). Specifically, a random allele of a random chromosome will have its content changed by values represented in the alphabet previously chosen. This method offers a global heuristic to be exploited. The method guarantees that all alphabet values participate in the mutation process.

- **Termination Process**: the last process stops the GA search (a) based on the number of generations achieved, so the last generation brings the

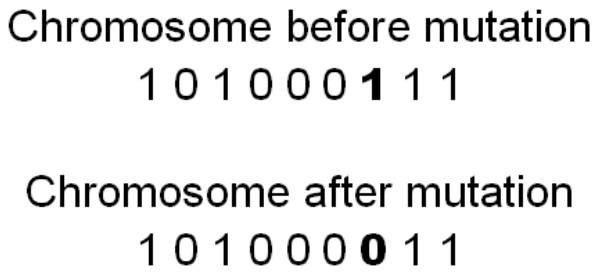

Figure 2.13: Mutation operator changing the allele value.

most promising candidates with the highest fitness scores, or (b) a value to which the best-of-generation score should converge. After that, the current population evolves.

Example: The Automatic Tuning of Artificial Neural Networks

Almost all Machine Learning algorithms have parameters that must be tuned to achieve a good quality for the constructed models. This is most often a severe obstacle to the widespread use of such systems. In this example we will test GA as a wrapper to automatically fine tune ANN parameters and obtain lower error rates.

Several studies concerning the automatic tuning of ANN parameters may be found in the literature. Most of them use Genetic Algorithm (GA) as a stochastic search method to find solutions [87]. For instance, Davis and Prado [50, 178] propose the tuning of the most usual parameter values using GA. In [91], Harp *et al.* describe a study to find a good ANN architecture by setting the number of layers and the number of neurones in hidden layers. Whitley [243] uses GA to determine the best weight of an ANN. Regarding the manual customisation of parameters, Shamseldin *et al.* [217] combine different transfer functions in a hidden layer to reach the best model with a purpose to apply them in the context of the river flow forecast combination method.

As proposed by John [105] one possible approach to overcome such a situation is by using a wrapper. This automatic tuning of parameters completely hides the details of the learning algorithms from the users. The differential of our experiment is on the use of the complete set of parameters, instead of only traditional ones. The tune that we propose includes the choice of the best ANN structure, the best network biases and their weights. The next paragraph shows all ingredients used to fine tune the ANN.

GA Ingredients

A first ingredient for using GA is to encode the chromosomes as linear chains of binary digits, using the following features, such as the learning rate (L); the momentum rate (M); the steepness rate (S); the bias for each hidden (BHL) and output layer (BHL); different transfer functions in every neurone of the hidden layers (THL) and the output layer (TOL); the number of neurones in every hidden layer (NHL).

Figure 2.14: The chromosome of ANN structure encoded in the first process.

Another ingredient concerns the evaluation of the solutions (chromosomes). Using the linear scaling, fitness function was implemented in GAlib. The error rate extracted from the current candidate is encoded to a proportional non negative transformed rating fitness or fitness score. A third ingredient to implement a GA concerns the implementation of the roulette wheel selection method. This method selects the most evolved candidate to reproduction based on the highest fitness score relative to the remaining part of the population. A fourth item required to implement GA is the combination of existing candidates and the use of a sexual crossover with one-cut point crossover technique, where the selection point is random. After that, the parents change the right side, generating two offspring and preserving the same previous population size. In sequence, we have used the mutation operator that "disturbs" the chromosomes. Finally, the last process sets the stop measure of the GA search. We chose the number of generations achieved because after several evolutional steps, the last generation brings the most promising candidates with the highest fitness scores. After that, the GA evolve.

Research Data and Experiment Design

The technique presented was evaluated using nine heterogeneous data sets from the UCI [58] repository. The classification data sets were Letter, RingNorm, Splice, Titanic, TwoNorm. On the other hand, the regression data sets are Abalone, Addd10, Boston, and Hwang. All data sets are presented in Table 2.1.

Two sets of experiments (tests) were devised in order to produce a fair comparison, as follows. In the **first experiment** the ANN was set by hand-tuning. ANN has been set to: three layers; Backpropagation learning algorithm; random

Dataset	Attributes	Examples
Letter	16	20000
RingNorm	20	7400
Splice	60	3190
Titanic	3	2201
TwoNorm	20	7400
Abalone	8	4177
Addd10	10	9792
Boston	13	506
Hwang	11	13600

Table 2.1: Data sets used in the experiments.

weights initialisation from [-0.5, +0.5]; five neurones set to Sigmoid transfer function in the hidden layer, and bias set to value 1; one neurone set to Sigmoid transfer function in the output layer, and bias set to value 1; learning rate, momentum rate and steepness rate set to 0.8, 0.2 and 1, respectively; the training phase stops if the error rate gets below 0.1 or the training epochs gets below 50. In the **second experiment** the ANN was set by using a GA-based wrapper. GA has been set to: 50 candidates; 30 generations; mutation probability set to 1%; crossover probability of 90%; population replacement percentage set to 25%. Consequently, the ANN has been set to: Backpropagation learning algorithm; random weights initialisation from [-0.5, +0.5]; three times more neurones in the hidden layer than in the input layer; one out of seven transfer functions in each hidden and output neurones, and bias set to value 1; one neurone in the output layer, and bias set to value 1; learning rate, momentum rate and steepness rate set to respective default internal range; the training phase stops if the error rate gets below 0.1 or the training epochs gets below 50.

Experimental Results

The experimental results are reported in Table 2.2. Both the average error rate and the std. deviation of training and test data sets are presented. The average represents the result obtained from the arithmetic sum of five cycles (K-fold technique) of the same data set together and then the total is divided by the number of cycles. The winner result percentage was obtained by the variation coefficient of the tests. The values in Table 2.2 were all multiplied by 1^{10}.

From Table 2.2, the following conclusions can be drawn. First, it is worth noting that all tests listed in this study show good results, but error rate decreased because of the use of wrapper. Second, as can be observed, the use of GA turns

Data set	ANN hand-tuning (Technique:T1)	ANN with GA (Technique:T2)	Winner
Letter	123.6(13.8)	102.6(1.9)	T2(9.3%)
RingNorm	981.7(123.0)	267.9(10.7)	T2(8.5%)
Splice	16.8(2.2)	16.8(0.6)	T2(9.5%)
Titanic	11.3(0.9)	9.5(0.5)	T2(3.3%)
TwoNorm	475.2(257.2)	141.8(23.5)	T2(38%)
Abalone	99.9(7.5)	70.1(5.1)	T2(0.2%)
Addd10	31223(950)	30460(855)	T2(0.2%)
Boston	107982(27495)	84110(17309)	T2(4.9%)
Hwang	1342.8(15.9)	1174.6(24.6)	T1(0.9%)

Table 2.2: Comparing hand-tuned ANN with ANN tuned by GA.

out to be better in three cases out of nine due to the structural risk minimisation principle of ANN, such as local minima and overfitting.

Under a wrapper technique, GA are the most used fine tuning technique to draw the best ANN structure by choosing the correct transfer functions, biases, and other essential ANN parameters.

Advantages

The advantages of GA from a common user perspective include the following:

- In a search method, GA strengths come from the fully parallel blind search on the solution space. GA drive the search to promising areas via a population of potential candidates, minimising the risk of searching for a solution in a maximum or minimum local;
- The blind search, referring to known only the necessary candidate cost function;
- As adaptive algorithms for solving practical problems, GA create new variants to generate a good chance of finding better solutions;
- GA use simple search methods that do not require extensive knowledge in the search space. Traditional non-linear solution techniques, such as solution bounds or functional derivatives are not used because, as a result, they cannot always achieve an optimal solution;
- GA prevent the optimisation problem being trapped in local minima or maxima by two methods: (a) the initial random population generated is a

multidimensional global sample of the whole solution space; (b) variation-inducing tactics, that is, crossover and mutation;

- GA combine representation of candidate solutions and problem-specific genetic operators [96], in which there is a trend to good solutions each time an evolving process is re-started [87]. This way, GA solve problems by collecting knowledge accumulated in earlier iterations about the problem and using knowledge to create acceptable solutions.

Disadvantages

The drawbacks of GA include:

- GA are a highly simplified system compared to the actual traditional evolutionary theory;
- Whenever multidimensional systematic searching would be technique of choice, except that the large number of comparisons make that approach intractable;
- it is difficult to encode chromosomes;
- Due to the probabilistic development of the solution, GA do not guarantee optimality even when it may be reached. However, they are likely to be close to the global optimum. This probabilistic nature of the solution is also the reason they are not contained by local optima;
- GA need to couple with other search techniques to overcome the rapid local optimisation.

Conclusions

Genetic Algorithms have been introduced as general stochastic search algorithms based on metaphors of natural selection and natural genetics. GA are robust to find the most promising hypothesis in a huge space of candidates by analysing simultaneously each new generation. To create a population, a pool of possible solutions is encoded like a chromosome. There are three central operators behind the GA method, such as selection, crossover and mutation. GA can be driven to cover fine tuning learning algorithms, such as ANN parameters; consequently, a hybrid system is composed.

The main benefits of GA are the unsophisticated operations, the easy implementations, the effectiveness in search of a global maximum, the applicability in situations where a mathematical model is unknown or imprecise, and in linear

and non linear functions. Conversely, the major limitation of this algorithm is its limited accuracy caused by discretisation of the search space, implied by the use of a fixed binary representation.

2.6 A Multi-Strategy Learning example

To illustrate the decision power of heterogeneous ML algorithms working together in a decision system, we elaborate a decision problem based on RoboCup Rescue domain.

The main goal is to decide whether fireman can reach the civilian position on time to perform a rescue action and left it in the nearest refuge instead of extinguishing fires in burning buildings with the aim of preserving the city. For both modules, the objective is to obtain correct responses to situations that do not belong to the training set. The civilian is somewhere in a burning building in the city. Typically, a rescue decision is taken based on position, agent's and civilian's life conditions. The system devised is composed by two modules that encode the fireman and the civilian decisions, respectively labelled *Civilian (FF)* and *Fireman (FF)*, a module that encodes the ambulance decision, called *Ambulance (CN2)*, and a fourth module that combines fireman and ambulance decisions, labelled *Rescue (J48)*. Figure 2.15 shows the modular and heterogeneous structure of the decision system, assembled in AFRANCI[2]. The fireman and civilian decision modules are encoded using Feedforward ANN, the ambulance module was constructed using CN2 Rules Induction algorithm [41], and the last module was constructed using WEKA J48 Decision Trees algorithm.

2.6.1 Part One

We now describe only *Civilian (FF)* and *Fireman (FF)* ANN modules are explained. To train the module *Civilian (FF)*, Figure 2.16, a data set was prepared with some ingredients: independent variables that include the coordinate (X,Y) of the civilian, the life measure of the civilian, the difficulty of the civilian rescue situation, and the coordinate (X,Y) of the nearest refuge (rescue building). The goal is to know whether the civilian can be rescued on time to be left in the nearest refuge. Additionally, to train the module *Fireman (FF)*, Figure 2.17, a data set was prepared with some ingredients, being two of them are feeding the module *Civilian (FF)*. In general, the data set is composed of independent variables that include the coordinate (X,Y) of the fireman and the coordinate (X,Y) of the civilian, and the life measure of the fireman[3]. ANN has been set to: three

[2]In AFRANCI, users design the whole system structure using drag-and-drop operations. In the final step, a set of C++ instructions is automatically encoded.

[3]A measure between 0 and 100 of the energy the fireman can use.

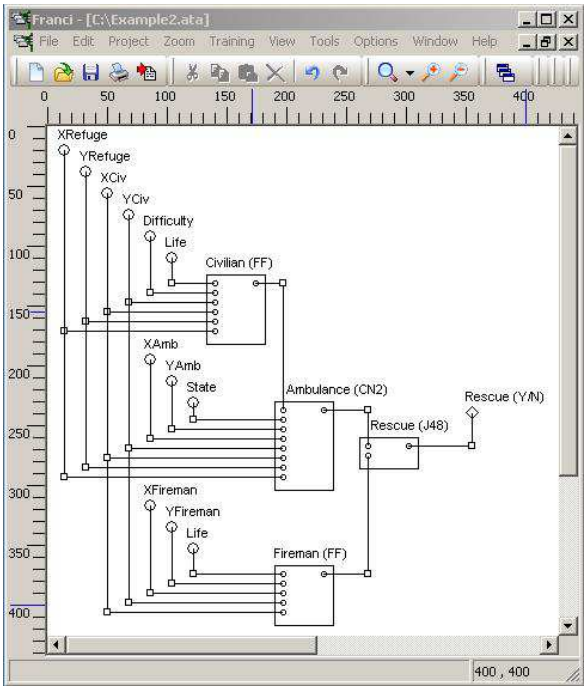

Figure 2.15: The Rescue decision system.

layers; back-propagation learning algorithm; random weights initialisation from [-0.5, +0.5]; 10 neurones set to Binary Sigmoid transfer function in the hidden layer, and bias value set to 1; one neurone set to Binary Sigmoid transfer function in the output layer; learning rate, momentum rate and steepness rate set to 0.8, 0.2 and 1, respectively; stop the training phase when the error rate gets below 0.05 or the training epochs reaches 100.

In modules, we use ANN generalisation capacity to correctly predict all the other values. Generalisation means the property to get the right answers to questions not previously seen in the examples.

The modules *Civilian (FF)* and *Fireman (FF)* modules are capable of estimating continuous functions by observing the relation of output data with the inputs. The module *Civilian (FF)* is responsible for verifying whether the civilian can be rescued on time to be left in the nearest refuge, and *Fireman (FF)* is responsible for verifying whether fireman can rescue civilian on time instead of extinguish fires. Once the modules have learned the desired relationship between the input and output data presented during the training phase, it is probable that they give right answers to other problems of the same type by means of generalisation.

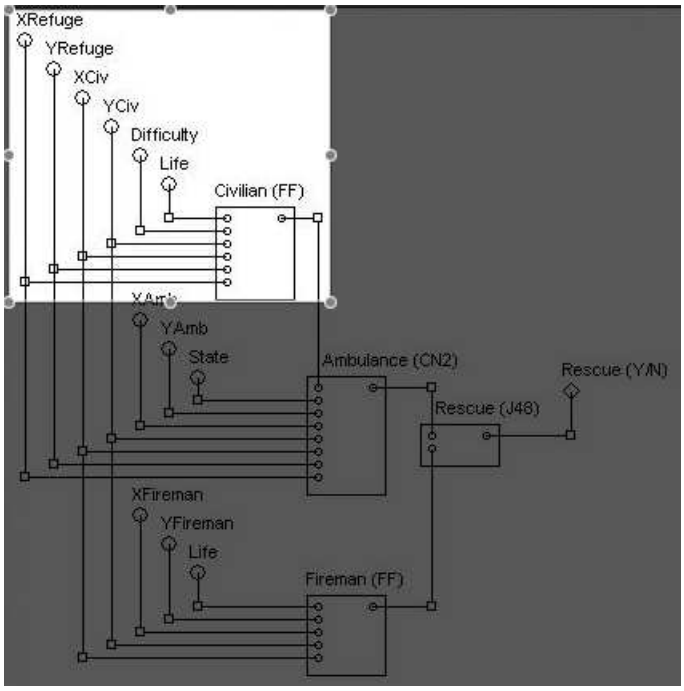

Figure 2.16: The module *Civilian (FF)*.

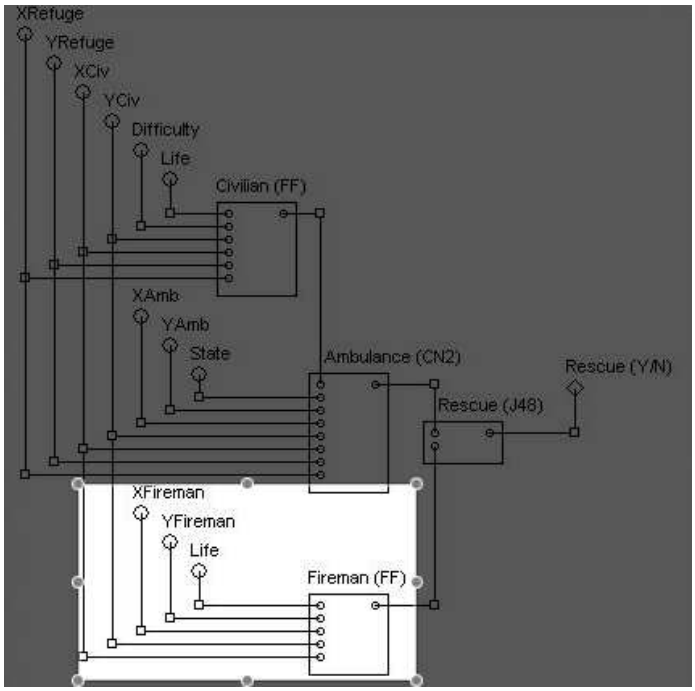

Figure 2.17: The module *Fireman (FF)*.

Advantages

The greatest advantages of Artificial Neural Networks compared to classic decision making methods include:

- Artificial Neural Networks possess the ability to implicitly guide an analysis with data both in linear and multivariate non-linear problems; consequently, detecting all possible interactions between dependent and independent variables. ANN also learn these relationships directly from the data being modelled, and there is not necessary any prior assumptions about these relations;
- Not surprisingly, the massive paralleled centre of units and the interconnections gives to ANN a high processing speed and data compression, as presented in Figure 2.6, over conventional well-ordered rule algorithms. For this reason, the ANN also have been dubbed the connectionist approach [67];
- ANN have capacity (a) to obtain meaning from imprecise or complex inputs, (b) to adjust their synaptic weights to any situations, and (c) even to model a complex decision system;
- The plasticity summed with parallelism, empirically "inherited" from biological neural networks, can achieve skills, such as robustness to noise, learning rate, generalisation and adaptability, association, and rule-like behaviours without hand drafted rules. For instance, ANN are capable of analysing the data, even if the data are incomplete or distorted. Distributing information redundantly on their axons, ANN can build a robust fault tolerant system;
- ANN learn from observed data (experience) by the arbitrary approximation function. In a non restricted pre-fixed sequential order, ANN also identify instances that are unlike any which have been observed before showed to the network;
- ANN can be trained to recognise uncommon events with a high degree of accuracy. ANN use past experiences to gain the ability to apply this knowledge to identify unknown instances. The probability of any action may be estimated and a potential response be flagged whenever the probability exceeds a specified threshold;
- ANN adapt their analysis of data in response to the training. The output of ANN, typically expressed in the form of a probability, provides a predictive capability to identify a particular event or pattern. Pattern represents a package of predesigned "chunks" (decisions) that have already been made and can be reused.

Disadvantages

Conversely, the ANN weakness are the following:

- The individual relation between the input variables and the output variables are not clearly presented so that the model tends to be a "black box" or input/output table without analytical basis;
- The proneness to overfitting requires a considerable computational burden to be minimised;
- The connection weights and transfer functions of the various network nodes are usually "frozen" after the network has reached an acceptable level of success in the identification of events;
- The training routine requires a large sample size to ensure that the results are statistically accurate;
- While the network analysis is searching for a sufficient probability of success, the basis for this level of accuracy is not often known;
- There is no cookbook which explains how to fine tune the technical properties of ANN.

2.6.2 Part Two

This part extends the description of the previous one about whether ambulance or fireman should rescue or not a civilian to a nearest refuge. For now, we try to solve whether ambulance is apt to rescue a civilian based on its actual conditions. This kind of decision-making problem is performed at *Ambulance (CN2)*, Figure 2.18.

The module *Ambulance (CN2)* is responsible for verifying whether the ambulance is entirely apt to rescue a civilian. To train the module *Ambulance (CN2)*, a data set was constructed with some inputs where four of them are feeding the module *Civilian (FF)*. The inputs are: independent variables that include the coordinates (X,Y) of the civilian and the coordinates (X,Y) of the nearest refuge, the coordinates (X,Y) of the ambulance, and state[4]. The CN2 algorithm has been set to: ordered rule list; Laplacian error estimative; threshold set to 0.8; and Star set to 5. The goal is to know whether civilian can be rescued on time to be left alive in the nearest refuge.

[4]The state attribute defines whether ambulance is free or busy.

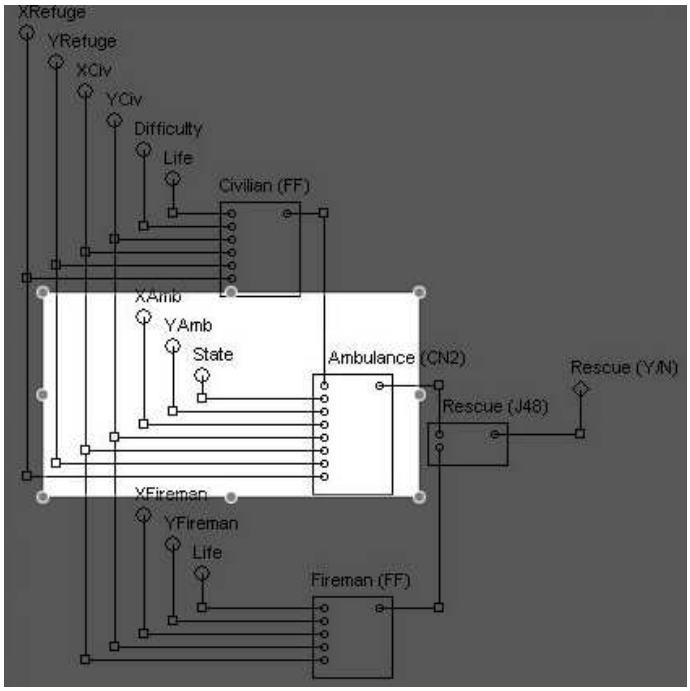

Figure 2.18: The Rule Inducer module *Ambulance (CN2)*.

In order to verify whether the ambulance is apt to rescue a civilian, Figure 2.19 establishes below the decision rules.

The rules induced by CN2 establish that: (rule 1) if the ambulance is occupied then it is useless to attempt the rescue; (rule 2) if the civilian has not sufficient time-life then it is not rescued; (other rules) the civilian will be rescued if it has sufficient time-life and the ambulance is between the civilian and the rescue place, otherwise it will not be rescued.

Advantages

The advantages of a Rule Induction algorithm are the following:

- Rule Induction algorithm are tolerant to noise. It correctly assigns examples to into sub-spaces, and classify all known and unknown examples in the training;
- Rule Induction methods simulate the non monotonic reasoning, that is, the reasoning based on general rules and accepting exceptions;
- Rule Induction provides a set of alternative rules;

```
IF state = busy
THEN class = n [52 0]
ELSE
IF civilian = notapt
THEN class = n [20 0]
ELSE
IF Yamb > 8164.50
 AND Xciv < 9204.50
THEN class = y [0 7]
ELSE
IF Yamb > 340.00
 AND 386.00 < Yciv < 7731.50
 AND Xrefuge > 3800.00
THEN class = n [9 0]
ELSE
IF Yamb < 5721.00
 AND Xrefuge > 866.00
THEN class = y [0 8]
ELSE
(DEFAULT) class = n [4 0]
```

Figure 2.19: The ordered rule list generated by CN2.

- The rules generated are "white boxes", and therefore can generally be understood and validated by domain experts. The rules learned with these algorithms can be used to predict information about new objects;
- The familiar structure of syllogisms, `IF condition 1 AND condition 2, ..., THEN conclusion` are a very powerful representational language, and can be fully applied to explain the meaning of the rules;
- Rule Induction algorithms assist users on rationale choices that were derived by observation so that patterns are discovered in a data set and the most promising set of decision rules is collected;
- CN2 is sufficiently robust to induce an ordered or an unordered list of `IF...THEN...` rules in domains where there might be noise [38]. Clark and Niblett [39] classify the noise into two different causes that are (a) Errors due to transcription: whenever an example situation is presented to a learning algorithm it must be described in some manner. The process of recording and transcribing the attributes of an example is prone to error due to several causes. For example, imperfect measuring equipment, mistaken classification by an expert, typing errors and so on, and (b) Errors due to an insufficient description language: a description language should provide

the resources completely and correctly, and classify all possible situations in a problem;

- Recent improvements to the output of the rules form have induced a set of covers where CN2 builds an expression `IF...THEN...ELSEIF...THEN...` structure that is called a rule list [203]. The important point is that the semantics of each individual rule depends on the previous one, that is, a rule will trigger when all the previous rules must have failed. This feature of the rule list excludes the possibility of a clash during the classification process, but increases the difficulty to be interpreted by humans.

Disadvantages

The general problem to create and test rules can be resumed in a NP-complete problem. It is well known that some questions have still no answers. For instance, (a) What is the best the stop measure of the number of rules generated? (b) How many hypotheses should the final theory have?

2.6.3 Part Three

In the previous parts one and two, we presented the rescue civilian problem, and explained some of the decision modules that compose the decision system to rescue a civilian in the RoboCup Rescue Domain.

In this last part, we present the module *Rescue (J48)* Decision Tree, Figure 2.20. This module will deliberate in favour of the apt agent to rescue the injured civilian. In order to deliberate, the *Rescue (J48)* Decision Tree module receives two outputs, each one from *Fireman (FF)* and *Ambulance (CN2)*, as was presented in Figure 2.15.

The *Rescue (J48)* is responsible for deciding which agent will rescue the civilian. The system devised is composed by two modules that encode the decision of the fireman and the civilian, a module that encodes the decision of the ambulance and a third module that combines fireman and ambulance decisions. The system will deliberate in favour of the fireman agent to rescue the civilian only if the ambulance agent is not able to do this. In case of both fireman and ambulance agents are capable of rescuing the civilian, the module *Rescue (J48)* decides in favour of the ambulance agent. This happens because a fireman agent has other priority such as to extinguish fires in burning buildings with the aim of preserving the city. Figure 2.21 presents the results of J48 Decision Tree generator.

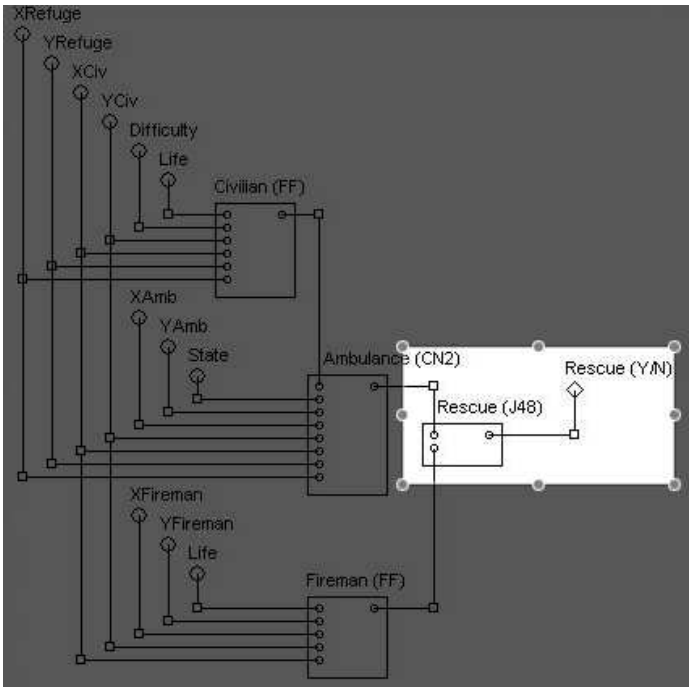

Figure 2.20: The *Rescue (J48)* Decision Tree module.

```
J48 pruned tree
------------------
ambulance_apt = TRUE: RescueAmbulance (7.0)

ambulance_apt = FALSE: RescueFireman (3.0/1.0)
```

Figure 2.21: Decision Tree generated by WEKA J48 learner.

Advantages

Decision Trees have the following advantages:

- Decision Trees produce tree models that are easy to be interpreted by human experts. They satisfactorily explain by graph why a decision was made;
- Tree models are useful to map any kind of data into groups. DT analyse data to highlight the relationships of a large number of candidate input variables to an output variable;
- Decision Trees determine the most promising strategies to achieve a goal by a set of business rules. Some of these strategies are: the memorisation of several internal states to be used by the classifier, the use of confidence measures, and the avoidance of conflicts among them;

- Decision Trees can be satisfactorily translated to a set of business rules, in which it can also be represented;
- Tree models implement the divide-and-conquer strategy. Typically, a problem is recursively divided into sub-problems to find a solution. Each sub-problem identifies some patterns in the data set. A solution is achieved when each subset in the partition contains cases of a single class, or when no test offers any improvement;
- Decision Trees can model uncertainty, predicting categories for new events;
- Decision Tree methods make no prior assumptions about the distribution of the data (nonparametric). Because the nonparametric technique selects the most important variables in order to determine the satisfactory outcome, a large number of those variables and their possible interactions can be explained.

Disadvantages

Although DT have numerous advantages over other types of Machine Learning algorithms, including the ones just described, DT have some disadvantages:

- Decision Trees need as many examples as possible to generalise, so the number of possible outcomes in the model can be extremely large. Similarly, some DT models tend to over-generalise with many examples. The same symptom is also caused by not much training data;
- Unfortunately, many divisions form large and complex trees that generate over-fitting. In order to generate simple trees and avoid over-fitting, a set of training cases should not be divided any further (stopping or pre-pruning), but some of the structure built up by recursive partitioning should be retrospectively removed (pruning) [5];
- Large DT or DT created from numeric data sets can become difficult to be interpreted. On the one hand, larger DT can be more consistent; on the other hand smaller DT generalise better.
- Developing and reaching agreement on regression may be difficult;
- Decision Tree engine requires more computation resources than Finite State Machines;
- DT obey a sequence of tests, being dependent of the structure generated;

- Some learned DT may contain errors and become unstable. Similarly, small variations in the training data can generate weird looking trees. This is commonly caused by problems with sparse data set (randomised data set) or small data set. In order to correct this instability, cross-validation is used to force the utilisation of the entire data set to train and test the model, and stratified sampling is used to balance the class distribution between them.

2.7 Conclusions

Symbolic and connectionist approaches, if rightly combined, they can obtain a complex decision system, always looking for improved performance in agents.

Machine Learning algorithms employ deduction or induction techniques to encode knowledge in a form of synaptic connection weights or production rules until a satisfactory description of each class is obtained. For instance, agents can behave or "think" under certain circumstances, such as to survive and respond in the world.

Using Supervised Learning (SL) method, a fast learning rate was obtained, instead of just letting the algorithm work out for itself. The most common heterogeneous ML algorithms for agents interact and reach goals in the world were Feed-forward Artificial Neural Networks, Rule Induction Algorithm, Decision Trees, and Genetic Algorithms.

Chapter 3

Architectures for Autonomous Agents

Architecture is an archetype of a complex structure composed of interconnected elements arranged in a specific manner. Agents are entities commonly made of a complicated structure of interconnected and organised elements. In this chapter, we analyse the influence of architecture styles on the development of agents, and describe a set of architectural styles adequate to construct cognitive agents. We also discuss a set of concepts regarding how intelligence can be implemented in agents. We review typical architectural styles used to assemble agent skeletons, and analyse classic architectures for agents suitable for the development of robust agents. Furthermore, a brief explanation about meta-architectures and their importance is also addressed.

3.1 Introduction

Generally speaking an architecture in software engineering can be conceptually interpreted as "structure" or "organising principles" of a system. Architectures specify the set of components (or modules) of a system and the way they are arranged and interconnected. Procedure calls may be used to promote communication among heterogeneous and independent modules. Modules are used to drive attention at an appropriate goal of the system without delving into programming details. In a complex system modules do not work alone, but they sum efforts from their parts (relationships among them) to achieve goals. They interact with each other by means of interfaces to share details, results, to produce signals, or

behaviours and possibly to generate actions. Depending on context, a system can comprise more than one architectural style to synthesise a complex behaviour.

Although there are many types of architectures they have similarities. There are common principles, design philosophies and theories that can be successfully applied to a wide number of types of structures. For instance, we can include learning algorithms in the modules, in an agent, architecture in the same way specific routines fill components in a software. It is possible to reuse or transfer what was learnt from one architecture into new ones.

To develop a good architecture, it is important to understand in detail how a system will behave to achieve the goals and what is an agent. Architects are the ones who understand the system context and establish what goals the architecture will support. They also design the system, encode the operation strategy into technical strategy (meta-architecture). The compilation of the architecture encodes deliberative reasoning into mechanisms of response more efficiently. The architecture establishes how technology will be used to deliver operation capabilities, setting direction for the architects and development community. The emergence of behaviours is dependent of the arrangement of modules in a specific style. Styles enhance the understandability and re-usability of the architecture, as a result of exhaustive generalisation and specialisation (decomposition) of huge holistic systems.

In order to clarify what agents are, or how agents can be autonomous, we present some classic agent definitions that can be found in the literature:

"An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors."[1] [212].

"Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realise a set of goals or tasks for which they are designed." [141].

Based on the definitions presented above, we define an autonomous agent as an adaptable software or hardware entity in the sense of controlling over its own actions and internal states. In other words, agent is a fully functioning system, like a biological, or simulated in software, or implemented in the form of a computational entity, that has an integrated collection of several heterogeneous but interconnected capabilities arranged according to an architecture.

In order to plan an agent architecture, it is important to understand the problem domain in details. For instance, agents can be planned simply to sense-act in response to their environments, decide some actions based on production rules or prewired connections, or store past environment states in order to remember

[1] Also labelled as rational agent.

and make plans for further decisions. In addition, the arrangement of modules influences the emergence of behaviours.

On the one hand, an agent that manifests a reactive behaviour is characterised by the absence of an inner classic symbolic environment model, dealing with short-term changes, and acting freely in accordance with stimulus-response from changes in the environment. It is worth noting that the terms reflexes and reactions have different meaning. For instance, the agent triggers a chain of several reflexive rules in order to produce a reactive behaviour. In this sense, Brooks [28], and Agre & Chapman [2] argue that agents can emerge "intelligent" actions without symbolic representations, so typical of traditional Artificial Intelligence. On the other hand, an agent that deliberates decisions is characterised by the use of a previous symbolic model of the world, dealing with long-term changes, and reasoning about what actions to do next. The ability to reason symbolically is based on knowledge stored in a form of a set of symbols. Reasoning represents the manipulation of these symbols, to make judgements and decisions that are logically valid. Oriented reasoning is a result of adaptation of its own rules by a sequence of specific actions,, scuh as planning to solve a problem.

In this chapter, we are surveying different types of architectural styles by software engineering and classic architectures to develop our own system that will be introduced later on.

3.2 Architectural Styles and their Control Levels

Artificial Intelligence attempts not only to understand, but also to develop intelligent entities [212].

Recently, new agent architectures are being designed based on the concept of modular structure, and modular structures are simple to be implemented. The selected criteria for the decomposition of a system in small pieces of function impacts on maintenance reuse, increasing consistency, integration among systems, and portability.

Autonomy is highly correlated with structural complexity. As the autonomy of an agent increases so increases complexity of its structure. In such cases it is advisable to adopt the principle of loose coupling by decomposing the system into reasonably independent modules making complex systems tractable.

On the one hand, system decomposition addresses such concerns as complexity, portability and flexibility. To handle complexity one applies the principles of separation of components and "divide and conquer". To address portability and flexibility one applies the principle of identifying areas that are probable to change. On the other hand, system composition addresses other concerns as

integrity mechanism that include internal balance, compatibility and harmony among the parts, as well as fit to context and to purpose.

The next sections present the key factors for the development of architectures as an understandable and manageable system, being better provided by architectural styles.

3.2.1 Architectural Styles

Architectural styles, also known as system patterns, represent a family of systems in terms of patterns of a structural organisation and their combination to guide less experienced architects in designing new architectures [219].

Typically, architectural styles determine exactly what a system looks like. In short, an architectural style defines a system providing the following information:

- A set of component types that defines the *locus* of computation. For example, a process that performs some functions at runtime;
- A topological layout of these components indicating their runtime interrelationships;
- A set of connectors that mediate interactions (communication, coordination, or cooperation) among components.

In general, architectural styles specify the circumstances in which should be relevant to plan and to construct the architecture of autonomous agents. We now present the architectural styles studied [83, 219, 33, 17], relevant to the development of the architecture we propose in this thesis.

3.2.1.1 Data-flow Architectures

Data-flow architectures represent systems that transform input into output or some final destination by a sequence of conversions one at a time. Having the properties of reuse and modifiability in mind, it is easier to build a data-flow architecture by simply arranging modules (blocks) in different manners.

Data-flow architectures may be divided into two subtypes, batch sequential, and pipe-and-filter.

Batch Sequential

In the batch sequential style, components are independent programs that obey a sequential processing completion flow, which means components will

start only if previous components have performed their tasks. Each batch of data is transmitted as a whole between the steps. A classical data processing is the most common application for this style. This style is illustrated in Figure 3.1

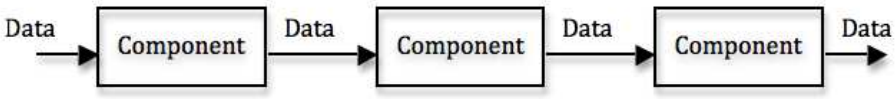

Figure 3.1: The batch sequential style.

Pipe-and-filter

The pipe-and-filter style transforms data incrementally by successive components. An incremental process means that later processes are started after the earlier ones have finished. Filters are independent programs (stream transducers) that load streams of data on their inputs, perform some computational task or data compilation, retain no state information between examples, and produce streams of data on their outputs. The transmission of data happens using pipes between steps. Pipes are used like channels of data-flow for the streams, passing outputs from one filter to the inputs of others. This style is flexible to build the system with blocks, and arranging them in parallel, as can be seen in Figure 3.2.

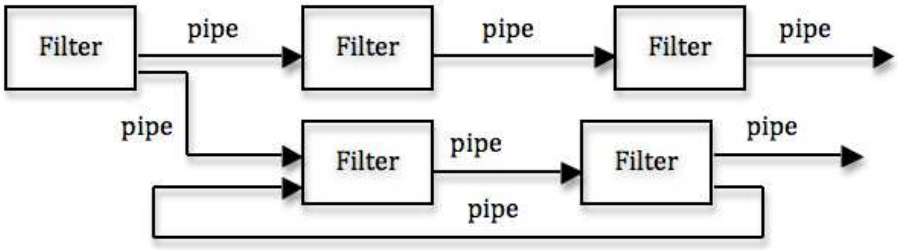

Figure 3.2: The pipe-and-filter style.

Advantages and Disadvantages

In batch sequential style controls are easily handled. Controls represent a collection of simple and atomic components, having neither concurrency nor interactions between those components. The major problem of this style is that the system has a tendency to become big and sometimes slow in time. Conversely, pipeline style produces fast first outputs, which is very useful in behaviour-based systems. Unfortunately, pipeline style may be too complex to

program because process operates incrementally. Its cyclic structure supports feedback and loops, that is, later processes can start before the earlier ones have finished.

3.2.1.2 Call-and-Return Architectures

Call-and-Return architectures are designed for agents in which modifiability and scalability are important issues. In the following paragraphs, we present some variations of that architecture.

Main-program-and-subroutine architectures

The main-program-and-subroutine architectures are based on the hierarchical decomposition of main program into subroutines; a typical programming paradigm that helps to achieve modifiability. Traditionally, each module in the hierarchy supports a single thread of control. The hierarchical reasoning will only perform correctly if a dependent subroutine transmits the correct data. The goal is to increase performance by distributing the computations and taking advantage of multiple tasks. Figure 3.3 shows the main program that delegates tasks to be performed by subroutines.

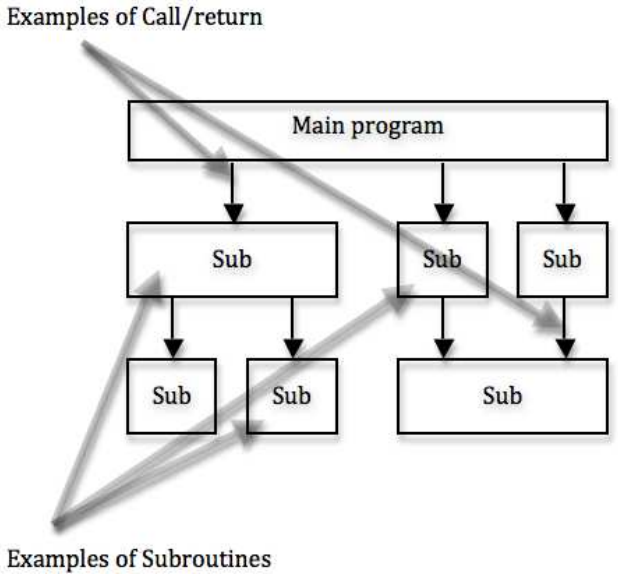

Figure 3.3: The main-program-and-subroutine style.

Object-oriented architectures

Object-oriented architectures represent the inner evolution of call-and-return architectures. In this style reuse and modifiability is achieved by encapsulated internal operations. The object-oriented paradigm emphasises: (a) the object definition; (b) the knowledge of how to manipulate and access data; (c) defines responsibilities; (d) the collaboration of the different objects. This style uses components, like black boxes and interchanging of data as represented in Figure 3.4.

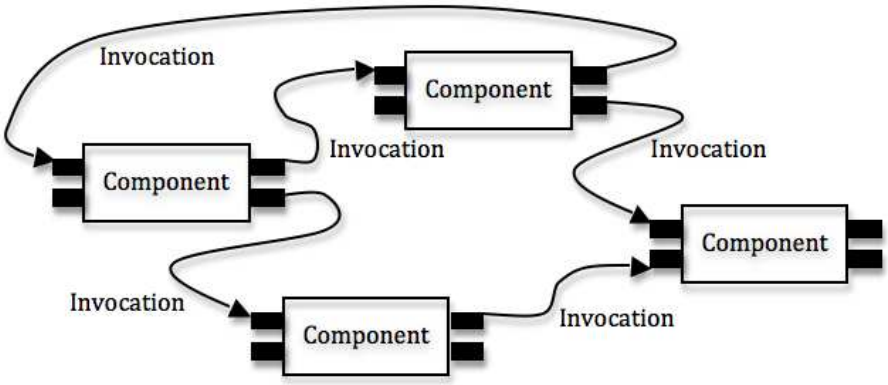

Figure 3.4: The object-oriented style.

Layered Architecture

A Layered architecture arranges components in layers. In the pure version of this style, each layer communicates only with its immediate neighbours, as we can see in Figure 3.5. A Layer bridging is a variation of the basic model in which a layer may interchange messages with others that are not its immediate neighbours.

Advantages and Disadvantages

The main-program-and-subroutines style is similar to the top-down or hierarchic reasoning. This sort of style is very useful to develop an emergent reasoning in agent structures. But, the correctness of modules/subroutines depends on the correctness of the subroutine it calls. Object-oriented style uses encapsulation to hide certain information and offers management of objects. In this style, we can interpret the idea of atomic control modules encapsulated details from others and sharing only the necessary data. The Layered style behaves like an organisation was divided in sub-levels. A module, sub-system,

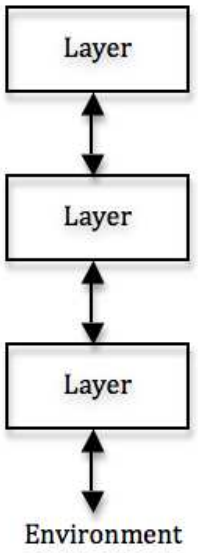

Figure 3.5: The layered style. Each layer represents an abstraction level.

or layer represents in the organisation a highly coherent set of functionalities, which suggest high internal coupling and low coupling to external entities.

3.2.1.3 Independent Component Architectures

Independent component architectures represent structures of independent modules communicating among them through messages. The strongest feature achieves modifiability by re-use and evolution. New modules can be easily attached to the structure and run in parallel. The components interact through the exchange of data only when they are selected, which leads to integration of environments.

Independent component architectures have two subtypes: Event-based architectures, and Communicating processes.

Event-based Architecture

In Event-based architectures, a message manager receives data from independent components (see Figure 3.6). Components listen/announce data to the messenger in order to be up-to-date. After receiving a message, the message manager forwards it to the component already subscribed that wish to receive the data previously announced. Message manager controls the interchange of messages among subscribed components, but it does not directly control the component execution.

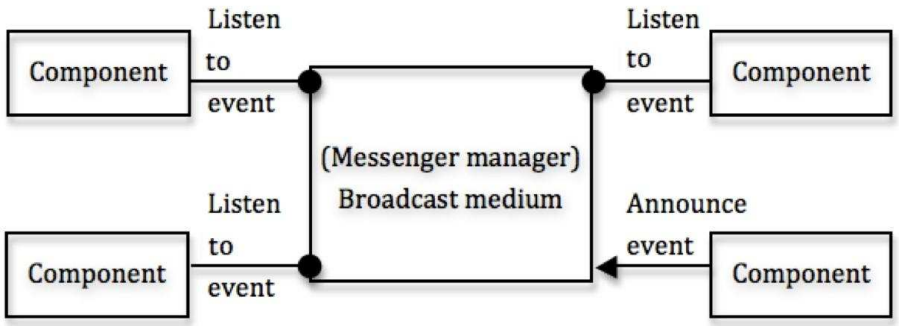

Figure 3.6: The event-based style.

Communicating Processes

Communicating processes are multiprocessing systems capable of scaling up without quality of service lost. This model is characterised by having a main component that serves data to one or more components connected to it across a network. The components request data to the main component by a call, which works, synchronously or asynchronously, as depicted in Figure 3.7. Processes are like black box modules interchanging messages among their pairs. The difference between Communicating Processes and Event-based style is determined by the autonomy of the components having or not their own control. For instance, if the main component performs tasks synchronously, it returns control to the component at the same time that it returns data. If the main component performs tasks asynchronously, it returns only data to the component that which has its own thread of control.

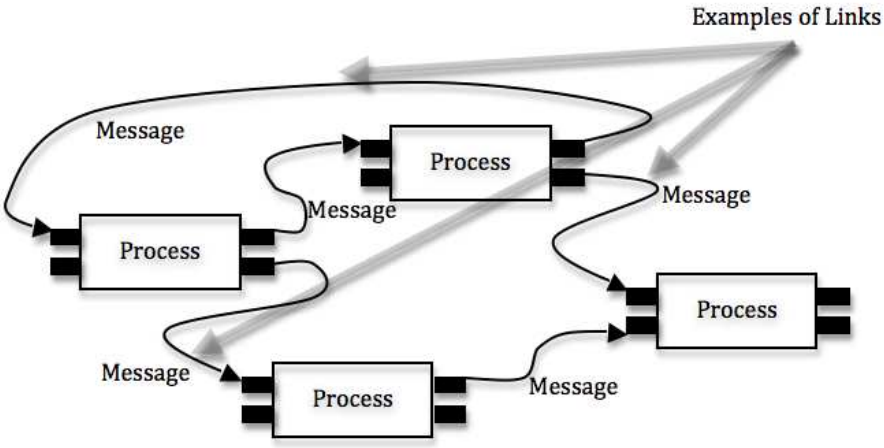

Figure 3.7: The communicating Processes model.

3.2.1.4 Data-Centred Architectures

Data-centred architectures are structural systems composed by client and shared data. A client is an independent execution module that accesses and updates shared data in a passive repository, like a file, or in an active repository, like a blackboard. The major difference between passive and active repository is the existence or not of notification messages. For instance, a blackboard sends notifications to subscribers when data of interest changes, and for this reason it is labelled as active. A blackboard has bidirectional control arrows to "interchange" shared data. The diagram shown in Figure 3.8 represents independent components sending/receiving up-to-date data.

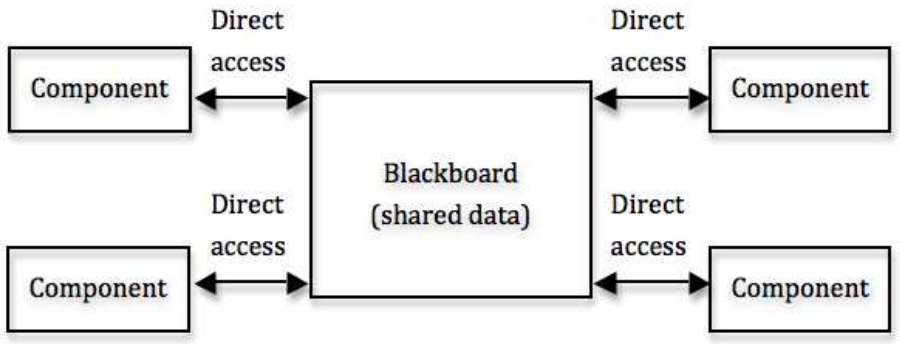

Figure 3.8: The data-centred style.

The advantages over other systems is that the clients are relatively independent of each other, and the data storage is independent of the clients, thus the style enables the development of scalable layers. New clients can be easily added, or changed with respect to the functionality. Coupling among clients will reduce this benefit but may occur to enhance performance. The weakness of these systems is the blackboard feature itself that can behave as a bottleneck of the system.

3.2.1.5 Virtual Machine Architectures

Virtual machines mimic some functionality (behaviours) not native of the system on which they are running. They allow the simulation (and testing) of behaviour-based models, as explained earlier in this chapter, complex decision processes, and even the simulation of uncommon situations that need fast agent attention to be solved both by reasoning and action. For instance, Rule-based systems are the most common example of virtual machine architectures, as depicted in Figure 3.9.

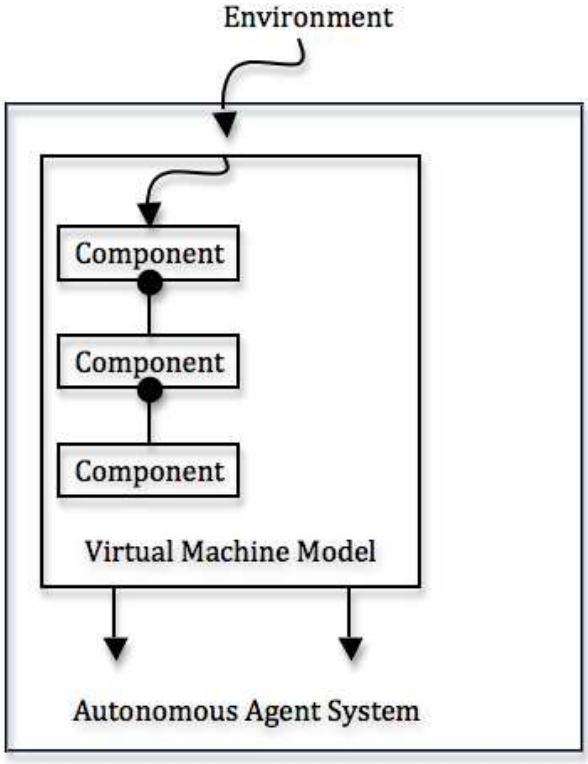

Figure 3.9: The virtual machine style.

3.2.2 Levels of Control

A typical agent architecture is composed by interconnected modules that are arranged in numerous relationships of grouping or subordination.

In order to deal with environmental diversities, four main control levels should compose an agent architecture. The levels are arranged from simple (bottom) to complex (top), respectively stereotyped, reactive, instinctive and deliberative behaviours [251, 252, 193]. In some circumstances, agents are made of an hybrid approach, but all of them are organised in levels to determine the course of the best action.

3.2.2.1 Stereotyped Level

The stereotyped level is characterised by instant and simple responses produced in the lowest level of the architecture (sense-act). The response is result of hard-wired "neural circuits". Behaviours like reflexes or taxies [193] emerge in the agent with no previous knowledge of the environment model (no mental states) [31]. Therefore, a real-time activity can be modelled by boolean functions.

3.2.2.2 Reactive Level

The Reactive level is set on an upper abstraction level just above the Stereotyped level, but it is still a representative of a low abstraction level. The reactive level is composed by collections of atomic rules (or modules) in the form of condition-action (stimulus-response) pairs in an ordered sequence. Modules receive sensory information from the environment or from other modules, and send signals directly to the actuators or other modules. This level allows a fast behaviour retrieval. At this level there is no symbolic reasoning (cognition) that allows agents to perform complex analysis of sensorial data. Although the previous centred form of environment model does not exist, there may be the case where the agent has knowledge that represents inner rules shaped along its evolution, the agent has (a) Believes because the term believe represents rules and rules are knowledge, (b) Desires because desires represents a set of pre-formed rules that will drive the agent towards a goal, and (c) Intentions that represents a ready cycle of training that the agent can iterate (BDI).

This unconscious level can be subdivided in two subclasses, Pure Reactive Systems and Behaviour-based Systems.

Pure Reactive Systems

The Pure Reactive systems use hard-wired circuits between perception and action. There is no state at all, no history, no information processing, nor even access to its current position. In other words, all behaviours are thus emergent once that they appear only as a result of the environment changes. For instance, an agent will only trigger an action if a raw sensorial value was recognised by the module.

This implies that the process occurs in parallel, and actions can be performed without having to wait for such symbolic complexity. The association between raw data and action is encoded in the system, such as intrinsic rules. These systems cannot improvise results when facing of unknown situations neither decide what to do next - they "decide" entirely on the present, with no reference at all to the past.

Behaviour-based Systems

The main characteristic of Behaviour-based Systems is the emergence of control by units that work in parallel. This decomposition made by activities contrasts with decomposition made by functions, and the monolithic control that are so typical of deliberative systems. The idea of decomposition shows that behaviour complexity of a system is an emergent feature that arises when there

are interactions among basic components. In general, each unit accesses sensorial data and decide in favour of adequate reactions to control the actuators.

Behaviour-based Systems have a flexible design. This heterogeneity is present in different behaviour architectures. Unfortunately, scalability is still a major limitation, and the lack of symbolic representation makes it difficult to achieve goals.

3.2.2.3 Instinctive Level

Instinctive level is the third level from bottom to top in an abstract layered architecture. Instinct is determined by an unconscious memory acquired from complex interactions with environmental stimuli [89]. The sequence of stimulus (instinctive action) runs through a chain of reactive modules. The final point of the chain may be reached using different ways when the previous stimulus was perturbed during the cycle. Instinctive behaviours are more flexible than reactive behaviours but much less complex than deliberative ones.

3.2.2.4 Deliberative Level

The Deliberative level is found at the top of abstraction layer [193, 102]. The decisions are taken via (pseudo)logic reasoning based on pattern recognition or symbolic manipulation. This class uses world representations and knowledge previously obtained to trigger an effective action. This upper level uses long-term knowledge for goals, differently from lower levels that use short-term knowledge for goals. The cognitive layer learns from past experiences, develops several combinations of actions, obtains a strategy that determines actions, and triggers them to aim a goal. For instance, in a first step, sensors receive stimuli from the environment. In the second step, a model of the environment is assembled by symbolic representation of the environment. Next, deliberation develops a plan about the modelled environment. Last, the actions are performed based on developed plan. If the goal was not achieved, new inner combination of actions will be fired again, that is, the upper level will perform several cycles until it finds the most promising answer. Unfortunately, the development of plans represent a combinatory explosion of paths to follow, and at the same time the problem increases in complexity. The idea of freezing a dynamic environment in memory to find the best answer is not useful since the logic mechanisms do not prevent invariant changes, leading to working with outdated values.

3.2.2.5 Hybrid Approaches

An Hybrid approach combines the most promising features of above approaches and balances the reasoning and action methods to react at lower level and to judge deliberative actions at an upper level. Typically, the lower levels are closer to sensors and actuators. It priors the reactive actions on the basis of organisation. Therefore, it is possible to offer fast answers to important events happening in the environment.

The lowest levels maps raw data from sensors and use them to perform the first processing. If deliberative actions are necessary, lower levels send their results to upper levels. Each upper level collects the early results from lower layers to perform different processes among them to reach the goal. Thus, a sophisticated sequence of behaviours is obtained in upper layers that control lower ones.

3.2.2.6 Organisation and Flow of Control

An agent architecture can be explained by the arrangement of independent elements interchanging data in numerous relationships of grouping or subordination. All proposed learning architectures can be briefly divided in a few main categories, such as centralised, modular, hierarchic or distributed, and one of two approaches that are bottom-up or top-down.

The cognitive agent architectures are organised in categories, as follows [177, 254]:

Centralised It is the most popular category. It deals with a well-defined problem using a rigid structure between functions and collected data. A single central unit plans the tasks and processes the collected data;

Modular This category has specialised modules that are arranged in a horizontal design of a system;

Hierarchic It is arranged by different levels of interconnected modules. Each layer has different features, and they are arranged in a hierarchic order of importance;

Distributed Each module receives a priority key to perform tasks. Each module receives results from other modules or sensory inputs, performs new processes, and sends the results to input of other modules. The whole architecture can be represented as a graph of modules interchanging information and scheduled with priorities in every moment.

During the development of architectures arranged in a vertical form, the designer defines the direction of information flow. The direction will determine how

an agent will behave in the environment. Two unidirectional routes are found in literature [28, 43]:

Bottom-up The flow of control starts from the lowest level to the upper levels where the action will be defined;

Top-down The flow of control starts from the upper level to the lower level where the action will be performed.

Some architectures use bidirectional flow of control, where the direction of the flow is from bottom-up to top-down. In case of bidirectional flow, the sensory inputs sent data to the upper layer to define the course of the best action. Consequently, the upper layer sends data back to the lowest layer to perform actions by its actuators.

1

3.3 Pure Artificial Intelligence Architectures

This section presents a set of well known pure AI agent architectures. Pure means a characteristic that does not "imitate" cognitive processes. The main goal is to provide an overview of the architecture towards our proposed architecture.

3.3.1 The SOAR Architecture

SOAR [102] is a general purpose symbolic AI architecture that integrates together basic mechanisms for traditional problem solving, learning, and perceptual-motor behaviour. SOAR is a good representative of symbolic goal-oriented behaviour, as described by Allen Newell in [168]. SOAR uses explicit production rules to govern behaviours. In accordance with other symbolic AI approaches, SOAR assumes that behaviour system must be motivated by goal-oriented states and, consequently, learning occurs in the process [130].

SOAR is a centred architecture that performs intelligent agent tasks, in the line of the main goal of Artificial Intelligence [168]. The main strengths of SOAR are driven to achieve a symbolic goal-oriented behaviour, that is, an agent should resolve any task proposed using all available knowledge. The knowledge acquired from agent sensors or data set is arranged on a single central unit that develops production rules using selection and application operators. For instance, one important type of internal goal is a goal that makes decisions about what actions to perform next [211].

The main characteristics of SOAR are:

- Breaking off;
- Knowledge integrated;
- Problem spaces represent all tasks;
- Productions provide all long-term memory (symbols): search control, operators, declarative knowledge;
- Attribute-value representation is the encoding scheme for all things;
- Preference-based procedures are used for all decisions: preference language: accept or reject, better or worse;
- Chunking of all goal-results occurs continuously.

3.3.1.1 Description of the Approach

The SOAR architecture accomplishes all tasks and sub-tasks (problem spaces) using an unique representation for the long-term knowledge (problem spaces, states, operators and so on), a representation to transient processes, an engine to generate goals (achieve sub-goals automatically), and a learning engine ("chunking").

In this architecture, every task-step is considered as a *problem* which is solved via search in the appropriate problem space, and all knowledge systems are stated in the form of production rules, commonly referred to as universal subgoaling. So, SOAR solves problems by triggering production rules, which are stored in long-term memory. When SOAR detects a most promising sequence of production rules, SOAR creates a chunk. Chuck, essentially, breaks large rules in a sequence of small ones.

Problem Spaces

The SOAR architecture describes a problem as an inconsistency between conditions. In this manner, problem solving can be described as a search for a solution through a problem space.

A problem space is a collection of different solutions of the problem (goal states) and operators. To be more specific, states of problem space represent situations, problem space operators are the synthetic actions, and motor commands are the primitive actions. There is an initial state, representing the initial situation and a set of desired states that represent the goal. An operator, when

applied to a state in the problem space, yields another state in the problem space. The goal is completely achieved when a desired state is reached as the result of a sequence of operators starting from the initial state. Thus, each goal defines a problem solving context (*context* for short) that contains, in addition to a goal, roles for a problem space, states and operators [102].

These selectable decisions influence some objects, such as objectives, problem spaces, states and operators. They ensure changes of contents in the declarative working memory (a set of objects and preferences about an object). In the problem space level, there is a variety of problem spaces that are in a task. Therefore, there are several operators that are used to solve the problems in the corresponding space inside of each problem space.

Long-term Memory

During the elaboration phase, each problem solving or decision making searches for a goal. This implies that all content of long-term memory is stored in a recognition-based memory and further encoded as *productions*. The memory will be accessed to retrieve new objects, new information about existing objects and preferences. The decisions selected to solve the problem will become the base to the learning. Productions are simple `IF...THEN...` rules that test some working memory conditions and retrieve the contents of these actions when a pattern is successfully matched. By sharing variables between conditions and actions, productions can retrieve information, that is, a function of what was matched. By using variables in actions that are not in conditions, new objects can be generated/retrieved. In this sense, a sequence of synchronous cycles (matched productions) are fired in parallel until no more rules can be fired. The productions fired during this phase do not change the working memory content, merely create references of those changes and generate effects to actuators. SOAR long-term memory is impenetrable, which means SOAR system cannot examine its own associations directly [102].

Short-term or Working Memory

SOAR supports operations of working memory contents, proving direct access to the relevant knowledge by triggering production rules to solve the problem at hands. A single knowledge access consists of a single cycle of elaboration, during which all of the successfully matched productions are fired in parallel. The single access result is the content of working memory with additional information.

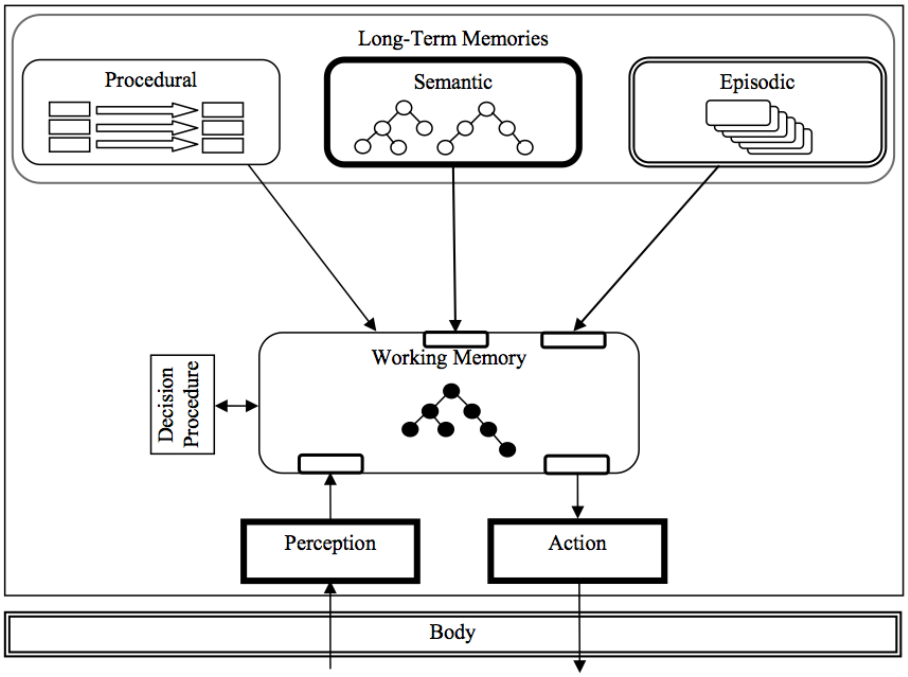

Figure 3.10: Structure of memories in SOAR as proposed in [102].

The interface is responsible for extracting sensory information from environment with the purpose of feeding the inference machine, and send to the environment the actions (perceive, sense, think and act) were selected by inference machine.

Decision Cycle and Goal-Directed Behaviour

Elaboration proceeds in a sequence of synchronous cycles, during each of which all successfully matched productions are fired in parallel. When no more productions can be fired in a cycle, a decision procedure is invoked to search for a specific object in a role in a context. The object will become the current value of the role if the preferences uniquely specify it. Additionally, impasses can occur when the available knowledge to the problem are either incomplete or inconsistent to perform the basic functions in problem solving. In this case, the system does not know how to proceed. To solve these impasses, SOAR provides a refinement of the goal.

Refinement of a goal into a set of independent sub-goals is another manner to minimise the complexity. Refinement uses a full recursive hierarchy function to automatically generate a sub-goal and associated it with problem-solving context. Thus, generation, selection, and application of a set of operators, which jointly accomplish the goal, can decompose a goal.

Impasses vary from selection problems (problem spaces, states, and operators) to problems of generation (operator application). The process of sub-goal generation is interrupted when an impasse is solved. Consequently, sub-goals offer another opportunity to learn, generating new rules in a problem space. The actions of new productions are based on the results of the sub-goal.

Perceptual-Motor Components

SOAR perceptual-motor behaviour is mediated through the state in the top context [247]. Each perceptual and motor system has its own field in the state. Perceptual systems behave by autonomously adding perceived information to their fields of the top state. Therefore, information is available for examination by productions up and further overwritten by later information arriving from the same system. Inside of the working memory, perceptual information acts just as if it were retrieved from memory. Motor systems behave autonomously executing commands that are set (by the firing productions) in their fields on the top state.

Chunking

SOAR learns by including new productions, a process called chucking [131]. Chucking is a general learning mechanism able to create new rules to avoid impasses in the future.

In his book [168], Allen Newel refers the main properties of chunking as follows:

- Converts goal-based problem solving into productions,
  - Action: based on the results of the sub-goal;
  - Conditions: based on the pre-impasse situation.
- Chunks are active processes (productions), but not declarative data;
- Chunking is a form of permanent goal-based caching;
- Chunks are generalisations implicitly ignoring whatever the problem solving ignored;
- Learning occurs during the problem solving, and chunks become effective as soon as they are created;
- Chunking applies to all impasses, hence all sub-goals,

  - Search control, operator implementation, whenever knowledge is incomplete or inconsistent.

- Learning only what the system experience,
  - Total problem solving system is part of the learning system;
  - Chunking is not intelligent *per se*, but a limited mechanism.

- A general mechanism to move up the preparation-deliberation.

All learning in SOAR uses the chunking mechanism. On the one hand, the learning is autonomous and does not intervene with other parallel activities. On the other hand, there is no simple command to be used to record a declarative track. In this kind of learning system, after an impasse is resolved, the fact previously called returns a result to a super-goal.

The learning engine has no possibility to improve itself by including knowledge. Nevertheless, the chucking mechanism ought to receive new knowledge for future resolution of problems. Particularly, the quality of future behaviours learned could be improved by acquiring knowledge that changed in sub-goals occurrences and change what was learned.

3.3.1.2 SOAR applications

SOAR has been successfully applied to many domains, such as agents for synthetic battle spaces simulation and computer games.

The TacAir-SOAR [236, 172] represents a generic automated pilot agent for battle spaces simulation environment. The pilot agent was planned using the SOAR integrated architecture [210, 209]. The pilot is an automated agent specialised by means of parameters and domain knowledge. The automated pilot includes a variety of important capabilities: goal-driven and knowledge, intensive behaviour, reactivity, real-time performance and so on. TacAir-SOAR pilots have already successfully participated in constrained air-combat simulations against expert human pilots[2]. Nonetheless, TacAir-SOAR is the first AI system to have participated directly in an operational military exercise [209].

SOAR is being also devoted to computer games research. For example, SOAR-based Quake bot [240] has an Artificial Intelligence Engine around the SOAR Artificial Intelligence architecture [126] that attempts to incorporate some of the missing human players capabilities, such as goal-oriented and multi-step look-ahead techniques. This bot is distinguished by its ability to build its own map

[2]It was provided by ModSAF [34], a distributed simulator that has been developed commercially for military purposes.

to use a wide variety of tactics based on its internal map, and in some cases, to anticipate enemy's actions. The bot uses a dynamic hierarchic task decomposition to organise knowledge and actions. It also uses internal predictions based on its own tactics to anticipate its opponents actions [127, 128].

3.3.1.3 Conclusions

The SOAR architecture attempts to encode the general human intelligence in small sets of basic decision-making mechanisms and representations. As a top-down approach, the architecture can perceive environment changes, learn the facts, remember similar situations, and decide the best agent goal by logical reasoning.

SOAR system was designed to extract implicit information from simple situations, and to reason about goals. SOAR requires either representing some goals implicitly or forcing unrelated goals into a single hierarchy [107, 88]. In addition, sub-symbolic I/O is not supported.

A problem that affects the SOAR learning process is the use of a single goal hierarchy. This approach produces over generalisation of chunks, and makes them expensive chunks. Effectively, the hierarchy performs few rules in fast simulations. Conversely, some catastrophic collapses can occur in long simulations, where a large set of good rules may be lost. Symbolic models are limited to maintain long simulations updated. Fortunately these lost rules are acquired again, but the price is instability of the hierarchy defaults. It is one of the most serious problems found in SOAR. Furthermore, as all symbolic systems, the representation of an object can scale-up as the size of the knowledge base increases. Other negative point is that small decision rules represent a huge effort to SOAR.

Although SOAR has a high-level inference machine that constantly operates in a decision cycle (perceive, think and act), there are some problems to this theoretical approach. Sometimes "thinking", is not needed but reactivity would be the best answer. For instance, one problem could determine how to allocate attention to features, depending on the task. Complex environments involve a very large number of features, and some allocation of attention is required to focus on the critical or most diagnostic features. The allocation of attention needs to be learned from experience for each type of inference task, and current SOAR exemplar models have failed to provide such a learning mechanism. One last problem is that they fail to account for sequential effects that occur during training. This failure results in systematic deviations.

3.4 Cognitive-based Artificial Intelligence Architectures

In a cognitive process, a decision-making route of beliefs or desires is fired to produce actions for emerging behaviours in agents [251]. A cognitive strategy takes advantage of known facts to improve a strategic plan to accurately achieve its goals.

We now describe the well known cognitive agent architectures, showing their most promising features and processes to build cognitive agents.

3.4.1 Subsumption Architecture

A Subsumption Architecture (SA) [28] is a hierarchic structure of distributed task-accomplishing behaviours used to control agents. The conception of the architecture was inspired in decentralised and paralleled reactive structures of primitive nervous systems. In his paper entitled *Elephants don't play chess*, Brooks [30] presents interesting ideas about the emergence of complex behaviours without learning in the process.

The main characteristic of a SA is the rational behaviour without explicit knowledge representation. Differently from traditional symbolic systems, Brooks argues rationality emerges from the result of interaction between reactivity and environment [31], and not from symbolic representations and their features (world model, and a shared global memory), nor logic reasoning techniques so commonly used by the centred approach. His technique avoids update of short-term or long-term memories because the response to stimuli is purely reflexive. Reflexive responses provide real-time actions between sensors and actuators in complex, dynamic and unpredictable environmental situations. As Brooks says, "*The world is its own best model*" [31].

3.4.1.1 Description of the Architecture

SA remodel the orthodox basis of the horizontal deliberative scheme, from the sequence sense-model-plan-act, Figure 3.11, into a vertical parallel ed model with layers of control, Figure 3.12.

In a SA, each layer only uses sensory information necessary to command the actuators, and it is filled in with a behaviour module that processes information coming from sensors to perform specific tasks. A behaviour module is a Finite Automata Augmented with timers (FAAt) [28]. Timers enable state changes after pre-programmed periods of time. Each independent and specialised FAAt is intended for a specific task (avoid-obstacle, go-home, follow-light, ...), and FAAt can be defined by several levels of abstraction (go-one-step-ahead, go-ahead, ...).

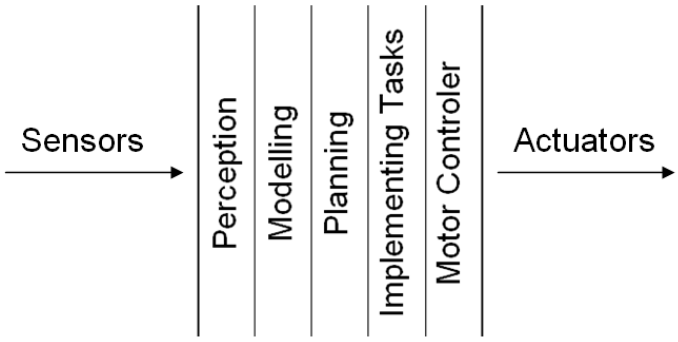

Figure 3.11: Traditional sequence of sense-model-plan-act as defined by [28].

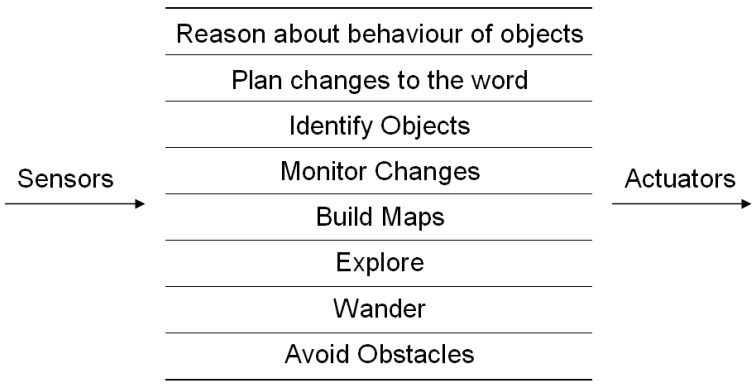

Figure 3.12: An approach based on task-achieving behaviours as defined by [28].

Each input and output of FAAt can inhibit other FAAt of low priority, in the same form that stimulus-response can be inhibited by other active FAAt. FAAt is manually implemented as follow: input and output signals; a limit of states; one or two internal registers; one or two internal clocks; and a simple calculus machine, such as vector sum. These behaviour modules are fired by input signals load from agent sensors and/or from other behaviours, and sent to the agent actuators or to other behaviour modules (stimulus-response).

The organisation of layers is based on a fixed behaviour-based priority scheme from bottom to top. Each layer operates asynchronously and always pays attention to its sensory readings and acts accordingly. In the hierarchic scheme of behaviour modules, the lowest layer performs actions that are not seen by layers above it. Consequently, upper layers can subsume lower layers by its own commands, deciding which behaviour must be triggered in each moment.

Agents controlled by SA are fully dependent of their sensors to decide the best course-of-action [233]. The stimulus-response technique interacts physically with their surroundings. Brooks argues that the best way to produce "intelligence" and

adaptive behaviours is from the combination of simpler, underlying behaviours. Thus, agents can perform their tasks autonomously based on reactive features. For instance, an agent can go from point A to point B without any knowledge concerning path planning by performing simple tasks. Figure 3.13 shows a robust architecture that subsumes commands from layers directly underneath when they wish to take control of an autonomous agent. In order to control and avoid conflicts, the architecture subsumes commands from layers directly underneath when they wish to take control.

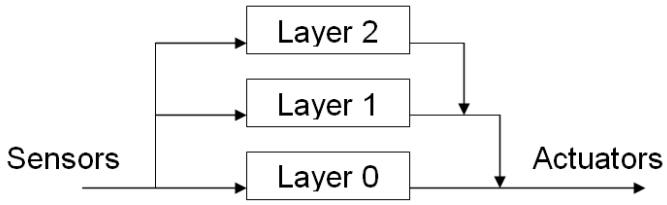

Figure 3.13: Brooks's Subsumption Architecture as defined by [28].

This example of hierarchic multi-layered structure is filled by specialised FAAt's that minimise the scalability problem typical of a central unit architecture, consequently reducing one of the biggest problems in implementation of behaviour tasks. Fault tolerance is also another robustness of Brooks's architecture because if any layer breaks down, this fault will not collapse the entire structure.

The mechanism used to resolve conflicts between competing or conflicting behaviours between layers, in its fixed topological network of simple FAAt, is the Hormone Activation System (HAS), Figure 3.14. HAS controls the behaviour by inhibiting or suppressing signals. This gives each level its own "rank of control". It is designed to modulate the agent behaviour by thresholding the layers of FAAt's and preventing the activity of those levels below the threshold.

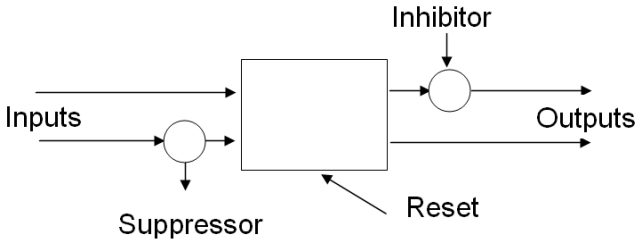

Figure 3.14: The Hormone Activation System.

The most important aspects of this hormonal mechanism are the adequate integration of many different modules made of low-level behaviours, and when higher-level behaviours dictate orders (suppressing) whenever a low-level response

is not needed. Therefore, FAAt's can be inhibited by the presence or absence of a hormone, allowing for higher levels to subsume the function of lower levels. In [31], Brooks refers that the activation system was inspired by the animal hormone systems. However, the abusive use of HAS can increase the size of the system linearly, endangering all structure [254], which is one of the unsolved problems in SA.

3.4.1.2 Deployment

The SA approach contributed for the development of the third generation of robots, called "intelligent robots". Based on SA, Brooks developed a six-legged robot called Genghis that provides a useful metaphor for understanding the functional architecture of insect nervous systems [29].

Allen was another "Brooksian" agent. The Allen architecture [28] is one of the most well known, and it is shown in Figure 3.15. In layer 0, the agent avoids stationary as well as dynamic obstacles that may appear in the environment. In layer 1, the agent randomly wanders around aimlessly without hitting obstacles. In layer 2, the agent explores the environment with its own sonars.

In layer 0, obstacles are avoided. In layer 1, the robot controls the progress and sends updated commands to the actuators. It is interesting to note that there is no conscious memory about the obstacles that were avoided. In layer 2, the robot visits places, which were not defined by Brooks. Each layer has a fixed scheme of finite state machines combined through suppressor and inhibitor mechanisms. These mechanisms are activated by messages or world state changes.

Other experiments like Attila, Herbert, Tom and Jerry, Seymour, COG and ATLANTIS have used Brooks's approach as described in [30].

Many additional extensions have been implemented in Brooks's architecture. Most of them were made by Maes [140], Mataric [143, 144, 145], and Firby and Slack [75]. The main point was to improve new capabilities of behaviours in reactive systems. The capabilities developed were [144, 146, 79]: object detection and map building, planning and learning to walk, collective behaviours with homogeneous agents, group learning with homogeneous agents, and heterogeneous agents. Latter on these extensions would be known as behaviour-based capabilities.

3.4.1.3 Conclusions

Simplicity and real-time performance are two features adequate to control agents in static, dynamic and unknown environments as argued by [233]. Thus, Brooks's architecture characteristics focus on reactive and modular organisation. The monolithic decision making controller was divided in layers with a priority scheme

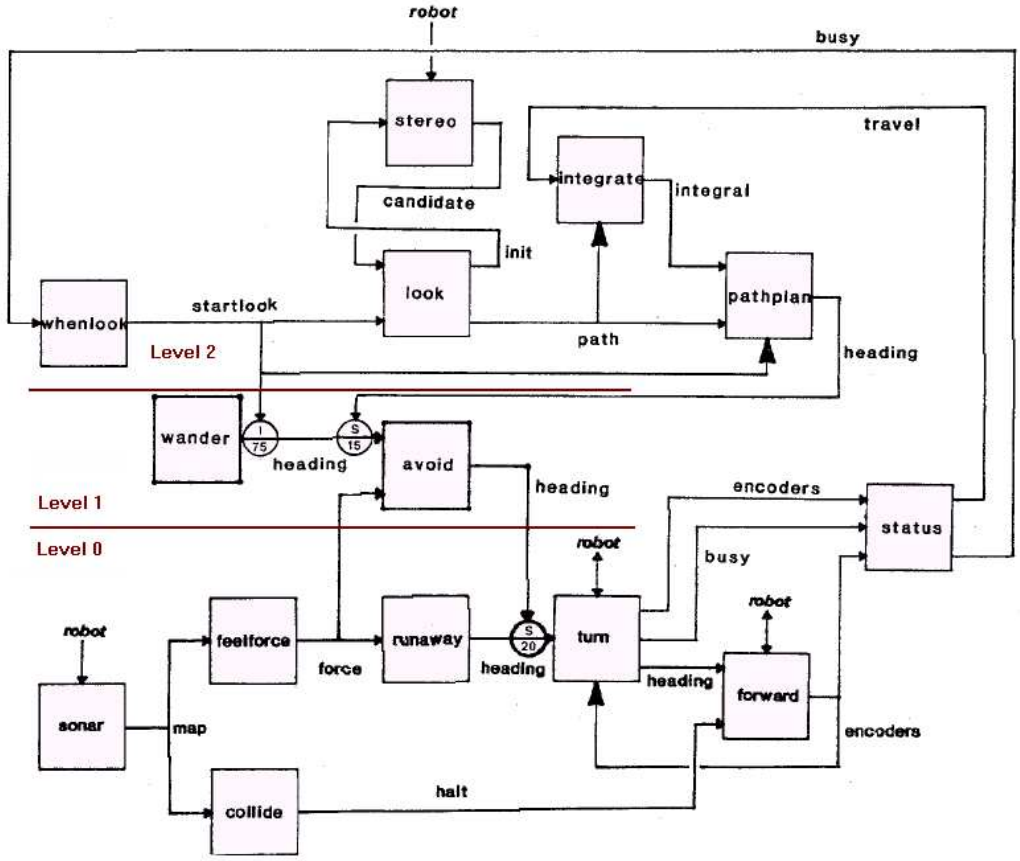

Figure 3.15: Allen Architecture. Reproduced of [28].

to achieve complex behaviours, that is, higher priority behaviour subsume the output of behaviours implemented beneath.

The absence of an inner world-model does not prevent the agent navigation. On the contrary, a fast navigation is one of the best features obtained by subsumption agents. Unfortunately, it is well known that finite state machines need to be implemented manually one by one. So there is a strong need for more and more modules, which could become a bottleneck in the proposed architecture. In addition, behaviours are fired by internal or external conditions and maintaining all modules is necessary. Additionally, the inflexible architecture proposed in SA prevents the progress of system migration to another agent in other environment. Consequently, the SA architecture does not offer the expected adaptability.

The Subsumption Architecture does not use cognitive operations involving representations, neither performs complex analysis of its sensory data or adapts its behaviours to become prepared to respond, unless the agent had sufficiently knowledge to perform such tasks. Conversely, *beliefs*, *desires* and *intentions* can be achieved, as was explained before.

Brooks also claims that rationality should not be seen as a computational processing, where a input produces a processed output. Rationality, claims Brooks, is so complex and simple at the same time as the interaction between agent sensors and the world. On the one hand, sensory input devices are rich enough for them to uniquely decide the next best action, without resorting to an internal model of the world state. On the other hand, agents will depend entirely of the world state to obtain the next "decision". It is well known that sometimes the current world state is not sufficient to provide a "decision" about what to do next.

Another problem occurs when enforcing sensorial information to be processed in separate channels. Separate channels reduce the possibility of unifying redundant evidences that could be used to estimate behaviours more accurately. A benefit could be to allow the interchange of sensory data among modules. As a result, the architecture would not have an overload of a full environment state. One last problem is associated with fixed behaviour-based priority scheme combined with movements of avoidance obstacles by a military vehicle, as cited in [176]. The SA solved the problem linking the output of the lower level with the entrance of the upper level. Unfortunately, this successful solution compromises the parallelism of the SA and its benefits. Because a parallel architecture cannot enforce upper layers to wait for the output of lower layers.

3.4.2 PyramidNet Architecture

The PyramidNet Architecture [204] was proposed by Mauro Roisenberg, as a modular and hierarchic approach composed by Artificial Neural Networks (ANN). The architecture was both inspired on Brooks's Subsumption Architecture [28] and hierarchic nervous structure of some animals. This connectionist architecture, in fact, was induced from modules both arranged in vertical and horizontal levels. These levels allow an increasingly complex behaviour structure [193].

Using the connectionist approach to be closer to biologic plausibility, Roisenberg's research has been focused on the simulation of many survival biologic behaviours [204, 206]. Unfortunately, the scalability problem arises when a large variety of behaviours need to be emerged from a unique ANN. A solution was the modularisation and hierarchy (distributed computing) proposed by Brooks [28] to solve the scalability problem and achieve expected agent responses.

The main feature of the PyramidNet Architecture is its robustness and flexibility to support different behaviour modules arranged in an hierarchic form. The information flow follows the bottom-up learning, that is, from reactive to deliberative behaviour in order to generate agents decisions. The advantages of using modules and layers arranged in an hierarchy is that each module has a specialised function, a minimisation of mutual interference and execution among simultaneous process. The use of ANN can represent many advantages because

it supports high noise immunity, fault tolerance and programming by examples [185].

3.4.2.1 Description of the Architecture

Inspired also on the modularity of human brain, and based on Brooks's approach [28], the PyramidNet arranges the modules in a pyramidal form to explore the vertical and horizontal approaches. In the horizontal approach, homogeneous ANN modules populate each level. Consequently, the parallelism of layers, in the vertical approach, is used simultaneously to trigger heterogeneous behaviours.

These mixing of approaches allows different tasks to be performed at the same time. These levels of function represent subsequent clusters of ANN and remind us of the book *The Society of Mind* written by Minsky [155].

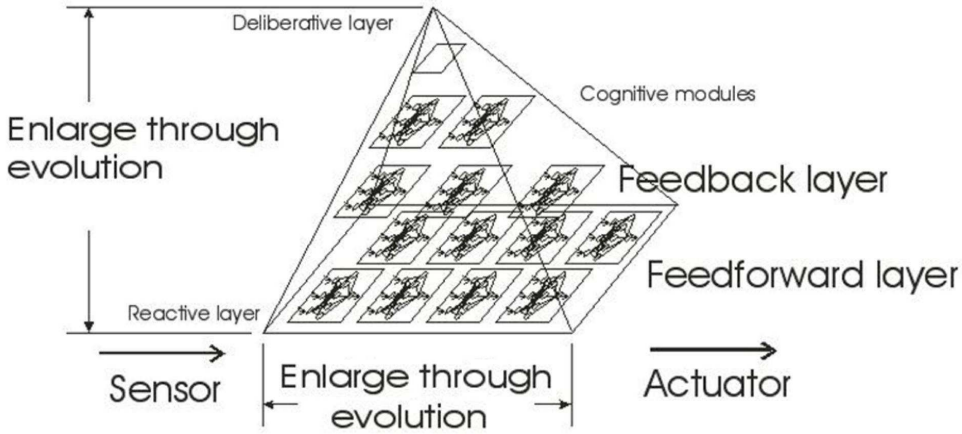

Figure 3.16: PyramidNet architecture as specified in [185].

In the base of the pyramid, at the effector level, plain or reflexive behaviour modules can be found exploring the straightforward performance. Static ANN, such as Feedforward ANN, model the behaviours. Additionally, complex behaviours are modelled by recurrent neural networks that are set on the top of the pyramid [204, 185], Figure 3.16.

The PyramidNet approach arranges levels in a bottom-up fashion, and divides the levels of complexity into incremental functionality to support adequately a large variety of behaviours without decreasing the performance. The idea focus on many cases where ANN modules are used to perform processes in different periods or a same process will be performed for many modules "in the same" time in distinct and functional areas of the brain [194].

The communication between levels and modules is implemented using multiple paths of internal connections. The vertical approach can be compared to

a brain when executing different behaviours simultaneously, where an intrinsic parallelism occurs with different sorts of information. The process course happens in a serial way, that is, the functions of different hierarchical modules are accomplished in a same functional layer and the processed information can be reviewed to the posterior hierarchic layers [196]. Thus, the hierarchic structure is flexible enough to be extended with more behaviour or levels in accordance with complexity of a problem to be solved.

3.4.2.2 Experiments

The experiments that follow were developed using the PyramidNet tool [185]. This tool has been successfully used to plan examples of PyramidNet architecture with integration of various independent ANN subsystems, providing a robust "nervous system" [192].

In order to assess the robustness of the architecture and feasibility of the tool, two different experiments were carried out, as follow:

- "Container Capturer": emergence of behaviours in a dynamic environment;
- "Follow Wall - Search Recharging Point": connecting heterogeneous learning modules, and emergence of complex behaviours.

First Experiment: "Container Capturer"

In order to show the emergence of autonomous behaviours in an agent, PyramidNet architecture was used as the backdrop support. The main intention of the agent was to identify and collect specific objects (container) in a dynamic environment. The robot Lego MindStorm [135] was used as the agent body, and the environment was carefully prepared to be unpredictable as possible. We refer the reader to [196] for more detailed information.

Figure 3.17 shows a structure composed by two layers. The lower layer performs basic tasks, such as backward movements, looking around and forward movements. The upper layer performs complex decisions about continuously searching for containers, retreating for wrong containers, and pushing wrong containers out of arena.

Figure 3.18 presents the architecture proposed. The learning module, Stereotyped Network, uses FeedForward ANN topology trained with the Backpropagation algorithm to achieve simple behaviours. This module receives signals from sensors and sends data to the Reasoning Net module (at the second layer). The Reasoning Net module, uses Recurrent ANN to decide about detected events in

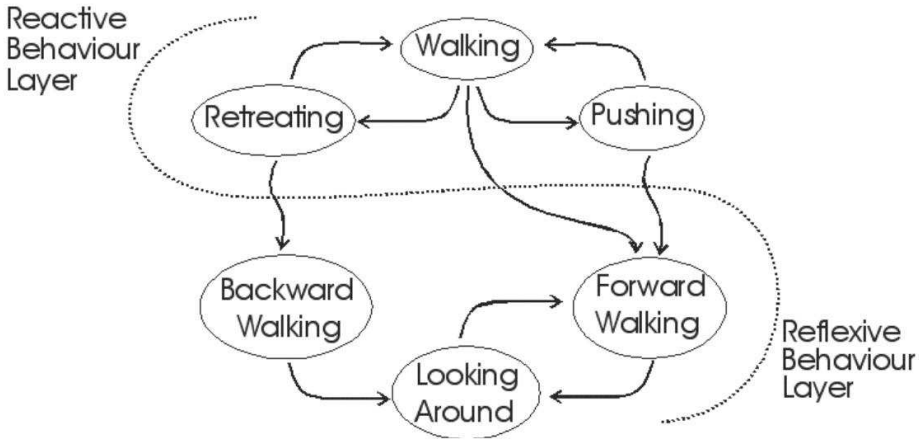

Figure 3.17: A global view (sketch) of Behaviour Task Plan.

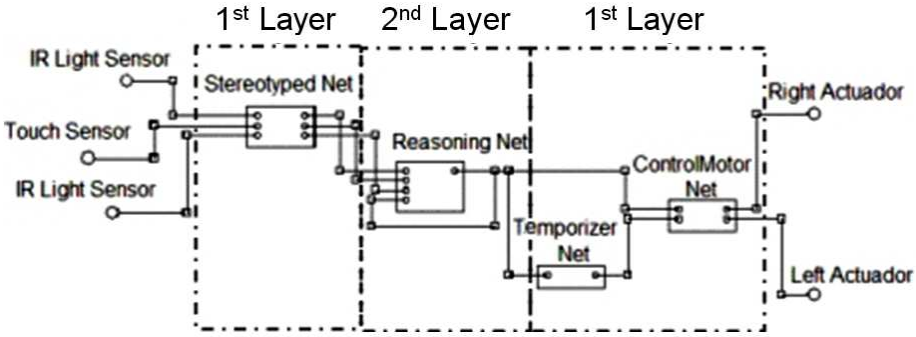

Figure 3.18: The diagram of Behaviour Task Plan - designed in PyramidNet tool.

the first layer and transmits them to first layer and overlapping it. The Temporiser Net module, uses FeedForward ANN topology trained with the Backpropagation algorithm to solve conflicting decisions. Finally, the Control Motor Net module, uses FeedForward ANN topology trained with the Backpropagation algorithm to control the tracking motors from superior layer.

Second Experiment: "Follow Wall - Search Recharging Point"

In the second experiment, a Khephera robot was used as the agent to test the theory about emergence of complex behaviours. The environment was carefully prepared to explore all behavioural characteristics of the architecture. In this project, the Lego agent wanders in an environment until the battery has no energy. The agent objective is to find a recharging point and wander around. For more details concerning the project we refer the reader to [196].

The architecture shown in Figure 3.19 is composed by eight sensors (from A

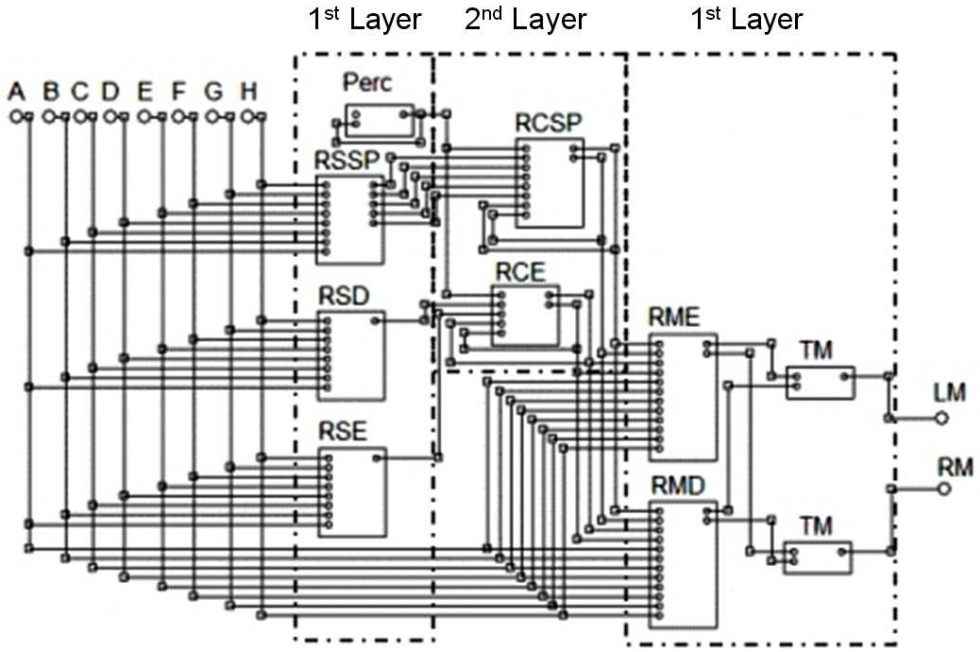

Figure 3.19: The "Follow Wall" and "Search Recharging Point" diagram.

to H), a Perceptron (Perc), two actuators (LM, RM), and nine Artificial Neural Networks: Sensory Follow Wall (RSSP); Sensory Distance (RSD); Sensory Energy (RSE); Control Follow Wall (RCSP); Control Energy (RCE); Walk Energy (RME); Walk Distance (RMD) and; Motor Controllers (TM).

The Khepera default behaviour modules, presented in Figure 3.19, are: wander in the environment, measure the wall distance, detect obstacles, and detect light direction. Two recurrent ANN receive their inputs from these previous Feed-Forward ANN and from the recurrent Perceptron that simulates the low battery.

3.4.2.3 Conclusions

PyramidNet architecture uses a connectionist approach to emerge complex behaviours in agents. In order to develop a robust architecture and minimise the scalability problem, interconnected modules compose the structure of pyramid. Based on a parallel and hierarchic distribution of behaviour modules, PyramidNet architecture follows the same modular principle. Therefore, PyramidNet architecture is composed by multiple layers, each layer with a function. These layers of function represent subsequent clusters of modules that are arranged in a hierarchic way, allowing emergence of more behaviours [193]. This biologic plausibility can be compared with a nervous system. In fact, our nervous system has

a hierarchic structure. For example, control of human skin temperature does not depend on the central body control [121].

Artificial Neural Networks usually achieve good generalisation, that is, they respond correctly to inputs they have not seen. Unfortunately, there are some disadvantages, such as the absence of a methodology to build agents, the difficulty to develop an automatic learning technique, and the need of a large quantity of environmental information in order to receive good and tuned responses. Additionally, an architecture only made of ANN can offer imprecise general answers when logic is required. For instance, an exact output of a mathematic expression is difficult to obtain using ANN. Other point is although ANN were deeply focused in brain mechanisms and biologic nervous system, PyramidNet architecture does not explain how the mind works, but makes some speculations.

3.4.3 Minsky's Approach and "The Society of Mind"

Minsky's approach addresses the modelling of how mind works and he suggests that intelligence was emerged from non-intelligence. In his book called *The Society of Mind* [155], Minsky did not follow the "basic AI principle" from which all cognitive phenomena in some way was emerged. Instead, Minsky suggests that the construction of mind was only possible from many small-specialised cognitive processes or agents working together.

The strength of Minsky's approach is on taking the perspective of the mind as a society of heterogeneous agents representing different processes. Consequently, the emergence of "true" intelligence happens by real formulation of mind as a society of agents without full knowledge of each other.

Agents are small pieces of specialised mindless entities of machinery with more autonomy than traditional modules or procedures. They do not need of higher level nor a structured hierarchy to cooperate with. The main idea is that agents interact with each other to build a large system called *Society of Agents* (see Figure 3.20).

According to Minsky, thinking and other mind abilities emerge from interaction among these agents. This heterogeneity establishes agents with their own distinct objectives, such as language to describe things, ways to represent the knowledge, and methods to produce results with efficient solutions.

Minsky modelled the human cognition as a complex system composed by several societies partially autonomous called (partial) mental states. For instance, suppose that society of mind performs tasks like an administrative organisation. In the highest level of this organisation, general divisions are established, such as sensory processing, long-term planning and so on. Inside of each division, there are sub-specialised agents, made of small elements of specific knowledge

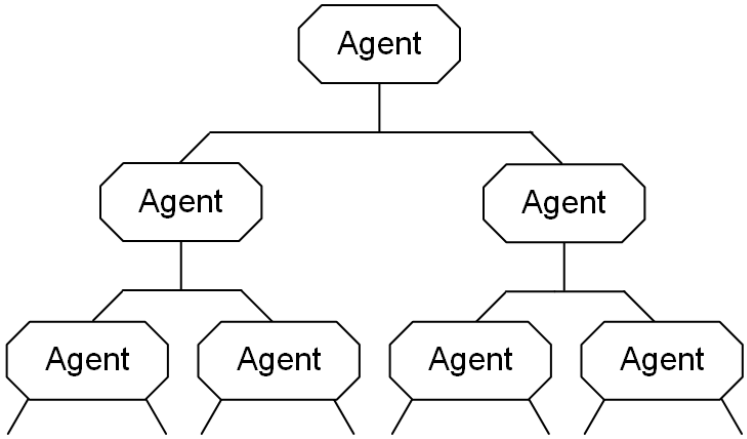

Figure 3.20: A society of interconnected agents according to Minsky [155].

and methods. Each agent knows nothing about itself, except that it recognises communication connection sets, and answers if its states are changed. The size of the society can be enlarged or reduced for each mental activity according to the need for more or less agents.

The variation of mental activity can result in hierarchies and bureaucracies, specialisation or other social arrangements. Generally, heterogeneous societies are more complex to control, but are robust to support a huge set of tasks. For instance, an analogy can be made with human immunological system, where each agent of immunological system is in a specific society. A large quantity of agents when combined to activate or not other immunity agents when needed, they can produce an amazing immunological system.

3.4.3.1 Description of the Approach

Minsky's approach addresses the highly evolved and very complicated human mind with different types of basic processing units working together as an agency. These basic processing units are now described.

K-lines or knowledge-lines perform a search in the past for similar solutions, when confronted with problems of a similar nature. A K-line agent has the potential of finding solutions to problems, which have a variable input, and which in the non-AI approach demand a variety of algorithms (depending on the input) in order to be solved efficiently. K-line agents connect with other K-line agents by connection lines. They are activated to assemble in cascade small information units. Eventually, K-line agents build their own societies to emerge specific abilities inside the mind.

Neme and Nome are two instances of K-line, respectively data and control line. *Neme* describes an agent output that represents a fragment of an idea or mental state. This example of agent invokes representations of how the environment and the learning coming from experience could be represented. *Nome* is other example of agent that affects an agency by its outputs, looking after the control of how the information is represented, processed and worked.

Polyneme and Microneme are two subtypes of Neme. *Polyneme* arises incomplete states within multiple agencies, where each agency is involved with representing some different aspect of a thing. For example, recognising a chair arouses a "chair-polyneme" that invokes certain properties within the colour, shape and other agencies to assemble mentally the experience of a chair, as well as it brings to mind other fewer sensory aspects such as the cost of a chair, places where chairs can be found, the kind of situations in which one might use a chair, and so on. Polynemes support the idea that a distributed way across multiple representations is the best alternative to express meaning of things. In the same line, *Micronemes* provide contextual signals of global connection for all agencies. It also describes dedicated aspects that are hard to be explained by words, like concepts partially determined, such as specific smell, colours, shapes and intuitions; aspects of current moment that are difficult to be linked to any particular object or event.

Nome has been divided in three subtypes: Isonome, Pronomes, and Paranome. *Isonomes* interchange messages among different agencies in order to perform the same kind of pattern achieved by a cognitive task. For instance, they can require a set of agencies to save their current states to short-term memory in other to be read in a different situation, or then require it to begin the training to a new long-term K-line memory to copy the current state, or ever require them to imagine/visualise consequences of a certain action. *Pronomes* are Isonomes that control the use of representation of short-term memory. A Pronome is frequently associated with a specific function in a large situation or event, such as a localisation of an event previously occurred. Pronomes can link to specific kinds of agents that are in short-term memory allowing to record only specific kind of knowledge, such as place, shape or path. Other Pronomes can be used as general purpose and achieve the majority of agencies in the brain. Minsky calls them "IT" because the big quantity of connections are required to be modelled. *Paranomes* coordinate the use of multiple representations. For instance, a Paranome could be connected to Pronomes that would be connected to two other different representations, one in terms of an egocentric or centred body that coordinates the system and other in terms of an external or third person that coordinates the system.

An active agent K-Line records the current activities. If the system suddenly receives a similar problem to be solved, this society previously formed will have

a previous solution and a starting point to solve it. This recompilation process of agents activated in that moment in mind, the experience acquired from the solution had included: memories of false initial points, sudden discovery, sorts of strategies to solution of problems, sorts of knowledge, sorts of subjects, memories of particular experiences and other links that had previous origin from past experience and that can help the system solve a problem.

Frame is other important element in the society of agents. *Frame* represents a package of information that helps agent recognise or understand something. A package of information can be built from Pronomes that control the connections to the port. When a frame is called, Pronomes start their relations to call partial description of aspects of something that is being described. Thus, the frame represents situations and discovers pathways to typical problems.

Memory is a network of Frames interconnected with common Pronomes. Each Frame is linked to each known concept. Each perception selects a frame and classifies the current situation into a category, in which it will be adapted to that situation. Frames offer computational advantages because they focus on the rationalisation of information in some situations. They are biologically plausible because they do not divide cognitive phenomena - phenomena like perception, recognising, rationalisation, understanding and memory.

3.4.3.2 Problem Solving

In order to solve some problems in a society of agents, problem-solving agencies ought to be organised at every level. A difference-engine is a goal-driven problem solving method. In the difference-engine, agents work together to minimise the difference between current and desired states of affairs as a typical means-end-analysis approach. The agents try to improve by invoking K-lines that turn on satisfactory solution methods. Inhibitors and suppressors are also used to attenuate the competition among agents [223]. Minsky explains in his book that inhibitors represent mental activity that precedes unproductive or dangerous actions, and suppressors suppress those unproductive or dangerous actions themselves [156].

3.4.3.3 Communication

An agency is made up of several interconnected agents interchanging signals among them by a communication line to represent knowledge. In order to understand the signals, agents shall agree well enough on the meanings of these signals, that could be “words” or other representational-construction operations [223]. In this context, agents can activate a Polyneme to arouse agents that “think” about some particular things (object, event, or situation), or it may activate Micronemes that trigger other agents to “think” about some general context.

3.4.3.4 Experiment

Minsky's approach is a highly speculative theory about how mind works, and how we can build a similar mind with a machine. In his book titled "The Society of Mind", Minsky documented all the process to decompose the human mind into a society of independent but simpler agents organised in levels.

Despite the great popularity of his book, there have been few attempts to implement the proposed theory. The main problem is the amount of fragments and many conceptual levels at his theory. While some tasks sounds simple to be modelled, in practice it is too complicated.

In all of the papers presented in his book, Minsky does not offer a detailed specification of the theory, but rather takes a higher-level perspective. Moreover, the "micro word" idea has been proven to be extremely scaling-up in practice. It has been written (programmed) by so many students, and the system was so large that no one could follow it because of its complexity, and it was abandoned in 1971.

It is well noted that the notion of decomposing complex processes into organised subsets of simpler pieces of semi-autonomous software were not really introduced by Minsky, as well as the notion of organisation of elements in a hierarchical form, and the consequent organisation that impose on the control of the system. Both of these ideas have a long and distinguished history in cognitive science, and can be found in [152, 222].

3.4.3.5 Conclusions

The Minsky's theory presents us the opportunity to understand the mechanisms of mind. In his approach, agents perform their tasks accurately, but they perform their conscience less tasks. Minsky proposes that a large-scale cognitive architecture emerges intelligent actions from a diversity of non-intelligent agents arranged in an organisational bureaucracy, and behaviour imposes coherence to experience.

The biologic plausibility that Minsky claims is associated with millions of cells in the human brain. Each one complicated by itself, and arranged in a massive and connected network. In general, each neurone represents a simple processing unit (or a sub-specialised agent) that receives signals from other neurones. Depending on such conditions, agents transmit signals to a set of other neurones in neighbourhood. Consequently, the complexity of human cognition emerges from these network of interconnected neurones.

Minsky claims that intelligent machines can be implemented if we are able to mimic the nanoengineering of mind. Thus, Minsky assumes that every cell

of a human brain could be changed by a simple processing unit with purpose of performing the same functions and linked to other.

In conclusion, Minsky answers that anyone can build a mind since there are small elements, each one independent and unconscious. It is our opinion that Minsky's theory is a good starting point for a materialist treatment of the problem of consciousness, but many more new ideas will be needed.

One interesting point to note in this theory is the similarity with Simon's book [221] and Braitenberg's book [26]. Both of these books advocate that simple systems on the presence of other simple systems, when set in a complex environment, they can produce complex, fascinating, and extraordinary phenomena.

3.5 Multi-Agent Systems

Distributed, open, and large-scale organisations, modelled as systems composed of many agents, have received attention in research community. These organisations involve various open challenges of monitoring geographically distributed and interdependently built multiple agents.

Multi-Agent Systems (MAS) is a powerful research area that investigates complex social phenomena. MAS is based on idea of a set of autonomous agents goal-oriented interacting with each other by communication protocols, and in a dynamic environment. Therefore, interactions can also result in many behaviours. Agents use events to interact with each other directly or using the environment [68]. This social interaction is the main factor to emerge intelligence in the complex systems [173].

In the MAS area, the major research interest is coordination of behaviours in agents. A MAS coordination offers guarantees that the agent community will act coherently, that is, must be present when agents are dependent with each other to achieve a global goal. If there is no dependency, agents can share information among other agents that can be useful at that time.

This thesis is not driven to study of behaviour in a community. On the contrary, the main goal of this work is limited to the development of a robust agent architecture without social behaviour but capable of performing autonomous tasks. We focus on the emergence of intelligent behaviours by interchange of signals among homogeneous or heterogeneous algorithms. We believe this kind of architecture supports a general and promising representation of knowledge from different behaviour modules. We also believe the architecture will provide answers to the questions of how sensory input and current agent state can determine future actions based on its internal states.

3.6 Meta-Architecture

A major drawback of traditional autonomous agent architectures rely on two main factors. First, monolithic cognitive agent structures that cannot be fine tuned to interact in a new environment by the designer nor by itself. Second, new functionalities cannot be extended because it is hard to be understood. In order to emerge dynamic and self-adjustable behaviours, agents should have a flexible structure made of modules in which the agent should decide (balance) its own performance by activating or not some parts of its own structure easily.

The architecture should support inner modifiable behaviours. Thus, agents should control its own internal states whether new or different runtime activities are required.

To observe and tune its own (autonomous) behaviour, agents use what we call a "self-modelling" feature, that is, reasoning about and acting upon itself. The modules are planned in design phase, but pragmatically, some will arise later in other strategy-level decision, which have their place in the meta-architecture.

Meta-architecture in software engineering is a new paradigm, which drives designers to reason about easily maintainable systems. Meta-architecture separates functional from non-functional code. Functional code is concerned with computations about the application domain (base level). Conversely, non-functional code resides at the meta-level, supervising the execution of the functional code.

Meta-architecture is arranged in the highest level of the structure. It provides the guidance to the system structure, rather than the higher level of abstraction (conceptual architecture) or detailed one (logical architecture).

The meta-architecture of an architecture is itself an architecture before its conception, which does not prototype a concluding application but an architecture [225]. Self-modelling of a meta-architecture uses architecture to define another architecture, and for that reason, meta-architecture provides the basic modelling of components that describe a particular system, called architecture. To Artificial Intelligence, a meta-architecture level offers a new way to produce agent architectures with only one architecture, where a structure of an agent can be migrated into other.

The main purpose of meta-architecture is to guide designers in advance on planning of a system structure. Decisions about components were made by designers during system structure design, and it can be viewed as a key part of meta-architecture.

Typically, a meta-architecture creates a small number of system concepts that are effective, that is, the meta-architecture has built-in a collection of high-level decisions (strategies) working together. The high level establishes open system

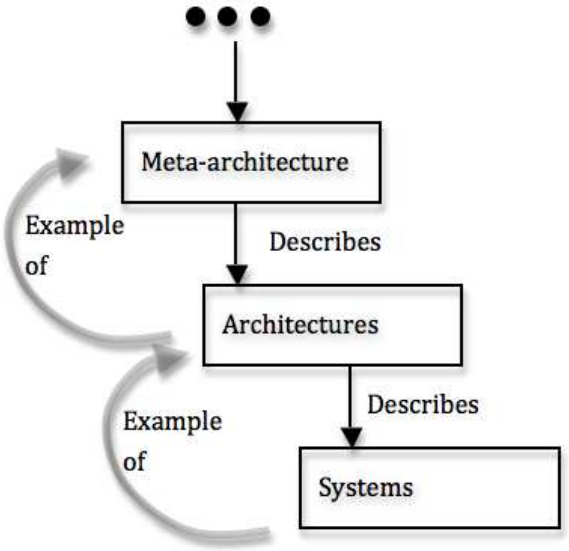

Figure 3.21: A Meta-architecture model of an architecture is itself an architecture.

architectures. The set of system concepts will shape the architecture, and strongly influence architects to deliberate about the structure of the system.

Deliberation is the key concern for an architecture strategy. Architecture strategy supports the operational strategy, and indicates how the technical strategy will be implemented. In order to set the right decisions, the preliminary decisions about the system, and in particular the definition of system scope should be taken.

Designers are who understand the system context and establish goals that the architecture will support. They also design the system, encoding an operation strategy into a technical strategy (meta-architecture), and leading the implementation of the technical strategy. The compilation of the architecture encodes deliberative reasoning into mechanisms of response more efficiently. The operation strategy establishes what capabilities are necessary to build or improve the high level objectives. The architecture objectives to establish how technology will be used to deliver these operation capabilities, setting direction for the architects and development community.

3.7 Conclusions

Before choosing the architectural style of any agent, designers have to have in mind a careful analysis of the problem and environment. Designers should have in mind that the development of agent architecture is beyond simple arranging of

control blocks, implementing of functions or operational issues, like programs. In order to construct true agents, designers should know about interaction among parts, composition of subsystems, declarative issues, system-level performance, context analysis in which agent is inserted, and what strategies are needed to achieve the goals. As we know, the true essence of the experiment is to evoke the problem and further analyse the variations of response by changes that were triggered by stimuli. Consequently, stimuli and responses are aspects directly connected to behaviour, and learning can or can not be part of this process. Stimuli and responses build a structure to study the intermediate "thinking" process in a legible and interpretable form. For instance, Pavlov's theory has used the notion of induced cortical excitation to explain the formation of neurones connections. His theory relates learning with responses to external stimulus. Conversely, production rules are another form to study the intermediate process, and should not be discarded. The interpretation of an stimulus-response is more comprehensible if the process is arranged in an architecture interacting between agent and the environment. In opposition to the traditional cognitive school, connectionist architecture can also be used as base to support heterogeneous inner processes, and achieve an adequate high behaviour level.

Architectural styles are patterns that define specific characteristics of agents (nature of decisions), and contextualise an operational strategy to an effective technical implementation. Additionally, styles offer a plenty of tools to develop systems more easily traceable and evolvable, instead of traditionally been largely informal and *ad hoc*. An architectural style will only be adequate, unless the designer knows the characteristics of agents in the context. In addition, Architectural styles define the characteristics of the planning system structure, so deciding which architectural style and feature to use is always difficult. Batch style, for instance, arranges and controls a collection of simple and atomic components, having neither concurrency nor interactions between those components. The problem of this style is that the system has a tendency to become large and sometimes slow with time. Conversely, Pipeline style produces the first output quickly, which is very useful in behaviour-based systems. Unfortunately, Pipeline style may be too complex to program due to the incremental process. In addition, its cyclic structure supports feedback and loops, that is, later processes can start before the earlier ones have finished. In Main-program-and-subroutines style, a main program controls modules; and consequently their subroutines are aggregated to modules. This style is similar to top-down or hierarchic reasoning. This sort of style is very useful to develop an emergent reasoning in agent structures. But, the correctness of modules/subroutines depend on the correctness of the subroutine they call. Object-oriented style uses encapsulation to hide certain information and offer management of objects. To this style, we can interpret the idea like atomic control modules hiding details from others and sharing only the

necessary data. Layered styles behave like an organisational pyramid with its sub-levels. A control module, sub-system, or layer represents a highly coherent set of functionality, which suggests high internal coupling and low coupling with external entities. Many architectures have used this concept to overlap/delegate behaviours to their subordinates. Event-based style and Data Centred architecture deal with access and update of shared data, like a repository of agent experiences. Independent Component styles support shared data among heterogeneous components. For instance, in communicating process style, control modules are independent process, connecting others like a point-to-point network.

SOAR is a sort of architecture that pertains to the traditional cognitive school. SOAR has proposed a robust and centred architecture of production rules that simulates the human cognition, but SOAR is not inspired on it. SOAR can perceive the environment changes, learn, remember, and decide the best goal by logical reasoning. Unfortunately, long simulations are its weakness. Subsumption is an example of an architecture that does not use symbolic approach. In a Subsumption architecture, modularity and hierarchical arrangement of modules are advantageous to control agents in static, dynamic and unknown environments. Notwithstanding, it depends entirely on the world state to obtain the next "decision", and sometimes the current world state is not sufficient to provide the necessary "decision" about what to do next. PyramidNet brought some light to the development of intelligent engines with minimisation of the scalability problem through biologic plausibility, but automatic learning and imprecise general answers are a real trouble when logic is required. Finally, Society of Mind is a very articulated architecture about how mind works, and how simpler pieces of entities working together in different levels can emerge the same intelligence that we know in human beings, but Minsky is in his own "micro word" in the same manner as its agents. His proposed architecture was proven to be extremely scalable in practice but difficult to implement.

Note that agents can be built with different but interlaced styles and a superior level that controls them. Thus, we may have a meta-architecture, that is, an architecture behind of an architecture. Each architecture is an indication of the context, and meta-architecture is a promise of a set of high-level decisions (strategic architectural choices) integrated in the structure that will strongly influence the development/actions of the future architecture, its objectives, and the nature of agent decisions. Meta-architectures collect lessons from past experience to activate some strategy (architecture characteristic). Meta-architecture lays about previous foundations, laying out the high-level path toward the architectural vision, before diving into system decomposition and design of architectural mechanisms.

This chapter showed that there is no unique and right architecture capable of solving all AI problems, but the union of useful particularities of each architecture

can emerge as a most promising and robust approach.

Chapter 4

AFRANCI for Multi-Strategy Learning systems

This chapter introduces the "Architecture FoR AgeNts with Cognitive Insights" (AFRANCI) tool, an intuitive and visual resource adequate for designers to develop Multi-Strategy Learning systems. AFRANCI provides a set of features, such as pre-encoded libraries, and heterogeneous Machine Learning algorithms to assist in the design, train, test, and deployment of cognitive agents. The chapter ends with the description of a set of experiments used to assess AFRANCI.

4.1 Introduction

It is well known that different problems require different strategies to be resolved. The development of reliable Multi-Strategy Learning Systems is most often a hard experience for common users. This chapter presents the AFRANCI [191] tool that helps common designers to develop cognitive agents in an easy and very efficient way. AFRANCI was developed to be more than a resource for the development of Multi-Strategy Learning Systems [193]. It also represents the combination of heterogeneous ML algorithms.

AFRANCI combines reasoning and behaviour levels, and constructs new agents by composing agent structures with drag-and-dropping specialised modules linked with external libraries. The tool offers flexibility, extensibility and integration of symbolic and connectionist approaches in the same environment.

4.2 Background

Cognitive Agents (CA) can be modelled as a collection of interconnected control modules with well-defined interfaces and fine tuned behaviours. A popular CA architecture considers several control modules arranged in horizontal and/or vertical layers to combine behaviours in different levels of abstraction. See Chapter 3 for a detailed description.

In order to implement a CA architecture from scratch, it is often required experts in Artificial Intelligence and/or Robotics. Unfortunately, designers and programmers tend to produce different programming codes, which may reveal problems whenever the code needs to be extended or updated. Subtle usability problems always creep in during implementation, as well.

To overcome such incompatibilities, MatLab© [147] and SNNS© (Stuttgart Neural Network Simulator) [256, 257] tools have adopted graphical user interfaces and some AI resources. Unfortunately, MatLab and SNNS have some limitations that compromise the agent architecture development of this thesis. The limitations of MatLab software are:

- There is no possibility to design a modular structure composed of several heterogeneous controllers interconnected in different levels of abstraction;
- Although the automatic code generation was offered, the final code is always,
  - fat to be built-in in small devices with low processors and memory;
  - highly complex to be understood due to the communication among internal procedures and routines;
  - rigid to be extended;
- The compiled code cannot be generated from the whole project.

The limitations of SNNS are as follow:

- A simple graphic environment;
- Limitations on the behaviour level development, only offering a level at time;
- There is no support for,
  - working with multi-environments;
  - training at the same time interconnected ANN;
  - a preview of the complete proposed structure;

– working with heterogeneous collections of control modules.

- Expandability is limited concerning new ways of graphic design.

4.3 AFRANCI Features

AFRANCI combines the best of software engineering to offer resources to achieve an effortlessly construction of an efficient model of a Multi Strategy Learning system. Figure 4.1 presents the AFRANCI splash screen.

Figure 4.1: The splash screen of AFRANCI tool.

AFRANCI lets designers handle a heterogeneity of learning algorithms in the same environment of development. Designers are responsible for choosing the components and assembling the agent architecture based on the specific context. Designers are responsible for arranging the components in the workspace (see Figure 4.2). In general form, users may represent both of them. workspace.

In AFRANCI, different designers can model a symbolic-connectionist system by arranging and linking heterogeneous learning modules on the screen, with a minimum knowledge of programming language. They may also work in parallel to modify the flexible agent architecture by changing the order or type of the control module, inserting or removing ones, and solving the problem of rigid agent architecture construction. After the training of the model or part of it, a concise source code can be generated. These features enable the simplest, lightest and fastest way to produce elaborated decisions and behaviour systems for agents.

The facilities offered for users to perform all development phases until an enhanced behaviour is achieved system file were carefully planned. Consequently, users do not need to use earlier holistic enhanced behaviour structures of other tools because AFRANCI promotes by itself clean and flexible routines.

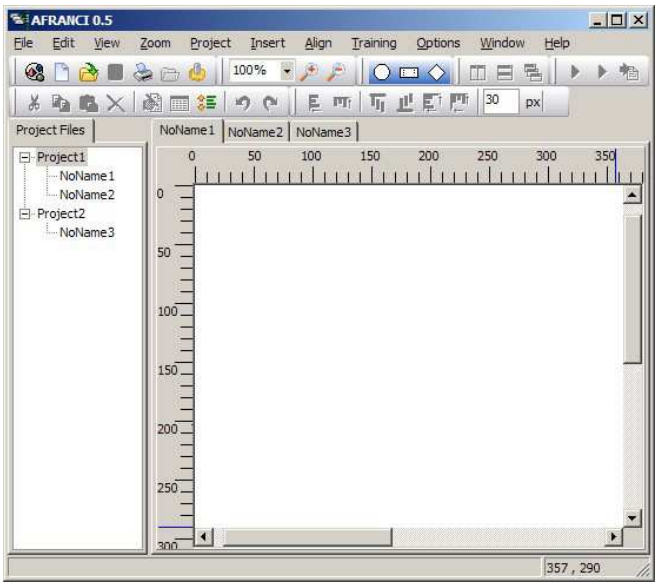

Figure 4.2: The environment used to plan the architecture.

4.3.1 AFRANCI Support for Reusability

Reusability is a technique to build larger things from existing parts, and to identify commonalities among those parts. Reusability also defines a degree of independence of a component in a system [85]. The highest degree of reusability means the component is more independent. The importance of reusability has influenced the development of methods during the design-time process [171]. Components previously developed are the main difference in a fast and reliable learning systems.

The AFRANCI tool uses the reusability concept to develop a set of cohesive and loosely coupled visual components (inputs, outputs, and control modules). Visual components save encoding time and eliminates bugs. From code, AFRANCI supports two control concepts: Behaviour Patterns and Templates. Behaviour Patterns are solutions formally defined and improved to develop projects. Each pattern describes a problem and the core of the solution. Behaviour Patterns offer sound techniques to be shared in different projects. Alternatively, Templates are prototypes that decrease the cost of models by (re)use. Prototypes refine the project ideas by multiple iterations, gradually moving from low-fidelity prototyping to high-fidelity representations of Behaviour Patterns.

4.3.2 AFRANCI Workspaces

In recent years, Multi Strategy Learning systems are becoming complex enough to be hard to plan. The best way to plan a complex system is dividing it in

small and specific sub-systems. This natural divide-and-conquer strategy can be compared to an evolutionary system. In this sense, complex systems can be partially developed into sub-systems, and each sub-system has its inputs, outputs, and control modules. Thus, different research centres can collaborate and exchange messages among them. Unfortunately, this computational resource is not so common in tools that offer learning algorithms.

AFRANCI Workspace was developed to support heterogeneous sub-systems (like a puzzle) and to promote sharing of models among team-mates at design-time development. Workspaces let users develop, debug, manage, manipulate, analyse, and tune sub-systems quickly and efficiently. In addition, they help users to avoid costly mistakes at development phase by training and testing sub-systems at first. A virtualised environment composed by two workspaces is shown in Figure 4.3. A large structure made of arranged control modules can be easily made collaboratively in different windows by common users.

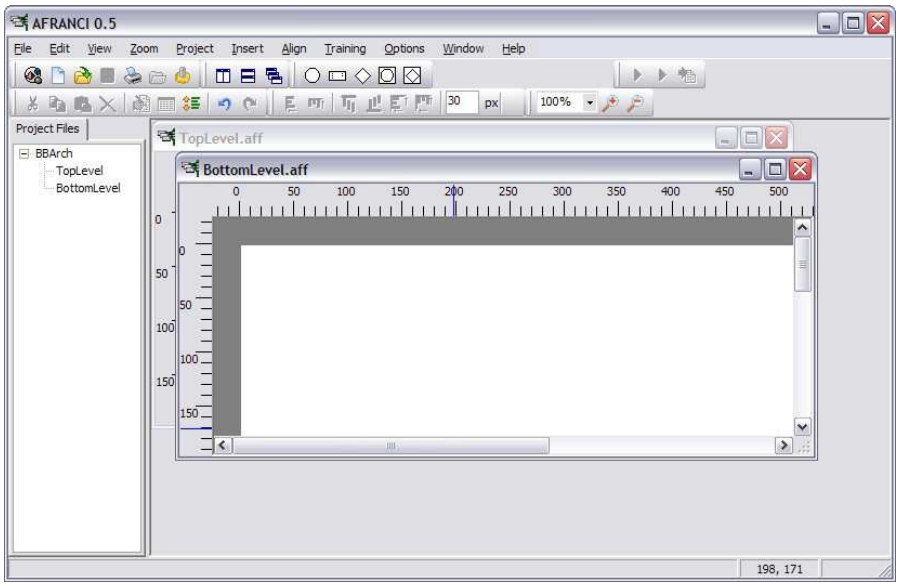

Figure 4.3: AFRANCI Workspaces.

4.3.3 The integrated Machine Learning Libraries

AFRANCI overcomes the lack of integration of heterogeneous learning algorithms in the same environment, to offer homogeneous environment conditions with easy access to learning algorithms (control modules) without any need of programming. Tuning and linking control modules like a circuit diagram, control modules will communicate from their interfaces to comprise a global solution among them. Thus, this resource completely hides from regular users' view the internal com-

plexity of the process. Users only need to know which learning algorithms will be set in every control modules.

To designers, the integration of control modules brought the following benefits:

Choice of components at design-time level, and still enjoy the benefits of using a single-source integrator that provides resources, in terms of training and system integration expertise, to assure that the system will work properly;

Improved user productivity by integrating many sub-systems in a large project and, consequently, improving the behaviour complexity of the final system. The facility of modular integration reduces the complexity of programming and the high cost of team-mates' training;

Sharing of information among objects effortlessly is the key benefit of AFRANCI. Information from the integrated sub-systems run more efficiently and at lower cost;

Cooperation among team-mates reduces time of development, increases safety, maximises time, and improves the effectiveness and productivity;

Expandability by adding new control modules or replacing the unused ones in older systems. Consequently, the older systems are extended at lower costs.

Users can handle several learning algorithms of many libraries at the same time. AFRANCI has two internal learning algorithms, such as Feedforward and Recurrent ANN, and supports external Machine Learning libraries, such as the ones available in WEKA [250], CN2 Induction Algorithm [41, 197], and GAlib [158]. Thus, users access an useful repository of algorithms to data preprocessing, classification, regression, evaluation, clustering, stochastic search, fine tuning, and association rules.

4.3.4 The AFRANCI Internal Structure

AFRANCI internal structure is made of Behaviour Patterns and Templates. Using modularity, high cohesion and low linkage, programmers obtain clear and strong methods to reach high degrees of inheritance, and possibilities to extend features from generic to specialised behaviours. In addition, the robustness of the structure allows AFRANCI to support external events but controlling the main application execution, such as supporting abnormal and non foreseeable conditions.

AFRANCI was built on MVC architecture local idea. MVC stands for Model, View and Controller layers. MVC is the way that the code was organised in AFRANCI. MVC uses DRY (Do not Repeat Yourself) to construct models or

objects. The MVC divides the functionality involved on application changes and presentation of data. Model layer encapsulates data in one place to access features of the application encapsulated to the controller by the Model. View layer represents what user can see and it is the most used by designers. It receives the data input and presents it at the output. It is not focused on how or where the information was obtained. View renders the content of a particular part of the Model and sends to the Controller the user actions, accesses also the data of the model via controller, and defines how these data should be presented. The Controller is the part of the architecture with define the behaviour of the application. It is responsible to interpret the user actions and map them to model calls. Also, the controller processes and responds to events, such as user interactions, and invokes changes to both model and view.

The AFRANCI internal structure is divided in three main parts [195]:

Integrated Machine Learning Libraries (iMLL) offers a repository of internal Machine Learning algorithms and connects to external ML algorithms as well. Additionally, iMLL offers normalisation methods that translates data to be sent to the control module;

Graphic Environment (GE) offers a set of main classes for modelling all visual components. For instance, users design their control models by assembling and linking visual components on the development environment;

Automatic Code Generator (ACG) represents a high-performance interpretation algorithm to automatically encode the diagram into a clean and ready-to-use standardised C++ open-source code.

The three main parts of AFRANCI structure are presented in Figure 4.4, which will be detailed in the next sections.

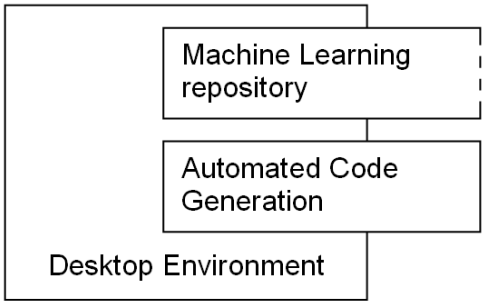

Figure 4.4: The general AFRANCI structure.

4.4 Designing a System's Structure in a Nutshell

We now demonstrate how we can, in three main simple stages, easily develop a behaviour-based structure, showing therefore the feasibility and efficiency of the tool. The Design and Set Up, Module Training and Code Generation stages will help users to avoid time consuming projects that comes from confusing lines of code.

4.4.1 The Design and Set Up stage

AFRANCI has a workspace to design and set up systems, like a circuit board of modules, called Desktop. AFRANCI tool was planned to develop and link several modular sub-systems, in the design-time environment. Design-time properties are characteristics of well-designed objects that influence either the visible format or their execution. AFRANCI builds a behaviour system model using graphic elements in the form of control modules. Users have full control of the hierarchical dependencies of the control modules as well as their portability to different contexts.

The design of the structure and the choice of the ML algorithms can be manually made by disposing graphic objects on the Desktop or can be automatically made by the use of the wizard.

Figure 4.5 shows the main visual parts used to diagram a control structure. The efficient highly interconnected model composed of inputs (circles), an output (lozenge), and a ML algorithm (rectangle). For instance, from left to right, three sensors (s_1,s_2,s_3) connect with three input ports (i_1,i_2,i_3), and the output port (o_1) connects with the actuator (a_1).

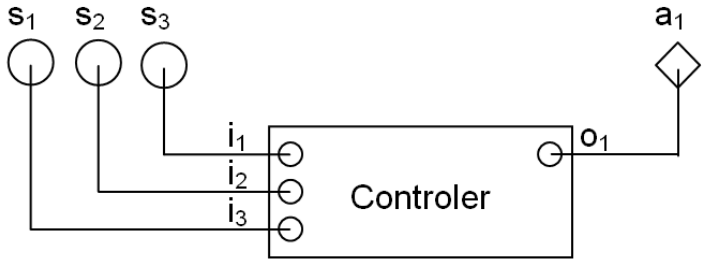

Figure 4.5: An example of an AFRANCI component and its parts.

The Manual process offers resources to perform drag-and-drop actions to arrange the input port, the control module and output port on the desktop area. Using drag-and-drop is a convenient way for users to manipulate objects, each visual component can be dragged and dropped from the standard

component palette onto the blank area of the form. The development plan starts with users opening an existent file or creating a new blank project, see main workspace in Figure 4.2. An input port can be an attribute or a sensor, as well as, the output port can be an actuator or an output variable. Every module is set to process the CSV (Comma Separated Values) sample files. A sample file provides data for learning algorithms in the training stage and defines the actual input and output ports. To develop a complete wired network, the interconnections are established from the input port to the input of the module and, consequently, from the output of the module to the output port. The users can change the default parameter values of the objects or choose the learning algorithm by accessing the module properties. For example, it is possible to set preferences like colour, dimensions, label, and coordinates of the objects or to choose different learning algorithm. In this case, further knowledge will be required from the users;

The Wizard helps users to upload training files and set up the correspondent ML algorithm, Figure 4.6. A Wizard or an Automatic Project Constructor (APC) is a piece of machinery that interprets CSV (Comma Separated Values) data set files and automatically draws a diagram in a workspace. The CSV file represents a training set of facts stored in a logic table. AFRANCI interprets columns of the table like input sensors, output sensors or output of control modules, and lines of the table as individual facts. Each CSV file can also be interpreted as an independent control module. If APC detects that there is relation between input and output elements of the diagram, a system will link and show it on the screen. In this sense, APC speeds up the development of the circuit diagram on screen.

Despite its graphical environment complexity, AFRANCI windows are more sophisticated than similar tools. Users can magnify or reduce images by Zoom mode. In addition, the right mouse button invokes a pop-up toolkit menu to customise visual components on the form. User can activate many other AFRANCI functions either with mouse or keyboard.

4.4.2 The Train and Test stage

The automatic training process is started after the agent structure was diagrammed, but before launching the training phase, AFRANCI checks if the whole system was fully interconnected as a single structure, and the training set files were load by the corresponding modules.

The automatic training process uses data flow sequence and low coupling to fire the training sequence and validate control modules in different processes,

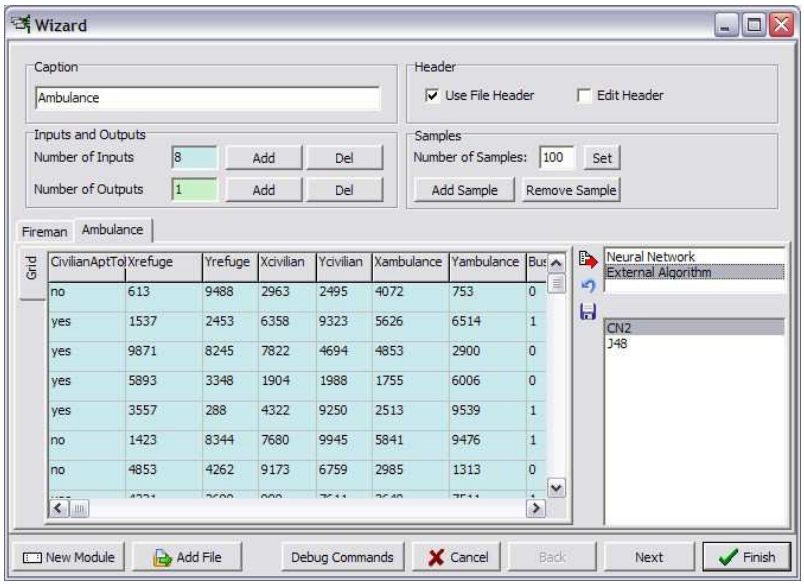

Figure 4.6: The Wizard window interface.

independently of the horizontal or vertical architecture level. Finally, the last inspection checks if all nominal values were load from data sets and previously normalised as numbers to be compatible with control modules. All other validations were made at design-time. It is worth mentioning that AFRANCI supports $K-fold$ cross validation is an advanced statistical technique useful to assess the bias in the training set results.

The $K-fold$ cross validation technique enables users to assess the stability of different structures, but confirming and validating the initial analysis [118]. For instance, the $K-fold$ cross validation verifies if the result of the first identified structure is repeated when investigated on the next structure. In such a process, the original set of samples is divided into K proportional subsets. From a collection of K subsets, the $K-1$ subsets are matched in a training set and the remaining subset is then a test set. Obeying the cross validation rule, each K subset will be received as a test set and the $K-1$ will be used as a training set until the K cycles finished.

The main benefits achieved by the use of control modules are presented as follow:

Avoiding bottlenecks by using multi-thread to train or test different control modules;

Multi-threaded of tasks enhance the editing, training, testing, and graphic analysis of control modules.

4.4.2.1 Wrappers

Almost all ML algorithms have parameters to be tuned in order to achieve optimal results. An experienced practitioner knows that changes in the parameter values may lead to quite different results. To tune system parameters, experts are required to have a deep knowledge of the system. Unfortunately, there is no cookbook made by experts that explains how to do this in order to achieve the most promising results. This is most often a severe obstacle to the wide spread use of such algorithms.

We foresee that the main reason of these unsolved problem is the substantial varying of the parameter values to tune each learning algorithm. Since an user has extensive experience to design and set up a control module to the problem, it is more difficult to obtain reliable results only by using empirical and manual set up.

As proposed by John [105] one possible approach to overcome such a situation is by the use of a *wrapper*. A *wrapper* produces several models using different combinations of parameters in the learning algorithm and returns the most promising model. The wrapper technique used to fine tune the learning algorithm was Genetic Algorithm (GA) [96].

In our tool the wrapper chooses the best parameters of ANN and, consequently, the details to set parameters of learning algorithms are completely hidden from the user. It is therefore a way to make the tool usable by a wide range of users.

4.4.3 The Code Generation stage

The Automatic Code Generator (ACG) has been implemented in AFRANCI tool to translate the diagram into a ready-to-use programming code, saving time and avoiding any percentage of programming error made by user. The purpose of ACG is to keep common users apart from the software engineering cycle, that is, instead of writing programming code, users should spend their time on elaboration phase of autonomous agent structures because the codification process is a responsibility of the tool.

The benefits that ACG provides are:

- Automatic generation of a standardised output of the diagram. This implies that if the agent produces a wrong behaviour, the designer needs only to fix the diagram, instead of debugging the whole source code;
- Clear and concise code generation, free of bugs, because ML libraries were already tested;

- Time and costs saving through automatic code generation.

The basis of ACG implementation relies on the fact that most projects created in the early stages of software development arise from diagrams. Since a project has all the visual components interconnected, it can be automatically encoded. Independently from the amount of elements used in the diagram, a short and clean source code is automatically generated, and ready to be used. Parsing the project into a ready-to-use source code is performed in a single step. ACG uses internal routines to recognise input/output ports, and control modules to encode into classes, objects, attributes, associations, and other information that compose a source code file. In the encoding process, users do not have to worry about syntax, creating class hierarchies and interconnection between Machine Learning libraries. As a designer, it is possible to specify which visual elements or sub-projects will participate on the encoding process. The encoding process is completely hidden from the users, and does not depend on the nature nor complexity of the graphical project, making an easy operation to common users. As C++ programmers, users have more sophisticated needs. Fortunately, AFRANCI recognises these needs, and provides benefits to easier the encoding process of the project. For example, Figure 4.12 presents a screen shot of the output of an encoding process.

4.4.4 Experiment: "Building a Rescue Decision System"

This experiment is the extended version of rescue civilian problem, first described in chapter 2. In this extended version, other modules and input variables were added, making the architecture more complex. The main independent variables include: the coordinate (X, Y) of the ambulance, fireman, building on fire, fire brigade, the nearest refuge (rescue building), and of the civilian; the life condition measure of the fireman and civilian[1], the building volatile information that is composed of earlier burnt, state and structure; the state of the ambulance (busy/free) to receive the civilian and of the fireman (busy/free) to extinguish the fire or to rescue the civilian; and an estimative of difficulty to rescue the civilian.

4.4.4.1 Design and Set Up

Figure 4.7 presents the modular Rescue Decision System, with heterogeneous modules linked among them and input variables composing an intuitive circuit diagram. Not perceived in the figure is the heterogeneity of the ML algorithms included in the control modules. Referring to the module labels in Figure 4.7 the following algorithms were used.

[1] A measure between 0 and 100 of the energy the fireman can use.

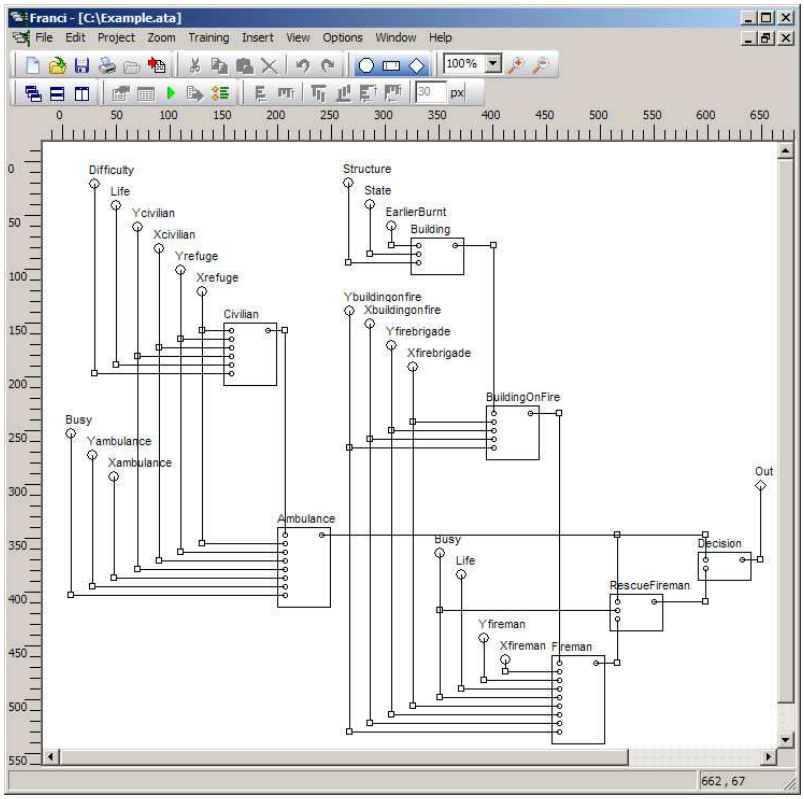

Figure 4.7: The architecture of rescue decision system (extended version).

The module *Civilian* was assembled using a feedforward ANN. The modules *Ambulance*, *BuildingOnFire*, *Fireman* and *RescueFireman* were assembled using CN2 rule learner. Other modules, such as *Building* and *Decision*, were assembled using WEKA's J48 Decision Tree algorithm. The user's drag-and-drop operations were: i) to drag the visual components, and to drop them on the black form; ii) to connect them and; iii) choose input and output variable names; iv) feed the system with the data set; v) train the modules in the correct sequence; vi) export C++ source-code that encodes the whole system.

The module *Decision* decides which agent, fireman or ambulance, will rescue the injured civilian (see Figure 4.7). There are cases where the fireman and ambulance are capable of rescuing the civilian at the same time. To solve the conflict, the module *Decision* decides in favour of the ambulance agent because a fireman agent has other priorities such as to extinguish fires in burning buildings with the aim of preserving the city. Rules induced by CN2 check if the ambulance is entirely apt to rescue a civilian obeying two main rules: (rule 1) if the ambulance is occupied then it is useless to attempt the rescue; (rule 2) if the civilian has not enough "vitality" then it is also not rescued; (other rules) the civilian will be rescued if it has enough "energy" and the ambulance is localised between the

civilian and the rescue building otherwise it will not be rescued.

4.4.4.2 The Train and Test Module

The training sequence is launched independently of the abstraction levels. In this new way to train interconnected ML algorithms, AFRANCI tool starts identifying and compiling the atomic control modules at first due to the output data that atomic control modules offer to feed other dependent control modules. The training stage of three control modules, with the built-in CN2, J48 and ANN ML algorithms, are shown in Figures 4.8, 4.9, and 4.10, respectively. Each trained module receives a bold border as shown in Figure 4.11, in which the user can follow the cycle of training. At the end, if all modules have been successfully trained, the user can export his/her diagram to a code (see Figure 4.12).

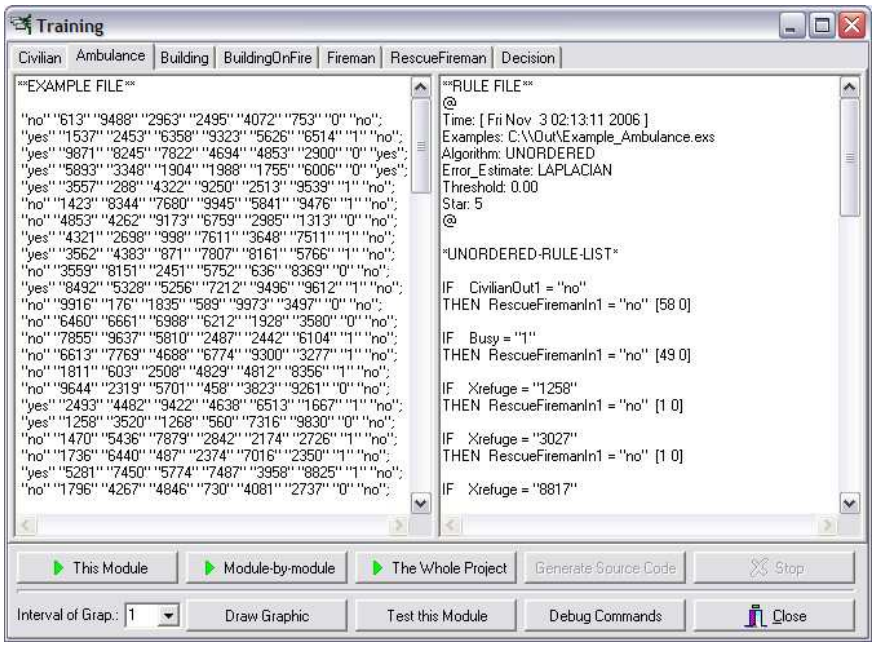

Figure 4.8: CN2 training phase (module *Ambulance*).

Figure 4.8 presents the training phase of CN2 algorithm where module *Ambulance* was trained. The rules composing the model are depicted on the right hand-side whereas the raw data is shown on the left hand-side. Figure 4.9 shows the training phase of J48 algorithm in which module *Building* was trained. The output tree model is shown on the right-hand side whereas the raw data used to produce the tree is shown on the left-hand side. Figure 4.10 shows the training phase MLP with Backpropagation algorithm where module *Civilian* was trained. The error curve on the left is decreased by iterations showed on the right.

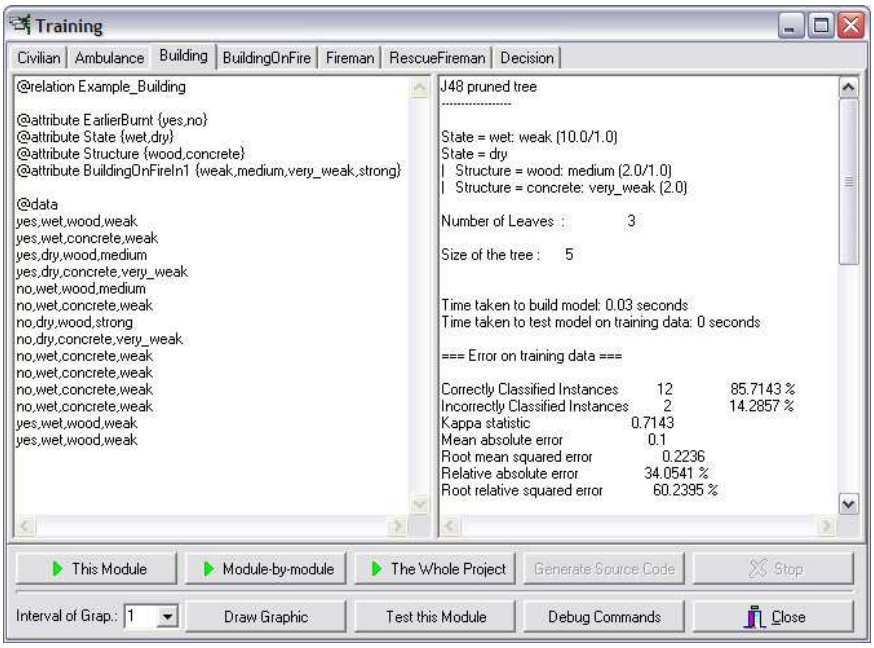

Figure 4.9: J48 training phase (module *Building*).

4.4.4.3 Code Generation

An excerpt of a ready-to-use programming code was presented in Figure 4.12. The diagram project was encoded into a set of C++ code that can be edited as text in the full-featured AFRANCI ASCII editor or in any other text code editor. The source code is composed of classes, attributes, associations, and other features needed to run the architecture outside of the AFRANCI development environment.

4.5 Conclusions

In this chapter we have presented the AFRANCI tool for the fast development of Cognitive Agents. AFRANCI offers visual resource to diagram Multi-Strategy Learning systems and commands to generate code automatically from diagrams. The tool provides fast and intuitive features, such as pre-written code libraries, and integration of symbolic and connectionist ML algorithms to assist in designing, training, testing, and deployment of the agents. The process of linkage among learning algorithms, in the same environment, represents many advantages; consequently, a system can become robust and support high noise immunity, fault tolerance and programming by examples for new control architectures. The experiment presented in this chapter, as well as other examples in the rest of the thesis, show the AFRANCI benefits to develop agent architecture.

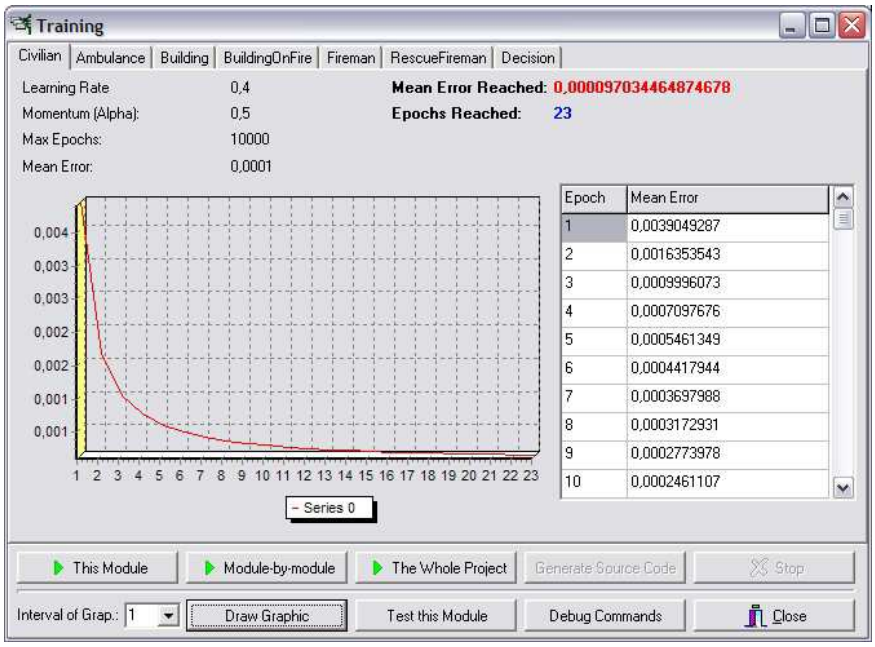

Figure 4.10: ANN training phase (module *Civilian*).

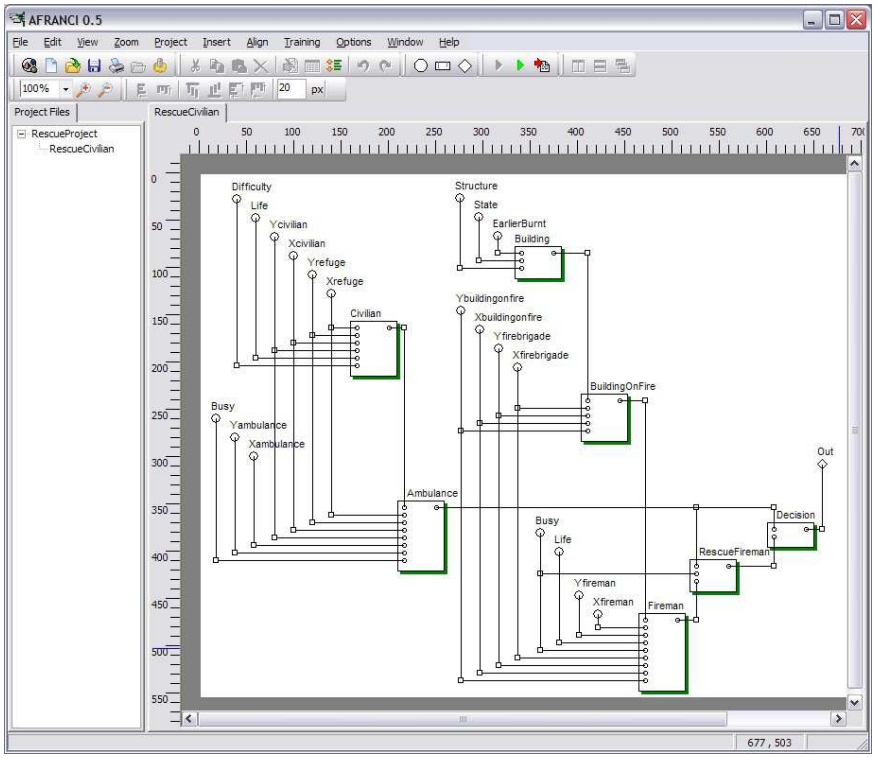

Figure 4.11: The Rescue Decision System entirely trained.

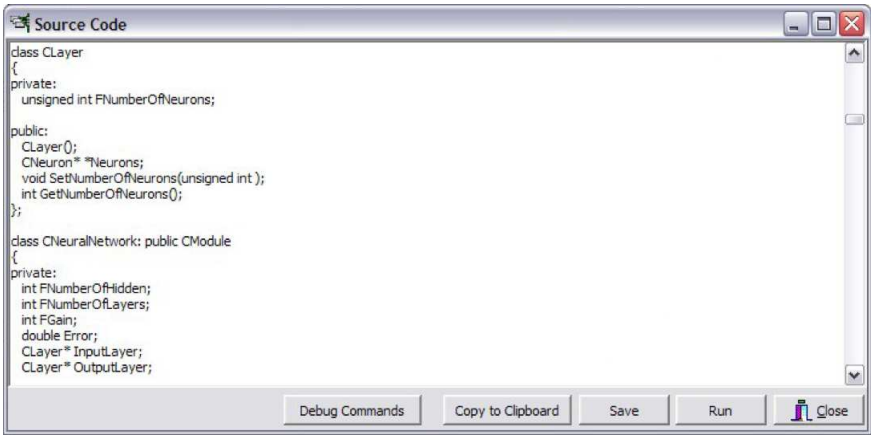

Figure 4.12: The AFRANCI ASCII editor.

Chapter 5

AFRANCI for Agents

The AFRANCI tool, presented in the previous chapter, combines basic functionalities required for general intelligent behaviours. AFRANCI works with both symbolic and connectionist approaches, and serves as prototype of state-of-the-art research on the hybrid approach. AFRANCI was created to be the support for emerging a "creative" thought through different micro-architectures. The micro-architectures "think" in parallel in order to find more than one solution for a problem. In the meta-level, the structure decides in favour of the scenarios.

5.1 Introduction

AFRANCI implements a new approach that may be associated with the prototype of state-of-the-art research on hybrid approach. It supports bidirectional communication between different levels of abstraction. This new proposed approach may then form the basis for understanding the emergence of knowledge in autonomous agents structures. The three main development aspects are: combination of hybrid approaches, modular and flexible structures, and cognitive and behavioural layers. A simple view of the AFRANCI model is shown in Figure 5.1.

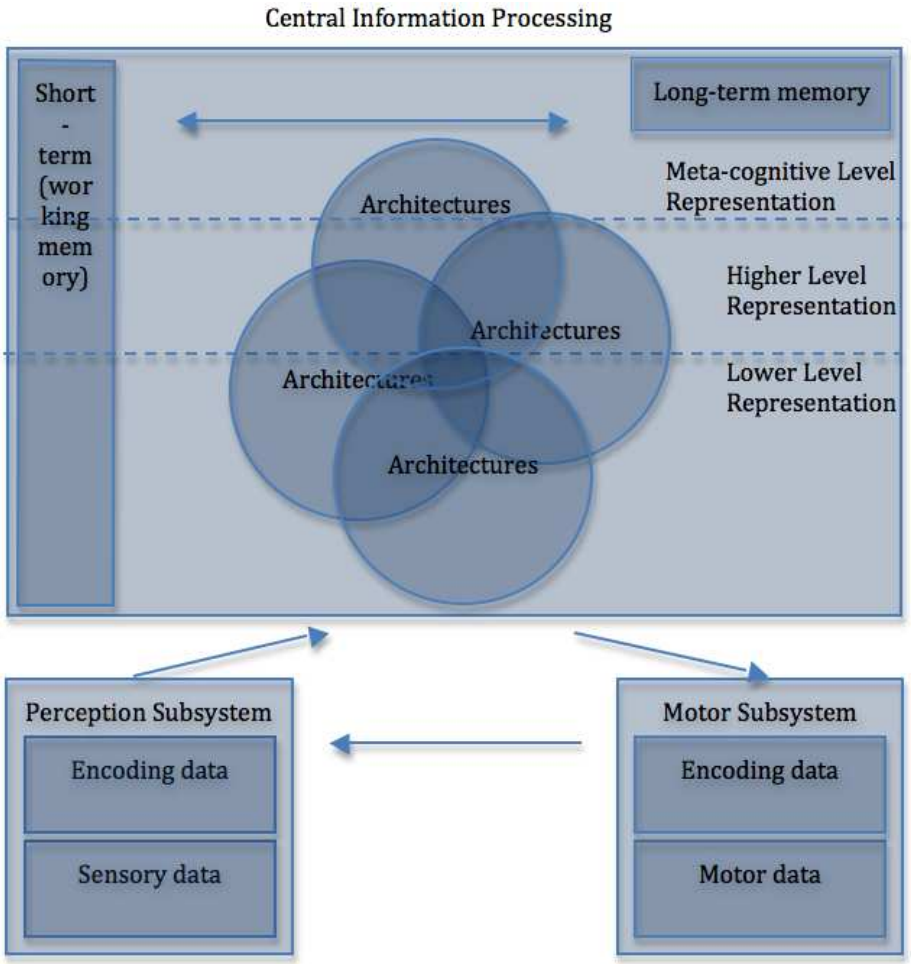

Figure 5.1: A global view of the AFRANCI (main components).

AFRANCI uses rational and cognitive principles to model the emergence of elaborated behaviours and to make decisions in favour of the most promising action for the agent.

Rationality uses the heterogeneity of structures to define the innate agent behaviours in the lower layers.

Cognition combines all relevant decision modules, short and long-term knowledge to define, in the higher layers, what should the agent perform next.

Simple or more elaborated behaviours or decisions are based on: (a) signals provided by sensors; (b) the content of working memory created to solve unknown problems; and (c) any other knowledge stored in long-term memory.

AFRANCI uses a meta-level to be conscious of its actions and capabilities. It is horizontally set in the highest abstraction level. In lower levels, modules

are arranged in parallel by levels of competence, but lower and upper levels are independent.

5.1.1 Motivation

Although classic AI architectures concede autonomy to agents, agents are restricted to foreseeable behaviours in a monolithic system. Unfortunately, monolithic systems imply pre-formed structures, which is their main weakness. As a consequence of their "pure" construction, monolithic structures do not allow designers to increase new components as in a building blocks formation, nor to share information between cognitive or rational approaches.

AFRANCI avoids the "pure" characteristic of the classic AI architectures and brings solutions for designers to update old agent structures. AFRANCI makes use of the most useful techniques and styles found in four main previous architectures, such as SOAR, Minsky's Society of Mind, Subsumption and PyramidNet architecutre. In this way, an agent can adapt in unknown environments and generalise new behaviours.

In fact, AFRANCI is a solid ideal for an integrated approach that harmonises multiple characteristics of other systems. The architecture has stand-alone modules for each different task that communicate over links. Through the incremental development of intelligence, designers can build new agent activities by the use of processes and competencies repetitively acquired, during the evolution and development phases. These include sensing the environment, building a representation of the world, and controlling the agent motors.

AFRANCI does not follow older philosophical questions such as 'what is mind' or 'what are the necessary and/or sufficient conditions for agents to be conscious'. In fact, AFRANCI focuses on several different kinds of "minds" of heterogeneous architectures and their capabilities arranged in a "correct" manner. Thus, designers may build new components to also fill in the lacunas of other agent designers, or change the performance to be top-down, bottom-up or hybrid. This is the main difference between AFRANCI and other architectures, as the next sections will explain.

5.2 Towards an Architecture

The flexible evolution of AFRANCI guides designers to (a) create new opened-mind states, which imply flexibility, tolerance to repetition, and receiving concepts; (b) rebuild concepts and evaluate them; (c) cope continuously with the maximum of adaptability; (d) achieve a satisfactory behaviour in each new existential situation; (e) build an agent structure, establishing communication among modules, without deep knowledge of programming language, differently from traditional architectures.

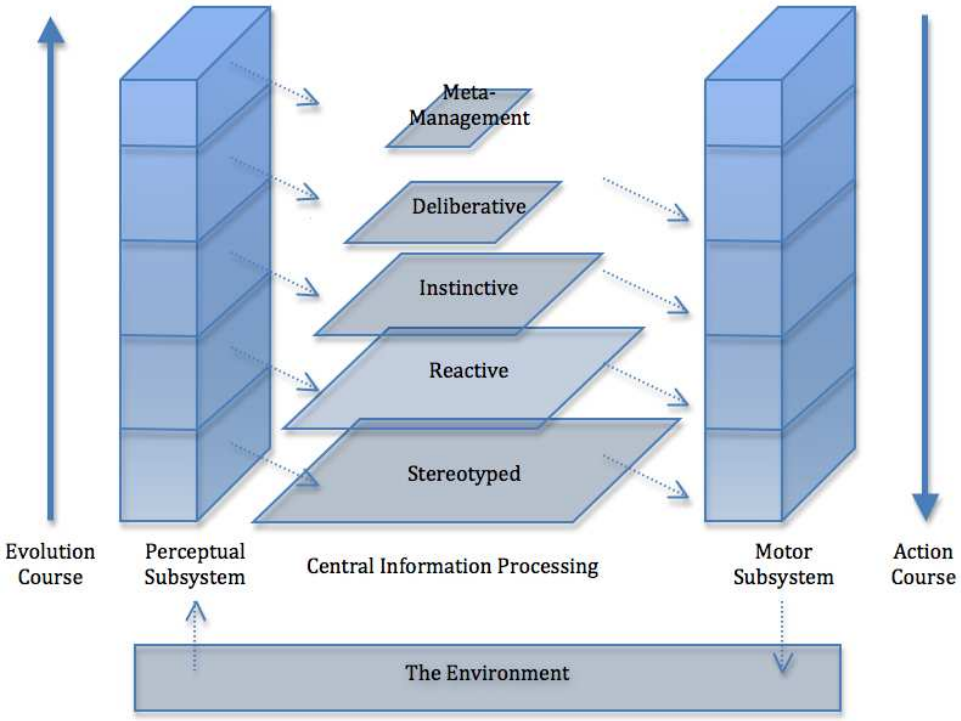

Figure 5.2: The prototype of AFRANCI.

Figure 5.2 presents the proposed AFRANCI architecture. We do not expect AFRANCI to be the "optimal" architecture constructed because there is no such optimal intelligent architecture design, as biological diversity on Earth shows us. Conversely, this model separates concerns, which helps designers to organise their decision-making process and actions; thus, they can develop the most promising set of concepts.

AFRANCI is vertically arranged in five abstraction levels of competence and evolution from bottom-to-top, that are, Stereotyped, Reactive, Instinctive, Deliberative and Meta-management (Cognition), and three horizontal functional subsystems, to be specific, Perceptual subsystem, Central processing, and Mo-

tor subsystem. Abstraction levels of competence and types of task-achievement behaviours are separated from function effectively in a hierarchically organised manner. The Perceptual and Motor subsystems are also divided into many abstraction levels as need. In accordance with Brooks [28], the main idea of levels of competence is that we can build layers of a control system corresponding to each level of competence and simply add a new layer to an existing set to move onto the next higher level of overall competence.

5.2.1 Learning in AFRANCI

Learning is based on decisions taken via agent, which could be by supervised or unsupervised methods.

AFRANCI is constantly performing a clockwise cycle (see Figure 5.3). The cycle involves to send signals from the environment to the respective control modules that can deal with, and send back to the environment the actions chosen by the architecture. The arrows represent the direction of information. Control modules use a built-in knowledge stored in training - learned *a priori* - and apply it to the current situation. Whenever an unpredictable situation happens, the generalisation happens depending on the architecture level. Therefore, the architecture directly supports the acquisition of new information.

Control modules can be activated or not by a fact. A fact connects (a) an event, (b) an environment action, (c) an output of other control module, and (d) a collection of input devices.

Each control module is ignorant of existence of others, but when working together they are capable of performing independent and specialised tasks by mechanisms of inhibition and excitation.

5.3 Levels and Layers

Layers are the implementation versus of the abstraction levels. Layers interchange messages with each other by means of bidirectional flow of information control - bottom-up activation and top-down execution - as explained on section 3.2.2.6 (of chapter 3). Module is the mechanism that interfaces signals between layers, others modules and external reality.

5.3.1 The Flow of Control Information

Basically, the flow of control information goes up to the sensory inputs and it reaches, at the same time, all modules of the lowest level in the architecture, as presented in Figure 5.4. For instance, if the Stereotyped layer has control over

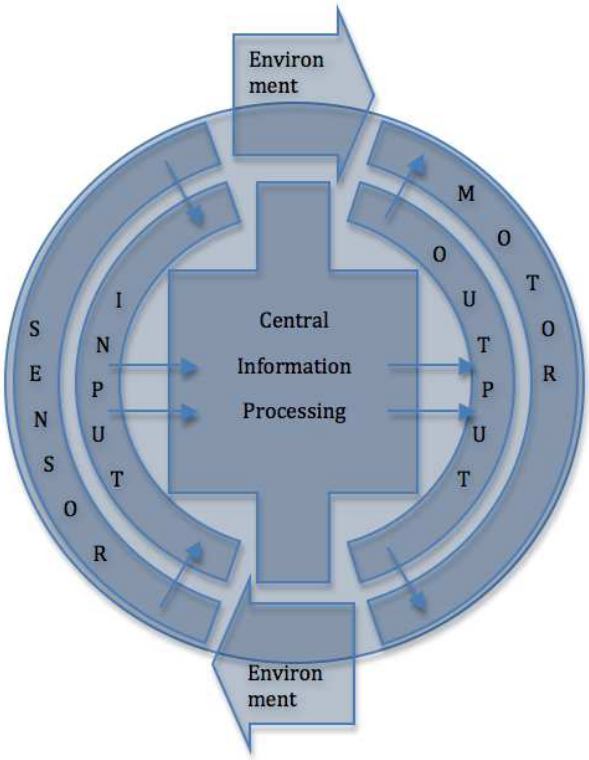

Figure 5.3: A model of learning cycle development.

inputs then it will do so in a priority manner, otherwise, bottom-up activation will occur and the control will be passed to the Reactive layer. This bottom-up activation will successively occur until the level of competence get the input. In another example, if the Deliberative level has competence to control a situation then it will do so, typically by top-down execution. Figure 5.4 presents the information flow between architecture interfaces and modules.

5.3.2 The Strategic Levels

The strategic levels have built-in a heterogeneous collection of internal and external but interrelated modules. The modules are implemented in accordance with the layer purpose into a coherent working system for building a modular agent mind. Such methods are in the form of symbolic (rule-based) system, subsymbolic or neural (connectionist-based) or both (hybrid) systems. Figure 5.5 shows a network of modules communicating between them along the structure.

In this scenario, Stereotyped, Reactive, Instinctive, Deliberative and Metamanagement layers interact with the external environment up to a degree. The interaction process happens via perceptual and motor action subsystems. Thus, sensory information acquired is converted to symbols, which are then processed

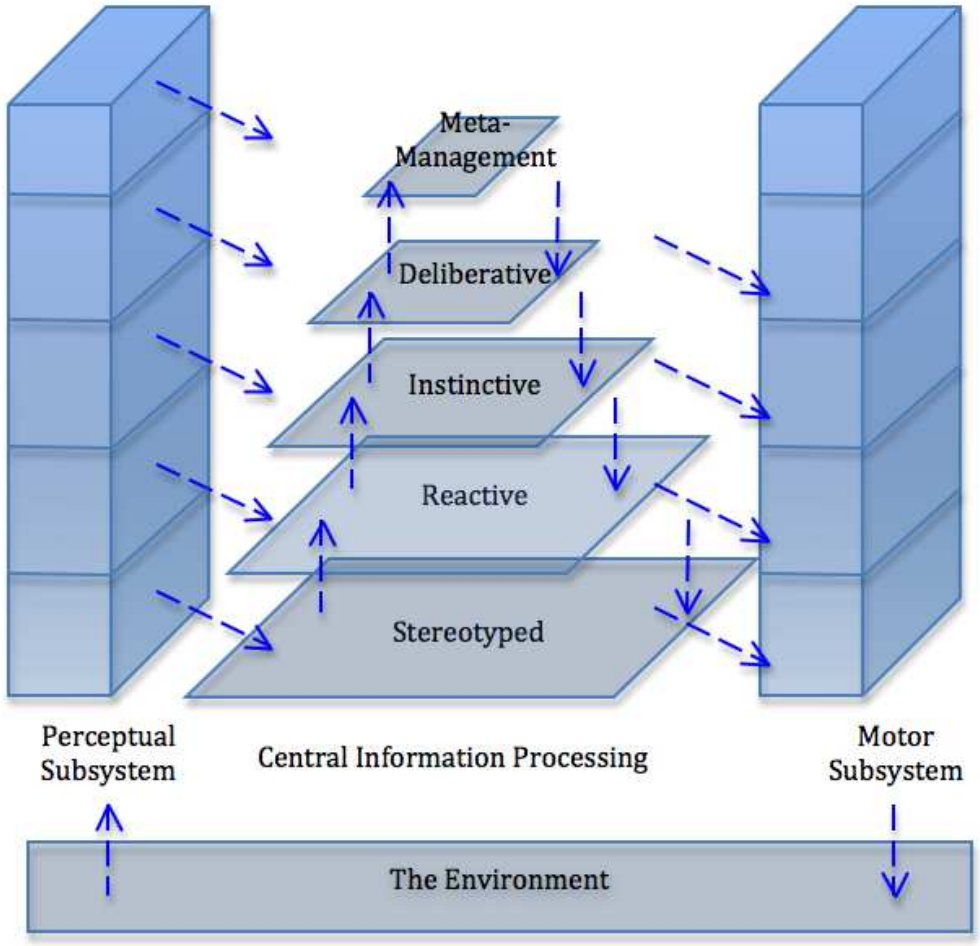

Figure 5.4: The Flow of Control Information.

and evaluated in order to determine the appropriate motor symbols that lead to generate motor actions - behaviours. Additionally, Instinctive and Deliberative layers internally interact with other layers.

Actually, the modular architecture is made of several layers. For instance, atomic reflexive behaviours compose the Stereotyped layer. A chain of atomic reflexive behaviours - or an innate cognitive modules - implement the Reactive layer. Instinctive layer controls a chain of reactive behaviours - or recurrent innate cognitive modules. Deliberative and Meta-management layers contain semi-autonomous controllers that represent the symbolic processing mechanisms of the system. Those modules have been described as the "building blocks" of knowledge and cognition. Symbols are described and entitled as parcels of information and then stored in memory, and retrieved for problem solving in a working memory. Particularly, the Instinctive behaviour supports working memory.

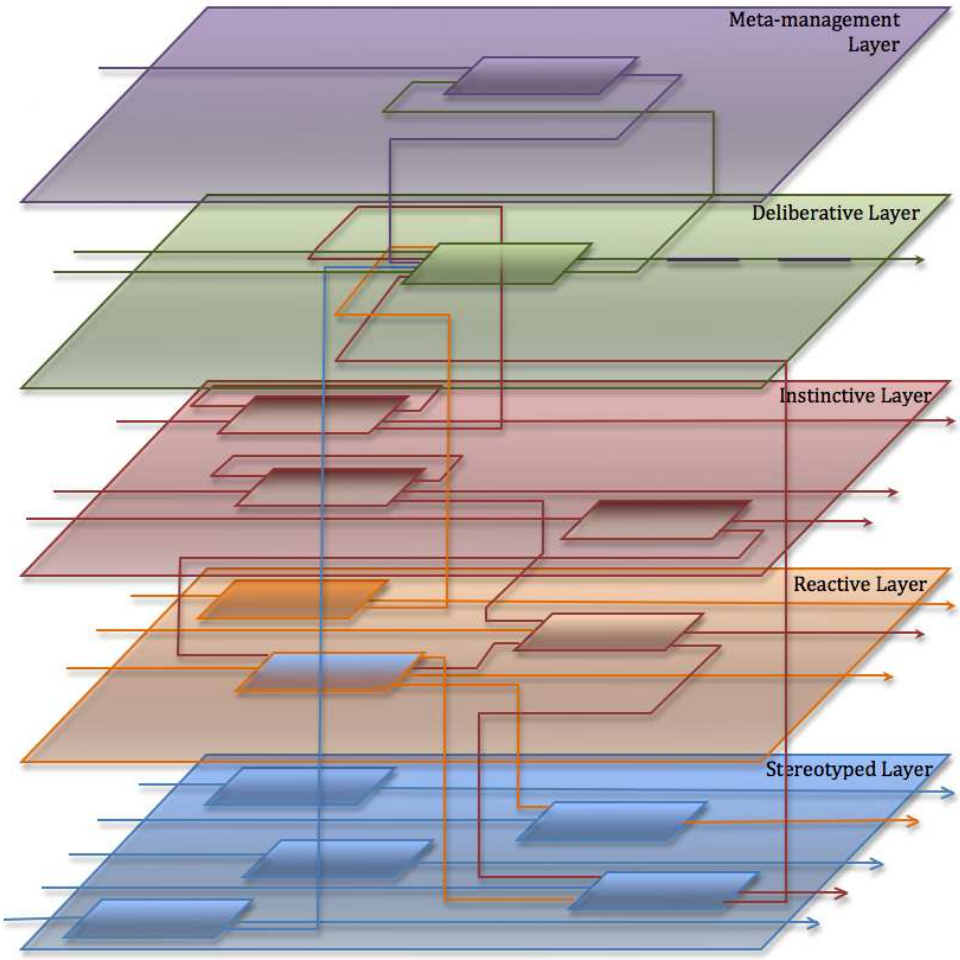

Figure 5.5: AFRANCI levels and layers.

Stereotyped, Reactive, Instinctive, Deliberative and Meta-management layers modules pertain to the central processing subsystem, mediating and moderating relations between stimuli and responses.

5.3.3 The Perceptual-Motor Subsystem

In the Perceptual subsystem, sensory inputs create a sort of representation of spatial relations in the environment. Over repeated experiences sensory inputs contribute to a higher level of representation about relations between objects. The representation permits predictions or expectations about them, a property that only requires that measure of activation about objects be maintained after they have disappeared.

The lower levels are directly in contact with the environment. Reactive and reflexive levels respond in a timely way to what is happening in the environment.

This low level has properties to sense-act without previous representations.

5.3.4 The Stereotyped and Reactive Layers

Encapsulated modules with innate information processing capacities fill in the lower layers. These rigid pre-compiled innate modules have rigid knowledge because of their own particular stimuli and goals were acquired during the evolution process. Thus, each of these modules can be developed independently of other modules, but associated to them to produce complex behaviours, as Minsky and Brooks proposed.

Richardson [201] believes that associations (group of encapsulated modules) are used not to explain mental states but to explain behaviours. Following this thought, we define cognition as the execution of behaviours and their regulations. Behaviourists argued that mind is too much abstract to be measured; conversely, only stimuli and responses can be observed.

Innate specified modules are implemented in Stereotyped behaviour layer by simple and specialised knowledge. Stereotyped behaviour layer is set in the lowest abstraction level. This layer provides many largely innate specified modules. The modules have a very fast simple maturation of pre-formed structures much of the organisms background knowledge. They can be implemented by rigid rules or feedforward Artificial Neural Networks with a specialised training. The Stereotyped layer input driven characteristic determines to the architecture a largely organised around bottom-to-top information flow.

The Reactive behaviour layer is set as a level, but is still set in the lower abstraction levels. It has been dependent on the adoption of certain assumptions and strategies. In the reactive layer, reactive structures interconnect well-formed innate specified modules producing a small chain of modules. Similarly, a chain of innate specified modules could be substituted by a little evolved module with a bunch of built-in pre-formed rules - processes. In fact, reactive modules can fire new changes or modulate changes launched by other events, that are, sensory inputs. The modules can be linked with other modules and form a fairly sophisticated reactive network with a variety of behaviour processes. Inspired by evolution, we have Artificial Neural Networks with tuned synaptic weights.

5.3.5 The Instinctive Layer

Innate behaviours, when controlled by Instinctive layer, permit the agent to transcend knowledge, towards a particular and elaborated instinct. The Instinctive layer does not focus on a specific behaviour, but controls long chains of reactive behaviours beneath it to produce cooperative behaviour with appreciable useful

difference. Instincts are also labelled as Fixed Action Patterns (FAP). FAP is the result of long chains of low-level atomic behaviour modules performing actions to reach a purpose.

The Instinctive layer contains a considerable intricacy structure of FAPs that are capable of going beyond the limits of the environment or consciousness thought. Instinct uses previous encoded knowledge to respond for specific external stimuli. The agent recall the instinct by finding the beginning of such a sequence in order to respond or react to certain sorts of stimuli, propagating its excitement into the other. The behaviour sequence runs to completion, but during the propagating of excitement, the agent is "blind" of external influences.

Inside of the Instinctive layer, rationality emerges from collaboration of a chain of stand-alone behaviours. The result of interaction among many reactive modules and the environment allows the layer to generate a symbolic action sequence without performing the corresponding actions, so the agent knows by anticipation what will happen.

The propagating of excitement can operate at different time scales. The layer can fire a bunch of innate stand-alone behaviour patterns contained in several reactive modules, such as feedforward or feedback ANN as well.

The responses to a given excitement are obtained by a sequence of reactions, but they are not notified to the upper level. This avoids decision taking, impasses, and control.

This layer uses a reusable memory in which the sequence of output can be built so that their consequences can be evaluated. Further developments could allow the memory to be used to construct more than one action sequence so that difference options can be compared and one selected.

As happens often in evolution this might be done by copying and modifying one of the pre-existing reactive modules. The modifications involved giving the module inputs from all over the system, making it work faster, and making it crudely classify inputs into categories relevant to a certain global behaviour.

5.3.6 The Deliberative Layer

At the Deliberative level, process and plan as well as prior decisions are taken into account when deciding on the next step of the agent. Decisions about whether new actions (motives) should be adopted or not happen all the time. To do so, the layer (a) receives output signals from some modules set beneath it as well as input signals of Perceptual subsystem. In accordance with the agent evolution cycle, lower modules may or may not be connected with deliberative level, (b) evaluates

and selects the next agent action in order to govern the actuators by explicit[1] or implicit[2] decisions, and (c) sends output signal to the Motor subsystem, allocating it exclusively.

5.3.7 The Meta-management Layer

In the highest cognitive level, the meta-management layer presents the promptitude of dealing with detailed management solution for unified control of multiple levels, which affect one another and the overall trends of the architecture (as shown in Figure 5.5). A meta-manager is conceived to manage the decisions of parallel modules throughout their functioning period and ensuring the combination of modules that contribute to the overall goals of the architecture.

The main goals include monitoring and evaluation the management of inferential internal processes, activation of new deliberative strategies, and behaviour changes and impasses detection.

Meta-management is set in the highest level of AFRANCI architecture. This top level implements consciousness to tune the agent to the "right" decision according to the situation in the environment. Nevertheless, the layer should have access, knowledge, and understandability of signals coming from the environment by the perception subsystem and from the deliberative layer.

Meta-management allows an agent to control its deliberative responses (states). Without this top layer, the agent would not be apt to identify and dynamically change its own behaviours. In this sense, the agent recognises itself as an entity in the environment. Consequently we can affirm that agent has "auto-conscience".

5.3.8 Short-term and Long-Term Memories

In computational neuroscience, memory is composed of interconnected processing elements entitled neurones. Each neurone receives signals (that are, adjustable synaptic weights) from neighbour neurones, except for those special neurones[3].

The structure of connections and the learning algorithm typify the long-term knowledge. Long-term knowledge aggregates new experiences in order to augment

[1]Explicit decisions are those understandable by human beings like `IF...THEN rules`, used as a white box model to explain the result provided by the model, which can easily be replicated by a simple mathematical operations.

[2]Implicit decisions are those codified by synaptic connection weights in Artificial Neural Networks black boxes.

[3]In Multilayer Perceptron Artificial Neural Networks, special neurones represent the input layer of the structure.

the cognitive universe of the agent. The propagation of activation of each neurone is a short-term reflection of the long-term structure. The network transmits information to its distal parts in parallel over a set of connections. The activation process is propagated until the network has reached the quiescent knowledge access. Quiescence refers to an external stimulus that triggers an organised behaviour in any time - behaviour is not necessarily innate, but generalisation can be reached. Perception, which occurs via the activation of special neurones, transmits knowledge from outside of the system to the inside. This specific behaviour changes the short-term activation of the system, without changing its long-term structure.

The long-term structure stores and recalls data or patterns, classifies patterns, performs general mapping from input patterns to output patterns, or finds solutions to constrained optimisation problems [65]. The knowledge is example of the whole base of representations stored in long-term memory. The relevance among knowledge, representations and reasoning is so strong that one complements the other. Reasoning is the process responsible to trail a situation to be understood. Whereas all knowledge of ANN are stored in the form of synaptic weights that will determine the behaviour of the network; conversely, Symbolic systems store them in the form of production rules, commonly referred as universal subgoaling.

Both Connectionist and Symbolic systems solve problems using the long-term knowledge. In order to retrieve information about existing problem and searches for a new solution decision, all systems develop solutions on elaboration phase (training). Training is a learning way that occurs via the adjustment of connection weights or division of production rules. Unfortunately, the size of long-term memory in Symbolic system needs to be extended, instead of simple adjusts of weights as ANN's proposes.

5.3.9 Impasse

Impasse is defined like a common situation that occurs in the higher levels when the system does not know how to proceed and then collapses. Discovery by trial-and-error is a sort of technique used to construct new solutions. To this technique, a nontrivial problem is presented to the problem space. Unfortunately, the problem space is either incomplete or inconsistent and annuls the technique. Nontrivial problems are also complex problems, which result in successive and inadequate or unknown answers of deductive logic and background knowledge in face of the initial problem.

Impasse is the result of partial, unpredictable, sequential and inadequate background knowledge in face of the initial problem. To resolve the impasse, AFRANCI architecture uses two cognitive strategies: convergent, and divergent.

The convergent strategy follows a logic path. It is most useful for situations under control, with well-done metadata that may be measured and predicted. For instance, in cases of Reactive layer, convergent strategy solves the impasses by making use of some classifier method or vector. Conversely, the divergent strategy searches in other domains for sufficient elements that could help it to solve that problem by analogy. The agent augments the problem space up to reach a satisfactory solution by acquiring new knowledge, which summarises the processing that leads to results, or up to another process inhibits it. For example, the agent changes the current synaptic connection weights by other synaptic connection weights stored in the database (long-term memory). This swap of new memories simplify the implementation processes, adapting the agent to inner beliefs already supervised. So, the agent uses new beliefs to take decisions and achieve answers - solutions. As it is was presented before, many unsupervised rules came collapse the decision system.

In a new problem space, which was augmented by the impasse, AFRANCI sets new knowledge by using synaptic connections weights or new production rules previously stored on long-term memory. Consequently the cognitive space of agent is augmented. Thus, the acquisition of new concepts is restructured by the problem space and representations of problems, that is, possible solutions extend the general domain by creating new concepts between facts.

5.4 Advantages of AFRANCI

In order to simulate reasoning in agents, knowledge represented by rules and tuned synaptic connection weights were both implemented. The result was the most promising multi-strategy plan with fast decisions.

Thus, we conclude that the main benefits of the architecture are:

- **Model an system at a high-level design** relieves the designer of significant system responsibilities, such as to fulfil particular requirements. AFRANCI comprises many control modules to emerge autonomy that can be understood at a large-scale abstraction level;

- **Support parallel and partitioned development** provides a structural decomposition of loosely coupled architectural patterns with clear responsibilities in the system design phase. Since the architectural components are relatively independent from each other, the subsequent development work can be partitioned. Each partitioned pattern may be analysed and developed by a team with specialised skills;

- **Hot swap long-term capabilities** automatically "rearrange" the whole system. This improvement offers the union of new collections of capabilities at any time; consequently upgrading old architectures;
- **Understand the human thinking** as a coherent and plastic to connect simultaneous control modules and simulate specific behaviours. Additionally, agent can take decisions based on its knowledge, learning and external environment interaction;
- **Cognitive indexing system** analyses many possibilities in face of the several neural records that do not make relation a priori with specific contexts, and searches its references by means of associations (relational) in events or other situations, avoiding mediation of the search (directive) as it is;
- **Consciousness level** is a mature general notion to solve real problems, such as a new technique based on function and decomposition. After that, the system uses a structure compilation to encode all decision taking into reactive mechanisms with high degree of efficiency. The differential of AFRANCI is on the capability of controlling its own reasoning process. Thus, the higher levels trigger actions to control modules beneath it;
- **Flexibility** of the meta-architecture handles different types of substructures, with the potential for self tuning; consequently diversifying the system. In general, there will not be unique design solutions. It is not to be expected that there is any one "right" architecture. As biological diversity demonstrates, many different architectures may be successful, and in different ways.

Chapter 6

Architecture Implementation and Experiments

This chapter presents an autonomous agent architecture that exerts control over behaviours in a simulated environment. We begin by describing both architecture and simulator and outlining the basic agent structure. We then take an inner look at each function and control module concerning the relation between implemented layers and evolution levels. From this background, we compare the proposed agent architecture with other techniques in order to find the most promising one. Finally, we consider how the agent architecture has adapted to its most complex functions.

6.1 Introduction

Autonomy and intelligence are different characteristics that focus on the same principle, the independence of the agent in the environment.

In order to capture the "beauty" of autonomy and intelligence proposed along this thesis, this chapter presents an architecture for computational agents that supports heterogeneous control modules distributed in distinct levels, as biologic human evolution dictates. The levels of importance act directly on mechanical and functional agent properties as well as control modules being governed by Conscience, which drives the agent to reach the target area.

The methodology used to evaluate the performance of the agent architecture is a simulated environment with certain operating conditions. The virtual world, known as CyberMouse, was used as the main environment due to the degree

of freedom (behaviour complexity) that any agent can obtain in the simulation. The agent will be exhaustively tested in different simulation phases. Each phase corresponds to different experimental prototype that obeys to a particular manifestation of autonomy.

6.1.1 CyberMouse Environment

CyberMouse is a simulator that virtualises a mouse, a cheese, a labyrinth, and the dynamic states of the world. Respectively: an agent, a target area, a virtual environment, and noise and/or latency. The primary agent goal is to reach the target area set somewhere in the environment. The agent needs to avoid collisions and reach the target area as fast as it can. The faster the target area is reached by the agent, the better is its performance, which reflects the right choices made by the inner system.

The labyrinth is a rectangular area with dimensions measured in U_m (a standard unit of measurement), the maximum 28 U_m wide by 14 U_m depth. The agent has a circular shape with dimension of 1 U_m. The simulator can be connected up to 3 agents at a given moment, but only one will be used in this thesis. Simulation time is measured in units of time (U_t) or cycles, being pre-configured to 1000, but it can be customised by the user. High or low walls in relation to the cheese represent obstacles. The state of the world offers noise and/or latency, meaning that the values captured by the sensors or actions sent to actuators may be inaccurate. Noise is an inconsistent reading that interferes in agent decisions. Latency is a delay of sensor readings or tasks implementations. For example, sensors of target area and compass have latency up to 4 U_t, receiving outdated information. There is a limit of up to 4 sensors for each 1 U_t to be implemented, except the obstacle sensor, which is always available. For specific details about simulator, simulation, and agent, please read [16].

6.1.1.1 Agent Specifications

The agent has a spheric shape with dimensions of 1 U_m of diameter. To navigate in the environment and reach the target area, the agent has sensors, engine and LEDs, respectively:

Obstacle Sensor : Three infrared sensors measure the distance between the agent and the obstacles around it. The values returned by the sensors are measured in $\frac{1}{distance}$. By default, the centre obstacle sensor is set on the central axis of the agent. The left and right obstacle sensors are set from -60 to 60 degrees of the centre obstacle sensor, respectively. Each sensor

covers an area of 60 degrees. The sensors provide an acceptable accuracy when the distance between the agent and the obstacle is less than 2 U_m;

Target Area Sensor : This omni directional sensor is set at the top of the agent. The sensor returns an angle between the target area and the axis of the agent. The sensor latency is set to 4 U_t, by default;

Ground Sensor : This sensor informs to simulator if the target area was reached;

Compass Sensor : This sensor of navigation measures an angle between the agent frontal axis and the north of the labyrinth;

Collision Sensor : This sensor detects a collision;

Engines : Right and left traction engines control the agent direction and speed of wheels. The engines are on their axis perpendicular to the frontal axis. The power of each engine varies between -0.15 and 0.15. The engines can produces a speed of up to $0.12\frac{U_m}{U_t}$. The power sent to the engines is influenced by noise, as well;

LEDs : Two LEDs indicate,

Visit : whether the target area was reached by the agent;

End : whether the simulation has finished.

6.1.2 Experimental Design

The architecture was shaped on the same principles of the generic agent architecture, presented in Chapter 5. Horizontal and vertical levels assembled by symbolic and connectionist modules suggesting an organisation capable of producing autonomous behaviours and able to emerge conscience[1]. Learning and reasoning features were encoded in rules to produce cognition/action. They harmoniously interchange signals among sensory and motor connections of internal agent modules. In the end, the agent introspectively evaluates and validates its thoughts in order to control the behaviour to a certain degree of freedom.

[1]Conscience is the part of architecture that transmits commands to the level beneath it.

Figure 6.1 presents the architecture to be evaluated. In essence, functions and modules are linked among them to interpret sensory stimulation and produce behaviour by actuators.

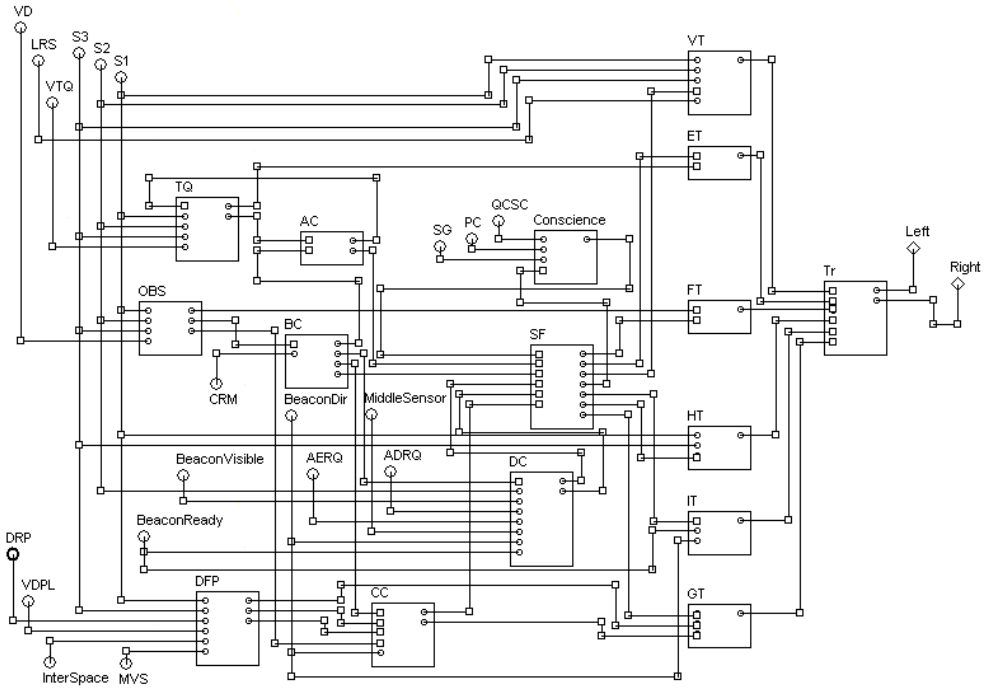

Figure 6.1: The Agent Control System.

Figure 6.1 shows a network of interconnected heterogeneous modules performing specific functions in parallel, as previously illustrated by a series of horizontal slices in Chapter 5. As with accommodation of multiple goals in parallel, each slice was explicitly implemented then tied then all together to form an harmonic Agent Control System. The architecture features are: low computational cost, high noise adaptation, modular, heterogeneity, bidirectional flow of information, biologic inspiration, robustness, flexibility, and the use of schemes of behaviour-based context.

In fact, the conjunction of heterogeneous AI approaches (symbolic/non symbolic) open the investigation to questions linked with agent behaviours analysis and digest.

6.1.2.1 Levels and Layers of the System

The architecture was modelled under two perspectives: logical and physical. The former distributes signals for four main hierarchic behaviour levels and one

meta-level, as follow: Stereotyped, Reactive, Instinctive, Deliberative, and Meta-management. The latter controls the agent system of values by arrangement of the abstraction levels in layers from simple to complex, such as: Traction Modules, Function Modules, Control Modules, Central Decision Module, and Conscience Module, respectively. Figure 6.2 and Figure 6.3 presents the arrangement of levels and layers in a stylised version, for easy undertanding.

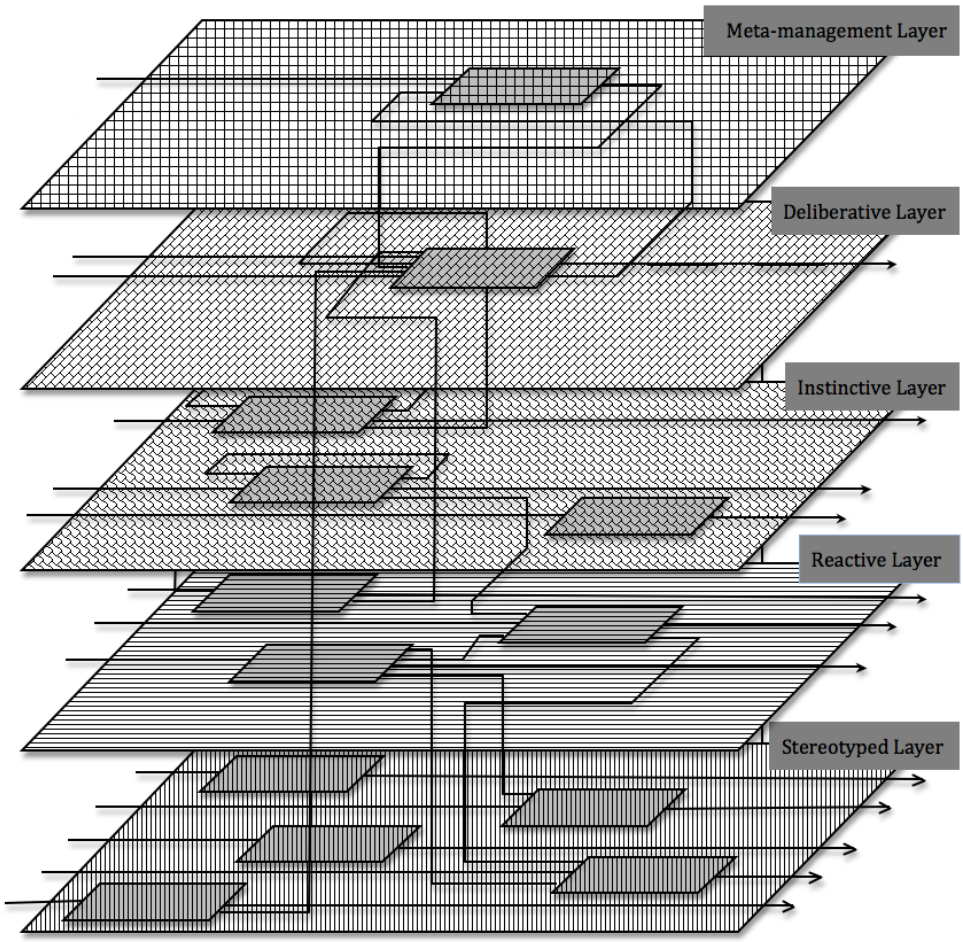

Figure 6.2: A stylised network of interconnected modules.

Figure 6.3 presents how the whole structure is arranged in order to judge and decide in favour of local sub-decisions and the main central decision to activate/inhibit the traction motors.

The Reactive layer, identified by light horizontal lines, implements *Inside Corner Detector Function Module (TQ)*, *Obstacle Detector Function Module (OBS)*, *Outside Corner Detector Function Module (DFP)*, and *Wander Traction Module (Vt)* with atomic rules encoded as production rules in the Stereotyped layer that control the agent locomotion, for instance. The Stereotyped layer, identified by

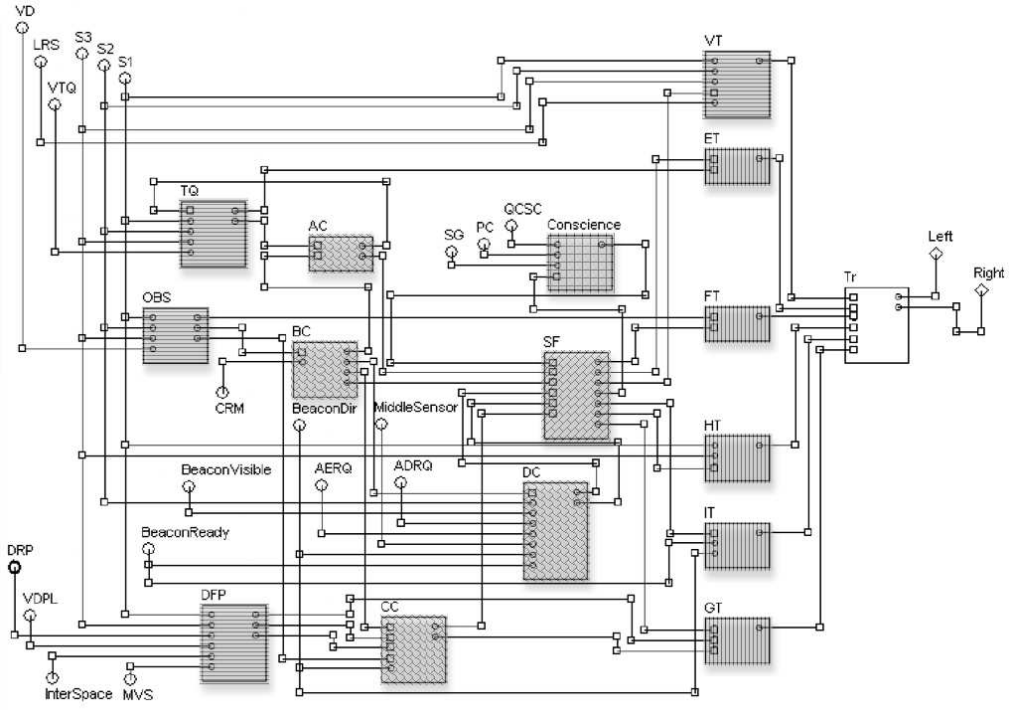

Figure 6.3: The Agent System diagram.

light vertical lines, the layer represents a Traction Module (*Tr*) that implement "toward" (reflexes) or "against" (taxies) actions in left and right actuators.

The Instinctive layer, determined by shingle lines, enables *Inside Corner Detector Control Module (AC)*, *Obstacle Avoidance Control Module (BC)*, *Circumvent Outside Corner Control Module (CC)*, and *Search Cheese Control Module (DC)*. Each instinctive module supports schemes of independent reactive modules. The use of Reactive Modules gives to the instinctive level a kind of "built-in knowledge" stored along agent evolution, that is represented by production rules or synaptic connection weights. These behaviour schemes are fired in accordance with the agent situation in the environment. In this level, Function Modules represent the schematic representation of perception and activity - signal detection and encoding. After data coming from the environment, Function Modules process them and send them already encoded to the level above to emerge instinctive actions for problem solving.

The Deliberative layer, identified by diagonal brick lines, determines the agent goals from a situation or stimuli. It is composed by a *Central Decision Making Unit (SF)* - reasoning of output process. This level measures the priority of all global objectives and triggers the correspondent control to solve the problem. The *Central Decision Making Unit (SF)* receive signals from Control Modules

spread in the structure.

The Meta-management layer, sighted by large grid lines, also known as module *Conscience*, represents the agent self control that analyses and judges the decisions made in Deliberative level by Central Decision Making Unit. The main goal is to autonomously suggests new control priorities to the Central Decision Making Unit by analysis of differences among a great variety of stimuli received from sensors to inner states and output of the *Central Decision Making Unit (SF)*. The module *Conscience* is based on a long-term knowledge base[2] stored in a repository of past "memories" - learning and external environment interaction. It was implemented to be a purposeful self reflection - introspection of decision activity. In the end, the overall activity of signals inside the architecture results on outcomes that will activate and/or inhibits the output of other modules.

6.1.2.2 The Priority Scheme

The priority scheme was implemented during analyses of Central Decision Making unit. In this priority-achieving scheme, the agent will decide in favour of the most current important task of a cached task-list to be executed. The level of importance of each task can be changed by intrinsic intentions, reading sensors, and conscience suggestions (perceptions of something).

Every module receive data in parallel and work in parallel with others of the same layer, but they obey a bidirectional degree of arrangement, as commented in Chapter 3. The levels of importance are arranged in execution time due to the situation and agent in context. By default, the agent issues are arranged in five main priority levels. The level of importance is from top to bottom, as follow:

1. Escaping of Inside Corner Traps;
2. Detecting and Avoiding Obstacles;
3. Circumventing Outside Corners;
4. Searching for Cheese;
5. Travelling Aimlessly in the Labyrinth.

6.1.3 Features of the Agent Architecture

The agent issues are distributed in the following modules: a module of Conscience in the meta-level, a central module of decision making in the deliberative level,

[2]Knowledge base is represented by rules encoded in a form of synaptic connection weights already fine-tuned - "encoded memory". It is used for the conscious/cognitive machinery to develop goals and perform multi-strategy plans.

four control modules in the instinctive level, three specialised functions modules in the reactive level, and six actuator modules in the stereotyped level. The respective modules are commented below obeying the priority scheme of arrangement.

6.1.3.1 Escaping of Inside Corner Traps

Basically, agents with simple behaviour (reflexive inner rules) are easily trapped in inside corners. To avoid or escape of inside corners, the agent needs to interpret correctly the signals received from all three obstacle sensors. Figure 6.4 demonstrates in four steps the action to escape of inside corners in the labyrinth.

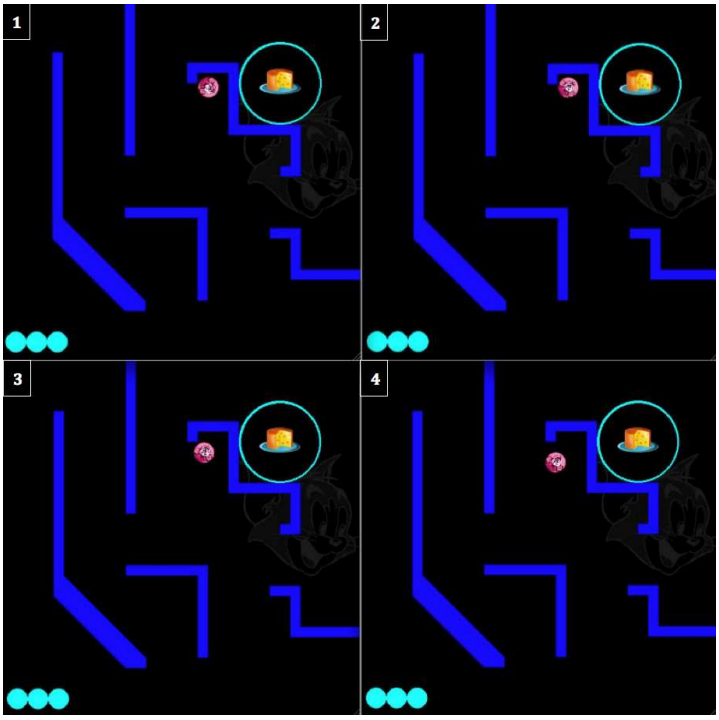

Figure 6.4: Escaping of inside corners.

The escape of inside corners behaviour is implemented in three modules, as follow, *Inside Corner Detector Function Module (TQ)*, *Inside Corner Detector Control Module (AC)*, and *Inside Corner Detector Traction Module (ET)*.

Inside Corner Detector Function Module (TQ)

The main function of *Inside Corner Detector Function Module (TQ)* is to check whether the agent is trapped in an inside corner, Figure 6.5.

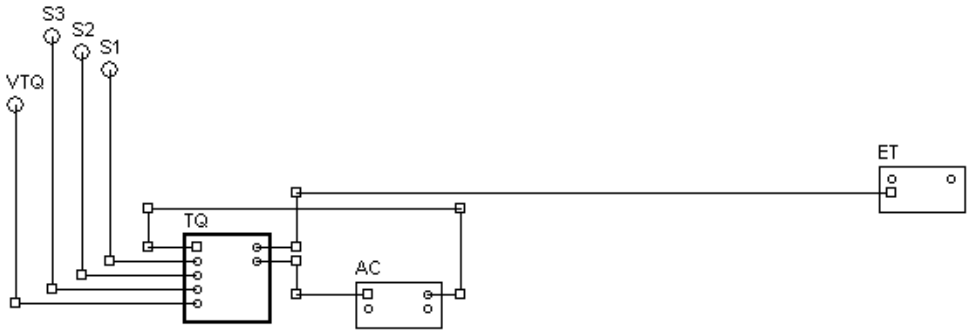

Figure 6.5: Circuit diagram of *Inside Corner Detector Function Module (TQ).*

This Function Module checks if the agent has moved out of an inside corner, based on output of *Escape Inside Corner Control Module (AC).* An inside corner is only detected whether left, middle and right obstacle sensors (S1, S2, and S3, respectively) had load, at the same time, a value higher than the default value set by user in distance vision architecture parameter (VTQ). In this case, the instinctive behaviour of basic survival is fired to discover some direction to get out.

If the left obstacle sensor had the lowest value among all obstacle sensors, but the left and right obstacle sensors are equal to and have the lowest value than right obstacle sensor, so the left agent side is chosen and sent to *Inside Corner Escape Traction Module (ET)*, and a number of times for the agent to turn around its own axis in anticlockwise direction is set by the user. During the cycle of rotation, the agent does not process any signal received by sensors until complete the task.

If the above conditions were not met, so the left obstacle sensor value is compared with the lowest value at time $t_{current}$, otherwise middle and right obstacle sensors have the same value, and if the value is less than the value load by left obstacle sensor, so the right side is the direction to escape of inside corner (clockwise direction). If any of these above conditions are met then the side to escape is randomly chosen.

Inside Corner Detector Control Module (AC)

This module decides the most promising strategy to move the agent out of the trap, based on reading sensors and inner agent states, Figure 6.6.

The *Inside Corner Detector Control Module (AC)* checks on time t_{-1} whether the agent was moving out of an inside corner. Maybe not, then *Inside Corner Detector Control Module (AC)* verifies the amount of steps that the agent rotated on its own axis, and sends it to the *Inside Corner Detector Function Module (TQ).*

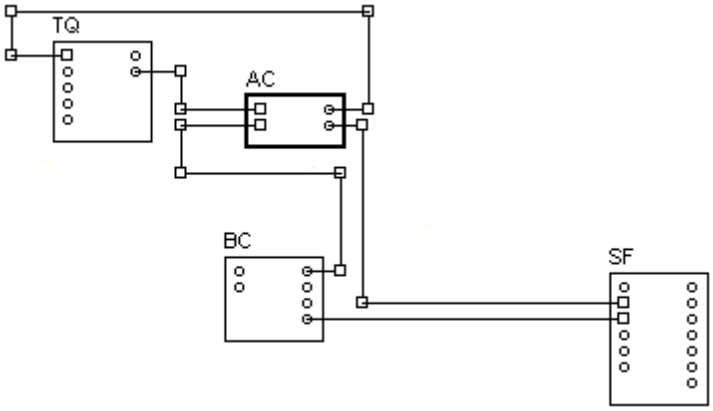

Figure 6.6: Circuit diagram of *Inside Corner Detector Control Module (AC).*

During the behaviour execution, the module decreases the number of steps by each U_t and requests execution priority of *Inside Corner Detector Traction Module (ET)* to *Central Decision Making Unit (SF)* in order to move agent out of the inside corner.

Inside Corner Detector Traction Module (ET)

This module implements in the agent the behaviour to escape of the trap. Figure 6.7.

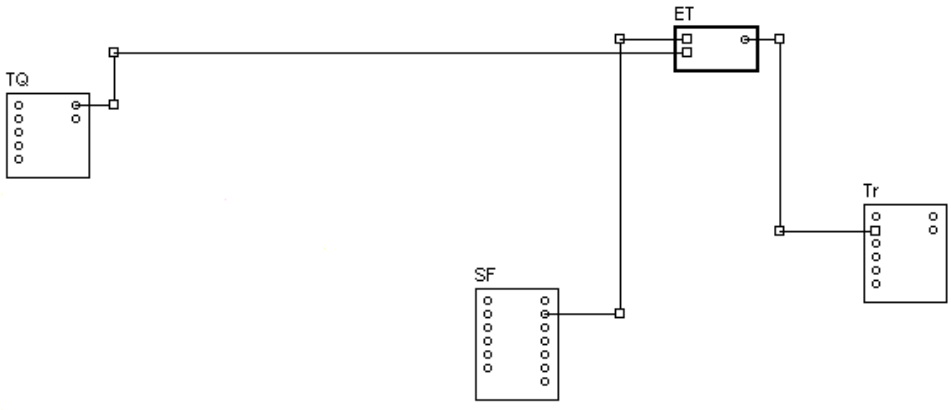

Figure 6.7: Circuit diagram of *Inside Corner Detector Traction Module (ET).*

This module receives the clockwise or anticlockwise agent direction from *Inside Corner Detector Function Module (TQ)*, and implements the order of *Central Decision Making Unit (SF).*

6.1.3.2 Detecting and Avoiding Obstacles

The agent performance is drastically reduced by collisions. To detect obstacles and avoid collisions, new guidance is suggested by the module to the actuator, as showed in four steps in Figure 6.8.

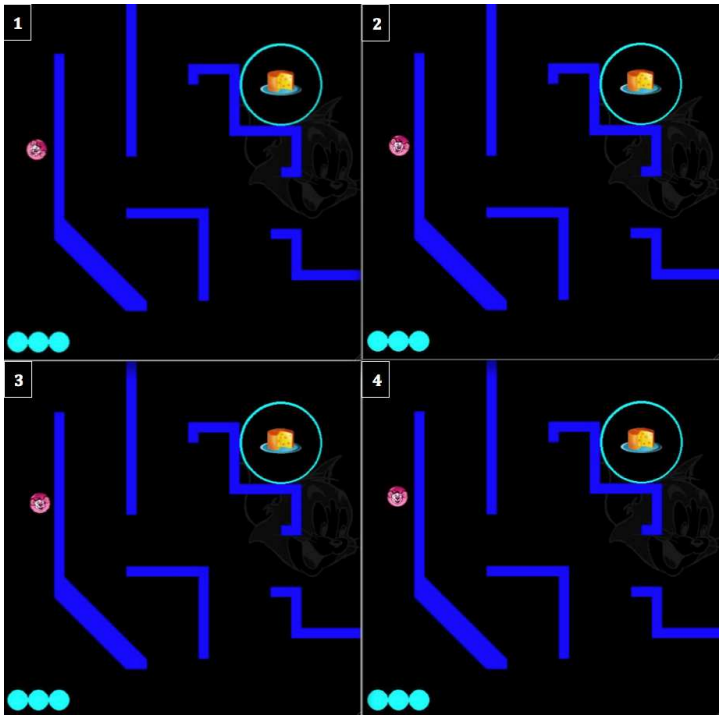

Figure 6.8: Avoiding collisions.

The avoidance obstacle planning is set in three modules, as presented below, *Obstacle Detector Function Module (OBS)*, *Obstacle Avoidance Control Module (BC)*, and *Obstacle Avoidance Traction Module (FT)*.

Obstacle Detector Function Module (OBS)

This module receives distance values from all obstacle sensors and interprets them to assert the presence of obstacles, Figure 6.9. The obstacles are only detected whether a reading value is greater than or equal to the architecture parameter of vision distance (VD), previously set by the user. The higher reading value means the closer the agent is to the obstacle.

The module checks and selects the sensor that reads the lowest distance between the agent and the obstacle, and suggests a new direction. The lower distance value means the opposite of distance load. The new direction chosen is

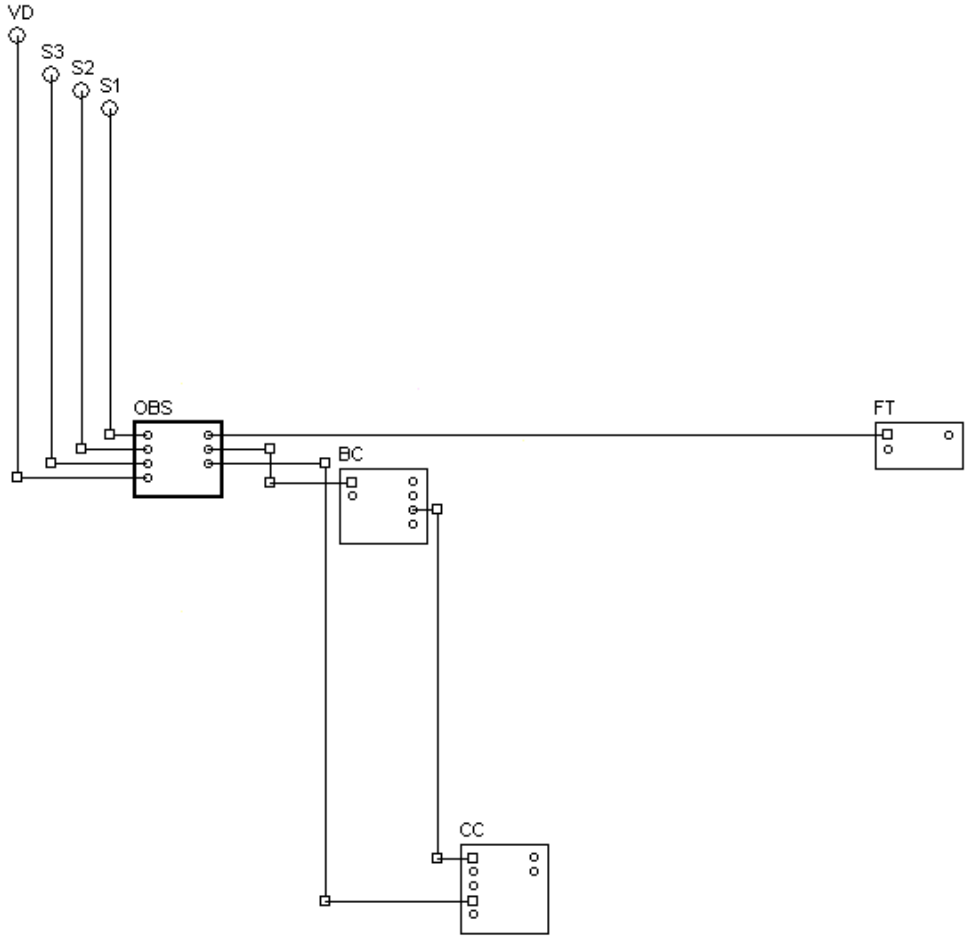

Figure 6.9: Circuit diagram of *Obstacle Detector Function Module (OBS).*

encoded as angle to be sent to *Obstacle Avoidance Traction Module (FT)* to implement the trajectory.

Obstacle Avoidance Control Module (BC)

This module decides in favour of the most promising strategy to avoid agent collision, Figure 6.1.3.2. All the strategies take into account the environment conditions and agent inner states.

Based on output of *Obstacle Detector Function Module (OBS)*, *Obstacle Avoidance Control Module (BC)*, and inhibitor agent behaviour architecture parameter (CRM),the agent requests execution priority for *Central Decision Making Unit (SF)* to trigger *Obstacle Avoidance Traction Module (FT).*

In certain situations, the path is blocked and the agent cannot reach the target area when detected. In this case, an inhibitor "hormone" is fired to block

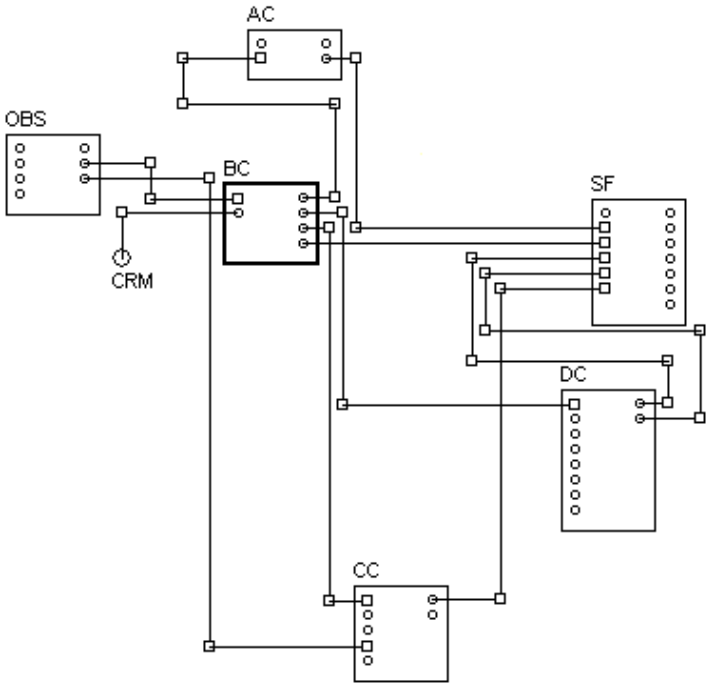

Figure 6.10: Circuit diagram of *Obstacle Avoidance Control Module (BC).*

the target area detection for a time period until it be in other place.

Obstacle Avoidance Traction Module (FT)

This module receives the obstacle direction from *Obstacle Detector Function Module (OBS)*, sets the opposite direction, and implements the order of *Central Decision Making Unit (SF)*, Figure 6.11.

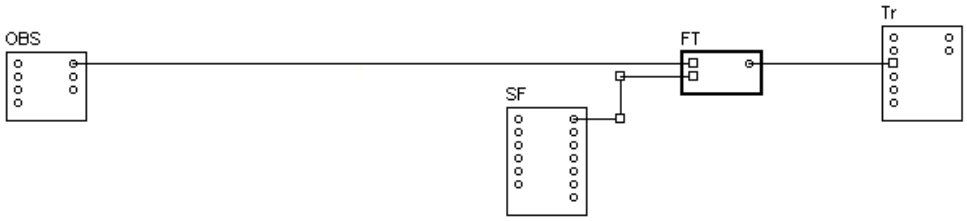

Figure 6.11: Circuit diagram of *Obstacle Avoidance Traction Module (FT).*

6.1.3.3 Circumventing Outside Corners

After observed it, it can be time consuming if thinking new strategies to control actuators for performing the best smooth curve. The solution proposed was to

built-in a natural behaviour in instinctive level that is capable of moving the agent from one side to other side, and making a curve. The coordinates of curve are automatically adjusted according to the wall corner shape. The result is the agent circumvents corners very fast and checks for obstacles that can put its performance at risk. Figure 6.12 presents the mouse using instinctive behaviours to circumvent the wall.

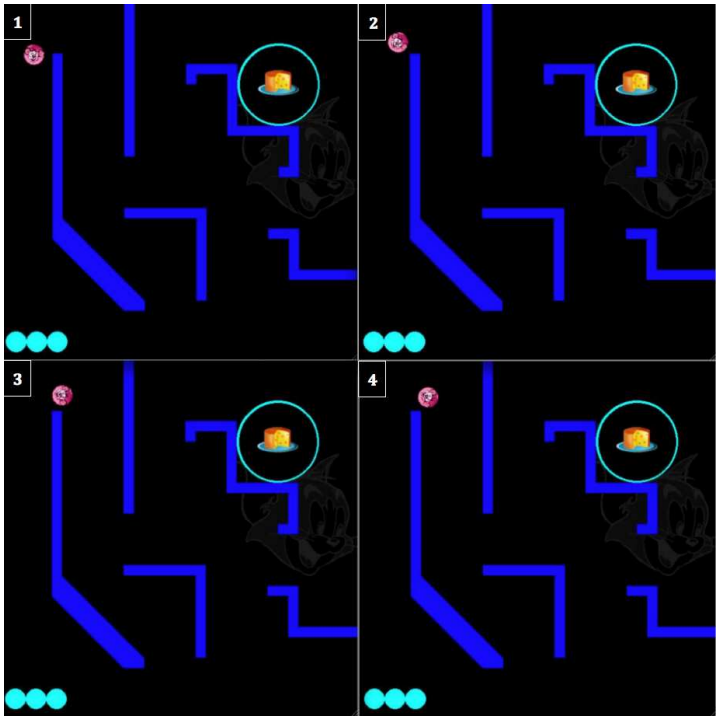

Figure 6.12: Circumventing the wall.

The behaviour presented above is presented below at three main modules as follow, *Outside Corner Detector Function Module (DFP)*, **Circumvent Outside Corner Control Module (CC)**, and *Circumvent Outside Corner Traction Module (GT)*.

Outside Corner Detector Function Module (DFP)

The main goal of this module is to detect the end of a wall and implement curves to circumvent corners in the labyrinth, Figure 6.13.

This module uses (DRP) architecture parameter to keep agent in a secure distance from lateral obstacle. After that, it checks whether value of left obstacle sensor (S1) is higher than value of right obstacle sensor (S3), and higher than distance view of wall (VDPL) in order to detect a corner or inter spaces, increment the agent left side counter, and reset the agent right side counter. But if the

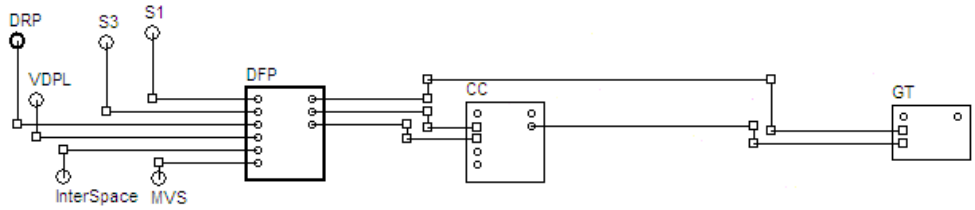

Figure 6.13: Circuit diagram of *Outside Corner Detector Function Module (DFP).*

condition was not met, the right obstacle sensor will be checked under the same conditions. If the condition is met by the right obstacle sensor, the right side counter is incremented and the left side counter is reset - these two conditions are checked using (MVS) set to 3 for previous information states at time t_{-3}, t_{-2}, and t_{-1} of each new simulation U_t. If some condition is met, the next action checks if the value of active sensor has returned a value lower than the secure distance of architecture parameter InterSpace. This verification is made in case of a slight distance from the wall; the agent does not detect it as end of wall, but continues implementing past behaviour. Conversely, if at time $t_{current}$ any of these conditions were not met, the counters of both sides are reset.

Circumvent Outside Corner Control Module (CC)

This module plans the best smooth curve to the agent if the end of a wall is detected, and sends a request to its actuator module by *Central Decision Making Unit (SF)*, Figure 6.14.

The module checks whether the agent is implementing the action of circumventing outside corner (requiring priority). Whether the condition was not met, the module checks whether the output of *Outside Corner Detector Function Module (DFP)* have detected the end of wall; additionally, the module *Circumvent Outside Corner Control Module (CC)* requires priority and starts circumventing the outside corner in a pre-fixed number of steps customised by user.

During the implementation of behaviour, the module uses BeaconDir simulator parameter to guide the agent to reach the cheese, but the environment is dangerous for the agent, so prevention decisions are built-in in its veins, such as: (a) do not circumvent walls to the side where there is no target area, (b) do not go directly to the target area, or (c) do not enforce a circumvent action if the target area is not accessible.

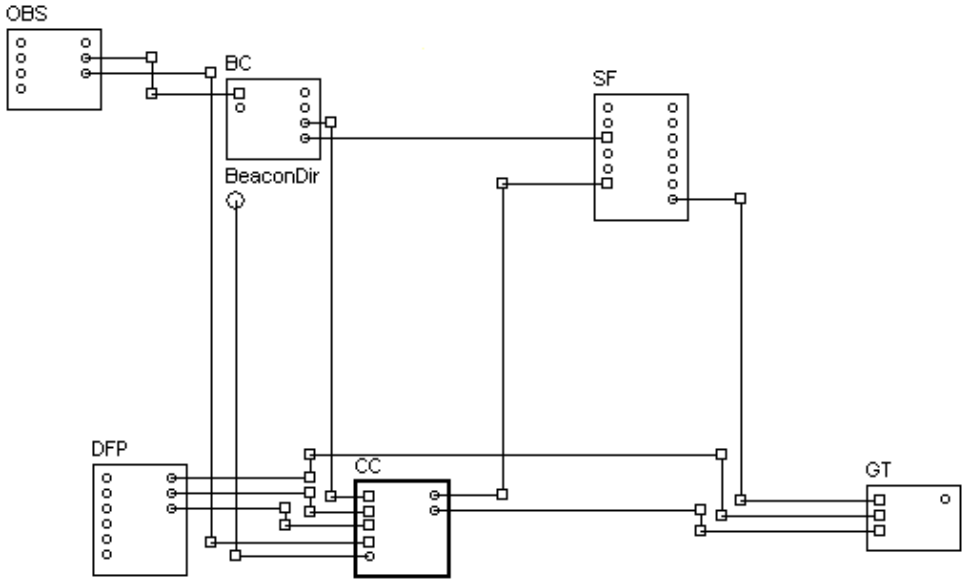

Figure 6.14: Circuit diagram of *Circumvent Outside Corner Control Module (CC).*

Circumvent Outside Corner Traction Module (GT)

This module receives the side of the wall and implements the action of circumventing outside corner, Figure 6.15. It happens at t_{-1} and $t_{current}$. At the time t_{-1} the agent moves forward, and at the time $t_{current}$ the agent turns a little to the side that circumvents the outside corner.

6.1.3.4 Searching for Cheese

The main function of this module is to guide the agent to reach the target area. Figure 6.16 presents in four steps the mouse reaching the cheese.

Nonetheless, the architecture may interrupt this particular inner stimulus for a short time period if a translucent wall and target area are both detected on the same direction. The purpose is to avoid iterative crashes like a bug trying to pass through the glass.

Once the agent is partially blind, it does not recognise the target area like an external stimulus of attraction, thus other modules can control the agent to successfully avoid obstacles. These behaviours are represented by *Search Cheese Control Module (DC)*, *Search Cheese Traction Module (IT)*, and *Best Side Traction Module (Tr).*

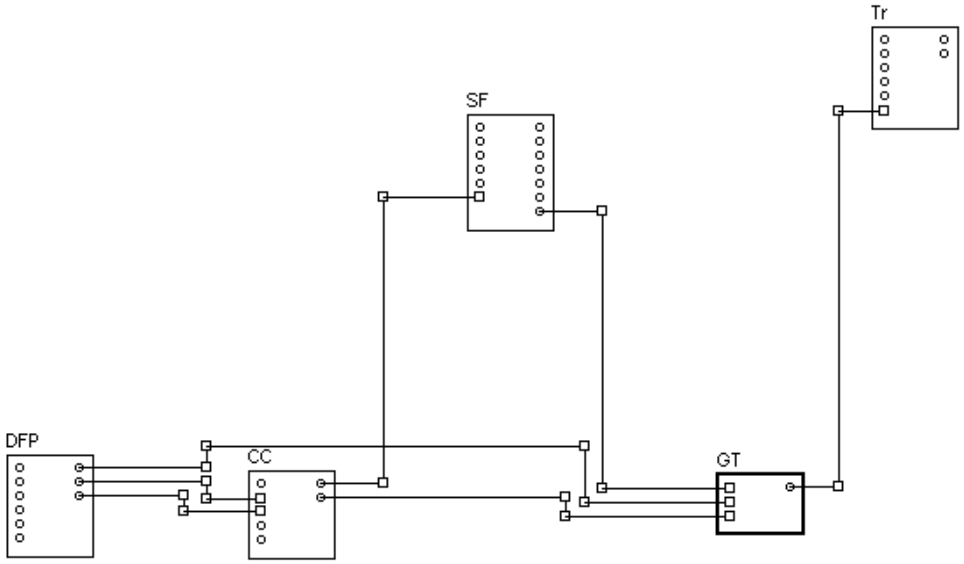

Figure 6.15: Circuit diagram of *Circumvent Outside Corner Traction Module (GT).*

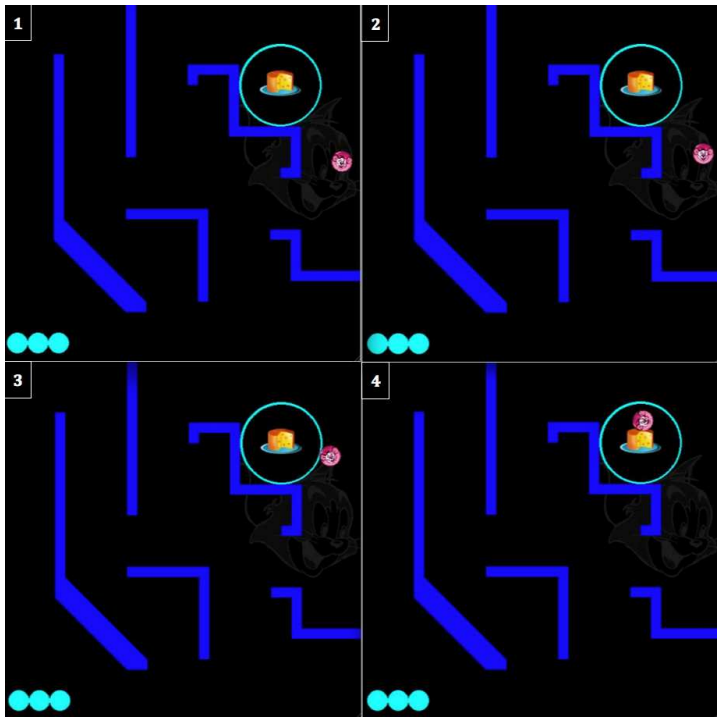

Figure 6.16: Reaching the cheese.

Search Cheese Control Module (DC)

This module receives from the simulator the following parameters: obstacle sensor (S2); target area visibility (BeaconVisible); target area direction (BeaconDir); target area was detected in that cycle (BeaconReady); the output of *Obstacle Avoidance Control Module (BC)* that blinds the agent for a time period; and the target area detector(MiddleSensor), as presented in Figure 6.17.

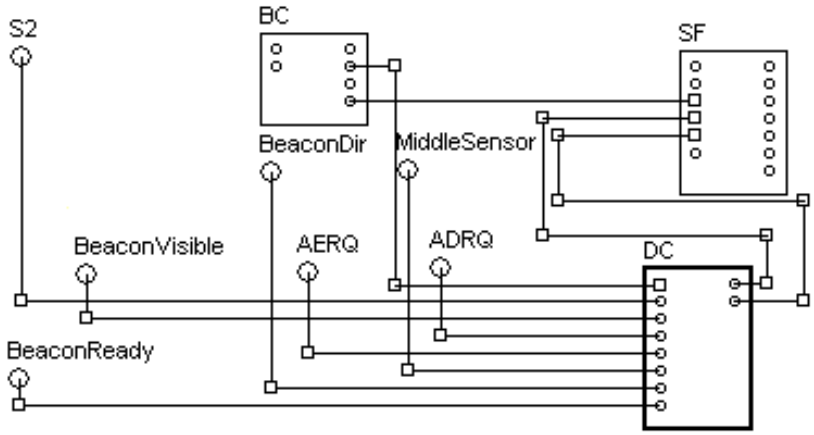

Figure 6.17: Circuit diagram of *Search Cheese Control Module (DC).*

This module uses two output bits to connect and control the *Best Side Traction Module (Ht)*, and *Search Cheese Traction Module (IT)*. The procedure verifies if the target area was found and the agent is not blind. If so, the next action is to check if the path to the target area is accessible.

Search Cheese Traction Module (IT)

This module guides the agent to the target area and fixes the path planning trajectory caused by the interference of sensor latency, Figure 6.18.

The module receives the following parameter data: from *Central Decision Making Unit (SF)*, the data that indicates whether the sensor implemented any measurements; from simulator, if target area was detected (BeaconReady); also receives the direction of the target area (BeaconDir). The output of the module is an angle that fixes the agent trajectory to the target area.

Best Side Traction Module (Tr)

This module is activated if a low obstacle was detected between the agent and the target area. The *Best Side Traction Module (Tr)* receives values from left and right obstacle sensors and sends an angle to the engine, Figure 6.19.

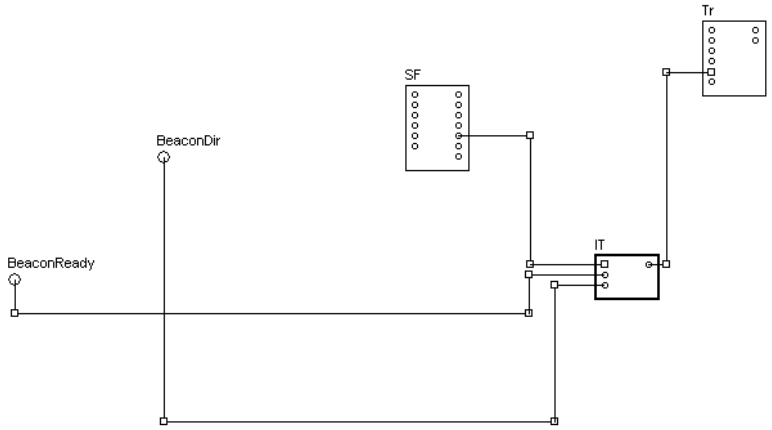

Figure 6.18: Circuit diagram of *Search Cheese Traction Module (IT).*

6.1.3.5 Travelling Aimlessly in the Labyrinth

The purpose of this module is to implement a simple wander behaviour in the environment when no other behaviour was implemented, Figure 6.20.

Wander Traction Module (Vt)

This simple behaviour has a built-in subroutine implemented that uses $t_{current}$ and correction value function (LRS), both hand crafted by user, to treat noises and latency, Figure 6.21.

6.1.3.6 Central Decision Making Unit

The *Central Decision Making Unit (SF)* is responsible for the reasoning of the agent. Its main goals are measuring the priority of all global objectives and trigger the correspondent control to solve the problem. The module receives signals from all control modules arranged in the structure, that also includes *Conscience Module (Conscience)*, Figure 6.22.

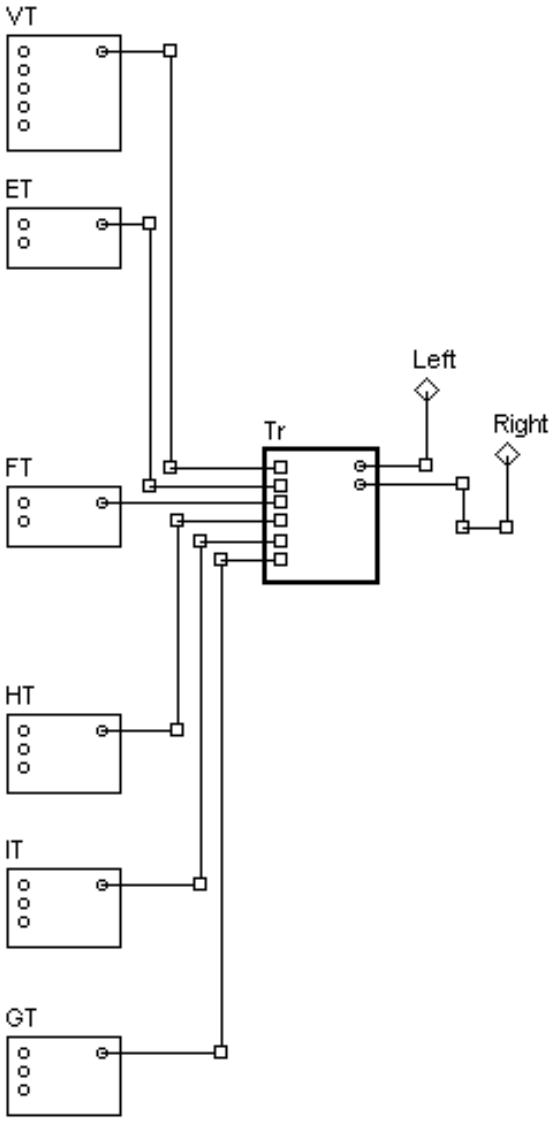

Figure 6.19: Circuit diagram of *Best Side Traction Module (Tr).*

6.1.3.7 Conscience

In general, conscience is correlated with standardised decisions rules (agent body intentions), and with "psychical" activities in the alternate states of long-term memories.

Conscience Module (Conscience)

The *Conscience Module (Conscience)* was implemented to monitor and think about decisions made by *Central Decision Making Unit (SF)*, influencing it

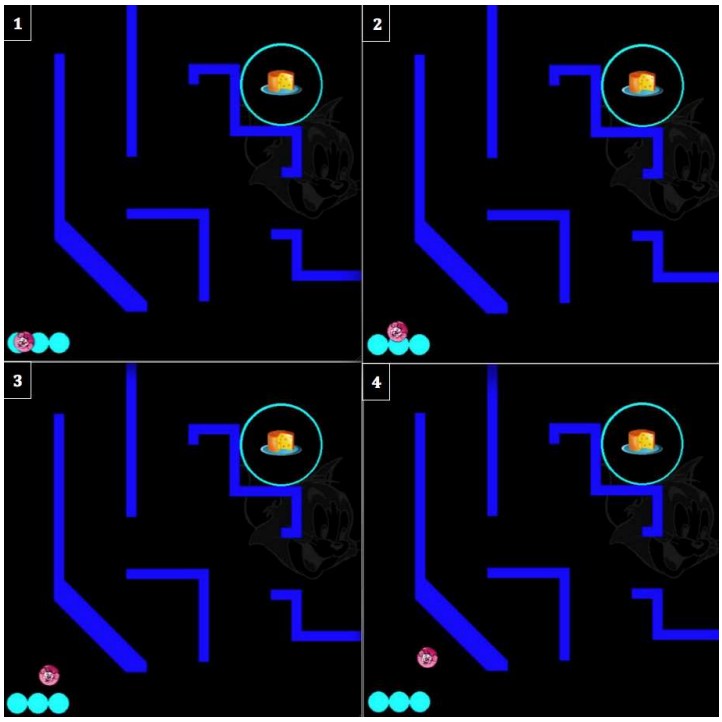

Figure 6.20: A simple Wander behaviour.

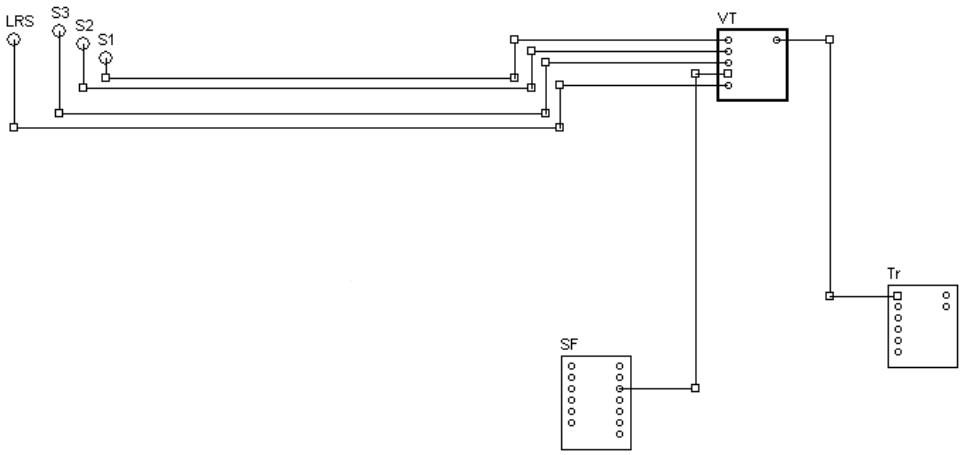

Figure 6.21: Circuit diagram of *Wander Traction Module (Vt).*

by approbation or rejection. Thus, *Conscience Module (Conscience)* intercepts the output of *Central Decision Making Unit (SF)* and makes judgements based on the past experience load.

The process is as follow. The *Conscience Module (Conscience)* intercepts the *Central Decision Making Unit (SF)* output value and brings it to itself. Inside the *Conscience Module (Conscience)*, Artificial Neural Networks will verify: a)

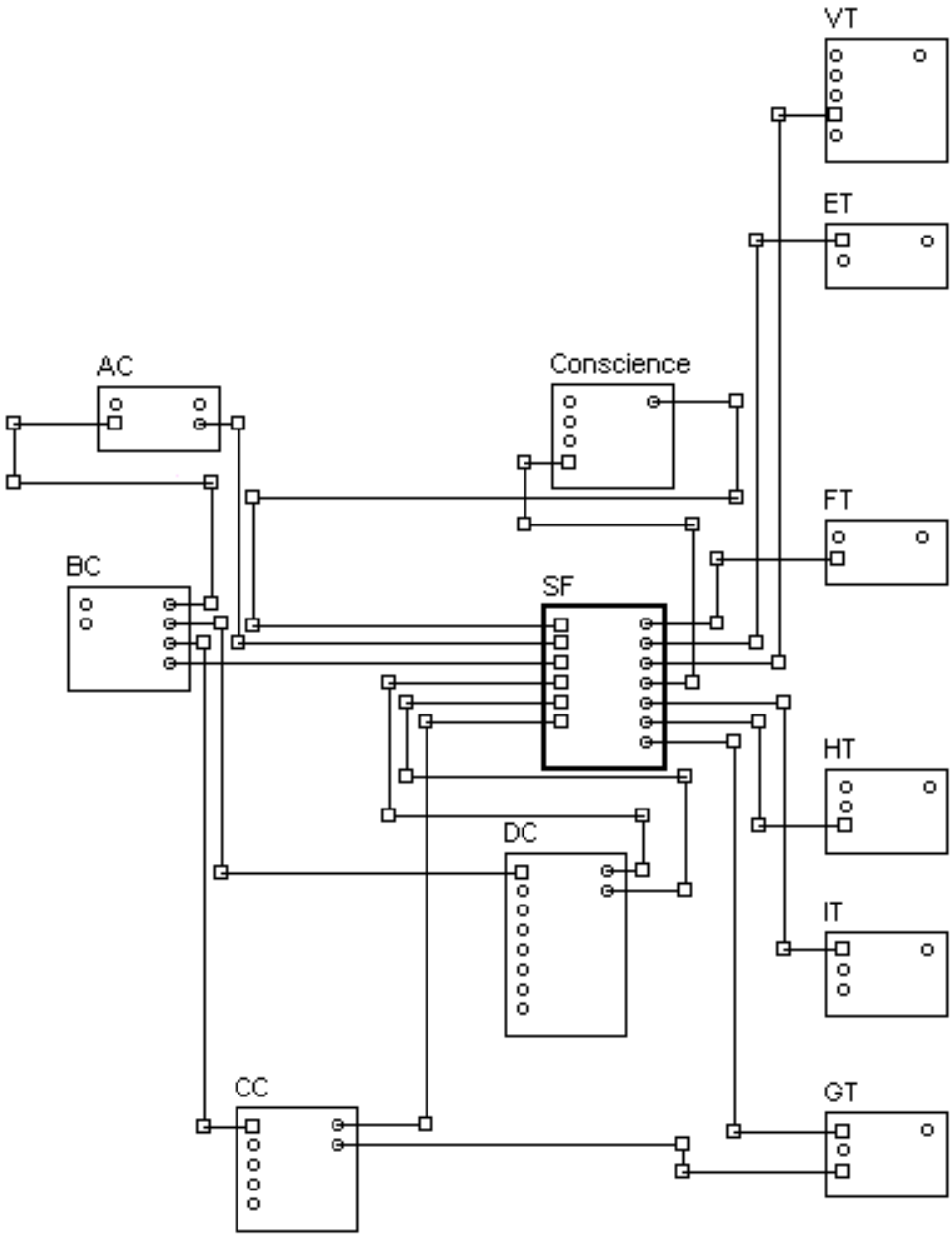

Figure 6.22: Circuit diagram of *Central Decision Module (SF).*

the output of *Central Decision Making Unit (SF)* and confront it with a data set of past experiences (PC)[3]; b) the input signals of collected data during the current agent performance (SG)[4], and c) how time cycle in past the ANN should look at (QCSC). Experiences are encoded during the agent training in a file of

[3]In this experiment, the data set was created during the *Wander Module* when running.

[4]The input examples represents the whole history of all sensors, all architecture parameters, all outputs of control modules, and the internal state of agent.

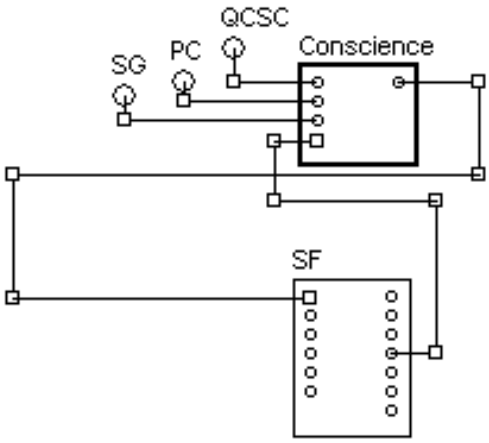

Figure 6.23: Circuit diagram of *Conscience Module (Conscience).*

synaptic weights. Such synaptic weights have been fine-tuned when observing the agent wandering behaviour and its integration with the system of searching for target areas. After that, ANN returns a vector of six cells that contains new priority weights to the *Conscience Module (Conscience).* The *Central Decision Making Unit (SF)* will check again and verify if the past decision was the best to that moment. This execution happens in one cycle of simulation and before the execution order of any traction modules. The priority weights will influence the output values of control modules previously load, respectively, the *Inside Corner Detector Control Module (AC)*, the *Obstacle Avoidance Control Module (BC)*, the *Circumvent Outside Corner Control Module (CC)*, two cells for the *Search Cheese Control Module (DC)*, and the *Wander Traction Module (Vt).* In addition, as what matters to the consciousness level is the majority activation of only one output Control Module, then an average of outputs is re-entered to the *Central Decision Making Unit (SF).* The *Central Decision Making Unit (SF)* receives the output of *Conscience Module (Conscience)*, and sums it with its output. *Wander Traction Module (Vt)* is the unique module that does not have its own control module, so the sum represents the following steps: If the output of *Conscience Module (Conscience)* produced to *Wander Traction Module (Vt)* is higher than value `0.95`, then the value 1 is summed to the *Conscience Module (Conscience)* output value of *Wander Traction Module (Vt)*, otherwise the outcome of sum of *Wander Traction Module (Vt)* receives only the output of Conscience of *Wander Traction Module (Vt).* Therefore, *Central Module (SF)* makes calculus between theses sums to know who has the highest degree to prioritise the activation of this respective module in its output.

6.1.4 Experiments

About 17 original scenarios with initial positions to mouse and cheese were generated to produce a pool of new scenarios with different initial positions. The unknown scenarios represent the impartiality of techniques. The mix of scenarios produced new 170 valid scenarios with a non repetitive initial random positions. From those scenarios, only 10 and non repetitive scenarios were randomly chosen to be used on each evaluation phase. To be closer of a reproducible result, the Table 6.1 presents the parameters of simulator used to configure the environment:

Parameters	**Values**
SimTime	1000
CycleTime	60
CompassNoise	2
BeaconNoise	2
ObstacleNoise	0.1
MotorsNoise	1.5
RunningTimeout	500
GPS	OFF
ScoreSensor	OFF
ShowActions	FALSE
NBeacons	1
RequestsPerCycle	4
ObstacleRequestable	ON
BeaconRequestable	ON
GroundRequestable	ON
CompassRequestable	ON
CollisionRequestable	ON
ObstacleLatency	1
BeaconLatency	5
GroundLatency	1
CompassLatency	5
CollisionLatency	1
BeaconAperture	3.141593

Table 6.1: Simulator Parameters.

Three main sets of experiments were devised in order to produce a fair comparison. First we use three Reactive Modules **G1** with different approaches to produce the *base-line* results, the results with which our proposed techniques will be compared. The second experiment uses Architecture without Conscience Module **G2** in 12 different test phases. The last experiment uses Architecture with Conscience Module **G3** to judge and suggest new answers to the system in 24 different test phases. Each test phase represents inner evolution of agent structure to be tested in 10 runs. To each new run, a

respective and non-repetitive scenario is load to the simulation. The stop value measure of all scenarios was customised to 1000 U_t. If in this time period the mouse has reached the cheese, the arrival time is set to 1000, by default. In the end, an amount of 390 runs were performed.

In the first experiment, entitled Reactive Module, we test *Wander Traction Module (Vt)* under three different reactive techniques to evaluate the agent performance. To the first technique, Rules: Hand crafted, we have hardly experienced to develop segments of knowledge by hand. The structure of production rules composed a small expert system that reacts to events of the environment. Additionally, the second and third techniques use ANN to prepare the agent to reach its goal by synaptic connection weights, instead of production rules. To the second technique, titled ANN: GA-based wrapper, ANN was tuned using a GA-based *wrapper*. GA set ANN to: back-propagation learning algorithm; random weights initialised from $[-0.5, +0.5]$; until three times more neurones in the hidden layer than the input layer; sigmoidal transfer functions set to hidden and output layers, and bias set to value 1; one neurone in the output layer, and bias set to value 1; learning rate, momentum rate and steepness rate set to respective default internal ranges; stop the training phase when the error rate achieves 0.05 in test phase or the training epochs achieves 150. Last, the third technique, called ANN: Hand-tuned, represents ANN parameters hand-tuned by the supervisor. To this technique, we hand set the ANN to: three layers; back-propagation learning algorithm; random weights initialisation from $[-0.5, +0.5]$; three neurones set to the input layer, which receives three obstacle sensors; fifteen neurones set to hidden layer, and bias set to value 1; one neurone set to output layer to feed the traction module, and bias set to value 1; sigmoidal transfer function set to hidden and output layers; learning rate, momentum rate and steepness rate set to 0.8, 0.5 and 1, respectively; stop the training phase when the error rate gets below 0.05 in test phase or the training epochs reaches 150. The first experiment diagram is represented in Figure 6.24 with three different reactive techniques. In the diagram, the main box on the left side represents the experiment, and the three sub-boxes on the right side represents the specific tests supported by the experiment.

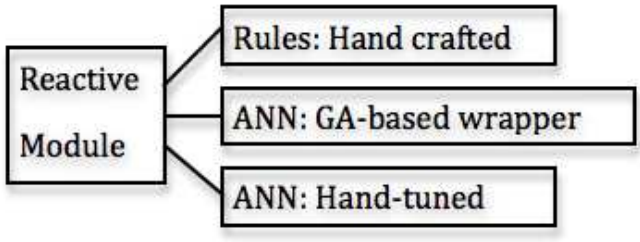

Figure 6.24: The first experiment diagram.

In the second experiment, denominated Architecture without Conscience Module, the *Central Decision Making Unit* filters various Control Modules request, determines the next agent action, and subsumes orders of execution/inhibition to Traction Modules. Moreover, a script file composed by all configurable agent parameters that feed the Function and Control Modules - made by production rules in whole structure. This script is responsible to particularise the agent behaviours and influence the agent decisions. In this experiment, the values are hand-tuned by a supervisor or automatically wrapped by GA; respectively, Script Architecture: Hand-tuned, and Script Architecture: GA-based Wrapper. In both techniques, the presence or absence of Previous Information States determine the agent performance. Previous information states is a matrix of memory cells that stores multiple reading signals and inner agent parameters and states by time-sequential. The main purpose is to use possible differences between the time-sequential to influence the agent decision. Other techniques and their configuration have been inherit from the first experiment. The second experiment is represented in Figure 6.25 by four horizontal branches. Each full branch of the diagram supports up to 3 different techniques, since each Reactive Module inherited three different reactive techniques. Each branch, from left to right represents the group that supports the specific tests.

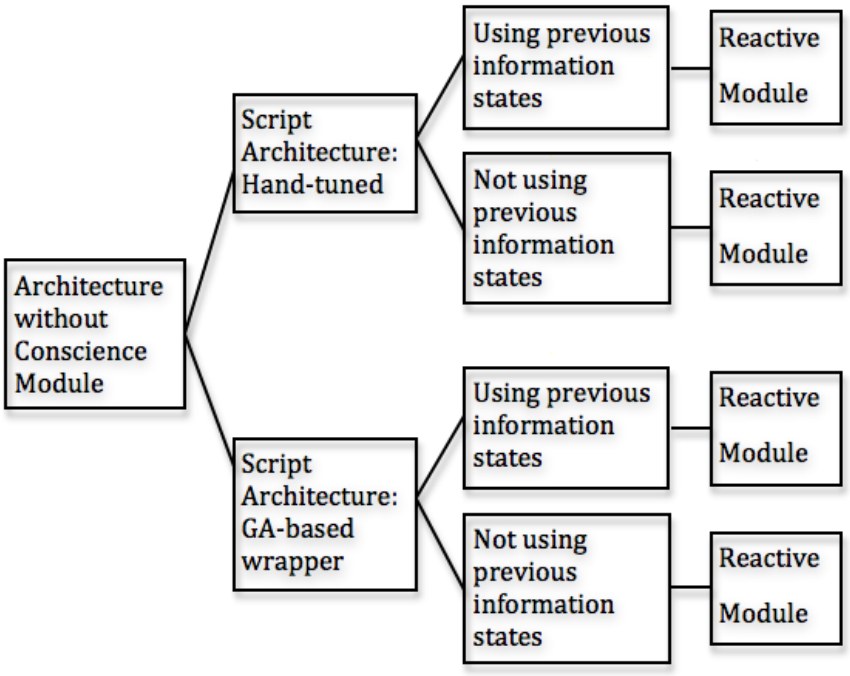

Figure 6.25: The second experiment diagram.

In the last third experiment, termed Architecture with Conscience Module,

the *Conscience Module (Conscience)* is used to reduce drastically the agent arrival time. The module analyses all sensors, and inner information states of agent and suggests new results. The reader can imagine the Conscience Module suggestion like an inner voice of agent mind saying which would be the best decision to follow in that moment. The Conscience Module machinery uses ANN, which is hand-tuned by a supervisor or automatically wrapped by stochastic search that GA offers, respectively, ANN Conscience Module: Hand-tuned, ANN Conscience Module: GA-based wrapper. It is worth noted that ANN used in *Conscience Module (Conscience)* is adjusted obeying the same principles and values used to configure *Wander Traction Module (Vt)* commented on the first experiment, excepted by 29 input neurones, 35 hidden neurones and one output neurone. Both robust methods inherit all techniques of the second experiment, which when combined they produce 24 test phases, differently from the second one that produces 12 test phases and the fist one, which produces three test phases. The third experiment is represented in Figure 6.26 that has two branches. Each branch inherits 12 test phases from second experiment. The difference between branches is the customisation of Conscience Module. After all, 24 test phases are run. Each branch, from left to right represents the group that supports the subgroups.

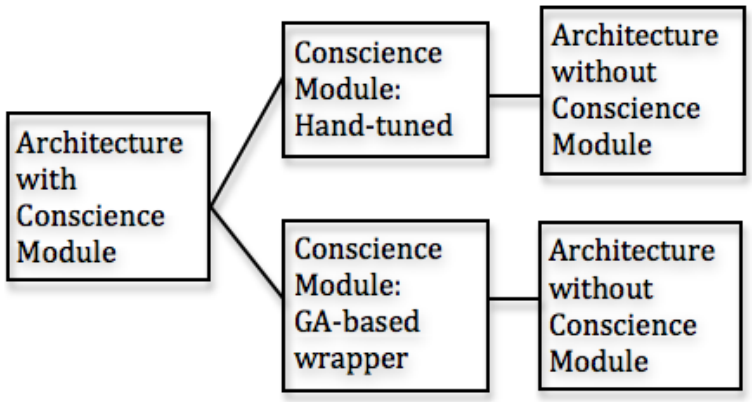

Figure 6.26: The third experiment diagram.

6.1.5 Experimental Results

The results of three main experimental groups G1, G2, and G3 are reported below from Table 6.2 to 6.4. Table 6.2 presents Reactive Module / ANN: GA-based wrapper, signalised by an asterisk mark, as the winner technique. In Table 6.3, the best technique is Architecture without Conscience Module / Script Architecture: GA-based wrapper / Using previous information states / Reactive Modules / ANN: GA-based wrapper, indicated by an asterisk mark; in addition, Table 6.4

presents Architecture with Conscience Module / ANN Conscience Module: GA-based wrapper / Script Architecture: GA-based wrapper / Using previous information states / Reactive Modules / Rules: Hand crafted, showed by an asterisk mark, as the best technique.

Analysis of Variance (ANOVA) and Tukey Test were used to identify statistical significance among experimental groups and their correspondent techniques. The winner technique of each main experiment is highlighted by an asterisk mark set on the right side of the std. deviation value. Figure 6.27 presents the Analysis of Variance (ANOVA) among the winner techniques. According to ANOVA, where $*$ achieved a $p < 0.05$, means all experiments are different among them. The labels are: dark grey colour representing G1 winner, light grey colour showing G2 winner, and black colour representing G3 winner, and Figure 6.28 presents the Tukey Test, where $*$ achieved a $p < 0.05$ between G1 and G2, and G1 and G3, but G2 and G3 were not statistically different. The labels are: dark grey colour representing G1 winner, light grey colour showing G2 winner, and black colour representing G3 winner.

From Figures 6.27 and 6.28, the following conclusions can be drawn. First, the 39 techniques listed in this study showed good results, reporting that arrival time decreased drastically during the evolution of the architecture.

Second, as it can be observed in Figures 6.27 and 6.28, statistical tests reported significant differences between winner techniques on experiment I and II, and reported on experiment I and III, but not too different between those reported on experiment II and III. Nevertheless, the winner technique in experiment III had the lowest variability of data around the average (high confidence level), if compared to the winner technique of experiment I and II.

Group	**Technique**	$\bar{x} \pm s$
Reactive Module	Rules: Hand crafted	898.57 ± 259.26
	ANN: GA-based wrapper	778.40 ± 300.64*
	ANN: Hand-tuned	898.57 ± 196.03

Table 6.2: Table results of Reactive Module experiment.

Groups & Techniques mixed					$\bar{x} \pm s$
ANN without Conscience Module	Script Architecture: Hand-tuned	Not Using Previous Inform. States	Reactive Module	Rules: Hand crafted	601.00 ± 406.52
				ANN: GA-based wrapper	569.80 ± 370.22
				ANN: Hand-tuned	583.03 ± 378.06
		Using Previous Inform. States	Reactive Module	Rules: Hand crafted	504.70 ± 366.44
				ANN: GA-based wrapper	490.30 ± 337.74
				ANN: Hand-tuned	496.60 ± 374.41
	Script Architecture: GA-based wrapper	Not Using Previous Inform. States	Reactive Module	Rules: Hand crafted	543.40 ± 356.98
				ANN: GA-based wrapper	526.60 ± 345.33
				ANN: Hand-tuned	546.70 ± 325.56
		Using Previous Inform. States	Reactive Module	Rules: Hand crafted	517.32 ± 316.83
				ANN: GA-based wrapper	449.20 ± 257.23*
				ANN: Hand-tuned	475.00 ± 239.57

Table 6.3: Table results of ANN without Conscience Module experiment.

Groups & Techniques mixed						$\bar{x} \pm s$
Architecture with Conscience Module	ANN Conscience Module: Hand-tuned	Script Architecture: Hand-tuned	Not Using Previous Inform. States	Reactive Module	Rules: Hand crafted	367.00 ± 249.06
					ANN: GA-based wrapper	451.10 ± 277.95
					ANN: Hand-tuned	449.50 ± 320.36
			Using Previous Inform. States	Reactive Module	Rules: Hand crafted	336.10 ± 263.04
					ANN: GA-based wrapper	388.60 ± 245.82
					ANN: Hand-tuned	387.12 ± 261.19
		Script Architecture: GA-based wrapper	Not Using Previous Inform. States	Reactive Module	Rules: Hand crafted	385.60 ± 298.35
					ANN: GA-based wrapper	458.20 ± 284.05
					ANN: Hand-tuned	487.30 ± 313.47
			Using Previous Inform. States	Reactive Module	Rules: Hand crafted	462.70 ± 323.36
					ANN: GA-based wrapper	447.11 ± 277.27
					ANN: Hand-tuned	451.90 ± 321.66
	ANN Conscience Module: GA-based wrapper	Script Architecture: Hand-tuned	Not Using Previous Inform. States	Reactive Module	Rules: Hand crafted	349.60 ± 237.81
					ANN: GA-based wrapper	443.80 ± 274.05
					ANN: Hand-tuned	331.30 ± 267.93
			Using Previous Inform. States	Reactive Module	Rules: Hand crafted	375.70 ± 292.24
					ANN: GA-based wrapper	481.10 ± 315.82
					ANN: Hand-tuned	463.00 ± 330.16
		Script Architecture: GA-based wrapper	Not Using Previous Inform. States	Reactive Module	Rules: Hand crafted	379.92 ± 275.78
					ANN: GA-based wrapper	428.20 ± 268.70
					ANN: Hand-tuned	466.60 ± 318.13
			Using Previous Inform. States	Reactive Module	Rules: Hand crafted	290.20 ± 182.49*
					ANN: GA-based wrapper	335.80 ± 153.26
					ANN: Hand-tuned	385.00 ± 261.95

Table 6.4: Table results of ANN with Conscience Module experiment.

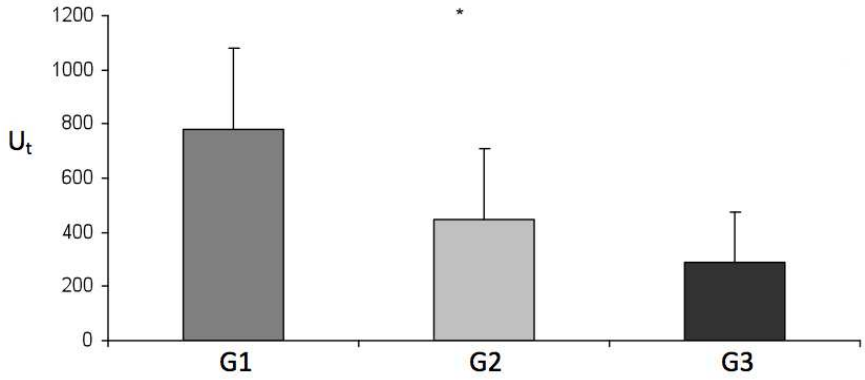

Figure 6.27: The results of Analysis of Variance (ANOVA).

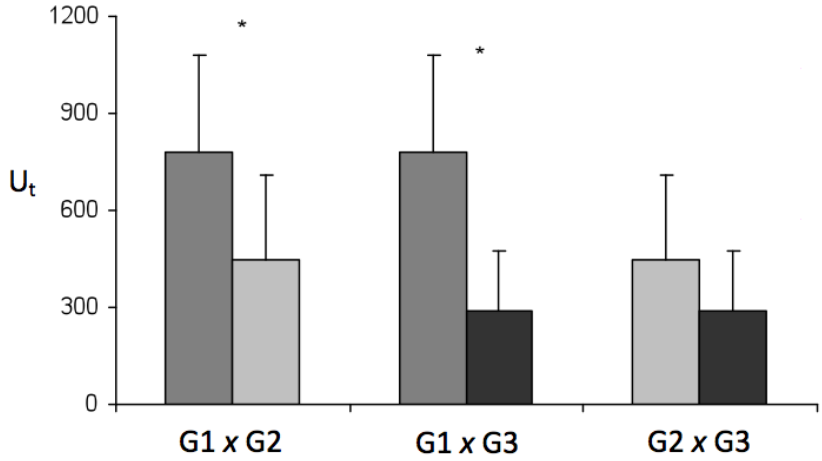

Figure 6.28: The results of Tukey Test.

Last, the harmony among heterogeneous approaches interchanging messages was fully aimed and was decisive for the agent to reach the target area. The use of previous information states, and past events stored up in *Conscience Module (Conscience)*, had rightly influenced all agent decisions, and demonstrated that using reactive rules hand-crafted in the lowest level was the most feasible and reliable way to quickly reach the goal in any scenario tested.

6.2 Conclusions

In this chapter, different autonomous structures made of individual or collective behaviours were exhaustively evaluated and compared. The main purpose was to find an efficient technique able to reach fast agent performance (short arrival time). Our proposed and winner architecture was a system of logic and rules,

stimulated by goals, linked by channels of communications, and shaped on biologic evolutionary principles. Its hybrid structure of interconnected modules supported perception, movement, cognition, and memory.

We observed that the winner architecture achieved an hybrid evolutionary configurable strategy by mixing approaches, reaching the equilibrium or balance between elements of agent intellectual faculty and simple propensities. By using previous information states - registered changes in time of inner states, physical activities, mental achievements, and conscience achievements - the agent travelled backwards and forwards in time, projecting itself in future. Nevertheless, we detected that in some simulations, the agent behaved "impatiently" when future time brought signals back to the current time, so every moment was a struggle. This particular issue was fixed since the blind mouse agent function was implemented - the architecture interrupts the behaviour for a short time period if a low wall and target area are both detected on the same direction.

We also observed that current and past events, or low and high levels had disputed strengths at each renewable U_t. In lower level, react to the environment was necessary in such situations. For instance, in face of extreme circumstances, such as avoiding collisions, the lower behaviour level was "what" had the agent control. The innate principles were not dependent of consciousness to be operated. Conversely, superior levels were responsible to "think" or find an answer to solve the problem. The upper levels seemed to be flexible and capable of changes in behaviour due to learning in past and swapping of long-term experiences in present to deal with an unpredictable events in future. Therefore, the environment changes was quickly assimilated by agent. Conversely, other techniques used ostensible knowledge that was divided in trivial disconnected fragments or was limited to specialised knowledge, and they did not reached the aim so fast.

On one hand, the arrival time was the unique evaluation parameter to discover the most promising technique among experiments that could be measured by simulator. On the other hand, the same parameter did not reflect all agent "creativities" during the simulations, the Artificial Intelligence main point. Having consciousness, the agent balanced, calculated, compared and selected what was the most important thing to do. Additionally, GA and *Conscience Module (Conscience)* were crucial to overpass latency and noise, reducing the agent arrival time due to the use of past knowledge, well-done decisions, and a fine-tuned behaviour. *Conscience Module (Conscience)* avoided robotisation of activities, and GA made reflexions more robust - repertory enlarged and sensibility accurated. Other point is even that thousands of simulations had happened, they did not give us a real and absolute truth of any outcome, since the environment is not previsible. For that reason we define the lack of statistic significancy presented by ANOVA cannot be taken a genera law. In addition, spectator directly observed vestiges of well-elaborated agent decisions emerged along the simulation

by *Conscience Module (Conscience)*. The agent consciousness, its judgements and decisions, and behaviour had evolved together, as a result, one is responsible for the other, which is responsible for the other, which is responsible for the other, and so on and on. Other remarkable agent skills that could not be measured were: best smooth curve, concise deviations, and the best decisions taken in unexpected situations (the implementation of hot-swap functions to load new learned experiences when necessary).

In conclusion, the main strengths of the agent architecture, as an extensible structure of modules, and in opposition to some basic principles or some simple systems, were the optimisation strategy to achieve the optimal solution to the most promising performance in an unknown scenario, and the effective managing of symbolic and non-symbolic modules by interchanging messages among themselves. The agent speed and adaptation in unfamiliar environments was improved; consequently saving time. The use of both hierarchic and parallel organisation of behaviour modules had enabled the agent to process information from the environment, interpret them, decide the best solutions and send them to the actuators by subsumption of modules.

Chapter 7

Conclusions and Future Work

This Chapter summarises and presents the main conclusions of the research work described in this thesis. The original contributions and the limitations are also identified. Finally, future research directions are proposed.

7.1 Summary of this Thesis

In our work we have investigated the most common types of agent architectures and Machine Learning algorithms, and developed a robust architecture that allows the encoding of different and sophisticated behaviours into an artificial Cognitive Agent. In the line of Brooks, the architecture has a layer structure where each layer corresponds roughly to a cognitive level. Each layer is composed by a set of modules that together enable the simulation of behaviours. Modules are interconnected in such a way that information may influence other modules. Differently from Brooks the modules may have interconnections with modules of different layers. Layers that are more abstract (upper layers) provide increasing complexity and subsume lower level functionalities. In the horizontal lines, homogeneous modules are arranged in the same topology. In the vertical lines, heterogeneous modules are arranged in different sorts of behaviour complexity from simple behaviours to complex reasoning. Also differently from Brooks the modules may be constructed by the use of Machine Learning algorithms. With this general architecture one can assemble a particular agent choosing and connecting modules. This is facilitated by the AFRANCI tool. AFRANCI has a user friendly graphical interface and a set of facilities to easy and speed up the development of new Cognitive Agents.

7.2 Summary of this Thesis

This work addressed the lack of tools to implement an architecture with a large variety of intelligent actions in agents without decreasing of performance. Intelligent actions are aimed at autonomy, solving problems and conscience by computational models inspired in other agent architectures. Thus, many bibliographic resources supported the research to establish the basis in which the proposed solution was developed.

The proposed solution was to investigate the most common types of agent architectures and Machine Learning algorithms, and to develop a robust architecture that could emerge behaviours (perception/action), mind agent control (learning), and conscience (meta-management) by using two different cognitive lines. The main idea focuses on running processes in different periods or the same process will be performed for many modules in the "same time" in distinct and functional areas of the architecture. Levels that are more abstract provide increasing complexity and subsume lower level functionalities. In horizontal lines, homogeneous modules are arranged in the same topology. In vertical lines, heterogeneous modules are arranged in different sorts of behaviour complexity from simple behaviours to complex reasoning.

AFRANCI architecture is a framework that supports multiple different levels of abstraction organised in a hierarchical manner. In every level, control modules made of ML algorithms interchange messages. We contemplate faster knowledge acquisition to the reproduction of intelligence by addressing specific parts of problem solving, such as emergence of inner thinking over structural arrangement, background knowledge, control over internal states, and the influences of Believes, Desires and Intentions (BDI).

To construct the AFRANCI architecture, various agent states (that are, activities and their respective functional attributes) were typified in control modules, which then were arranged in structural layers. A fourth-layered hybrid architecture was implemented from simple behaviours to complex reasoning, in order to explore the structural and dimensional attributes of the layer. The layers were labelled in reflexive/reactive, instinctive, deliberative and cognitive (meta-model). Thus, we combine symbolic and non-symbolic approaches with respective Machine Learning algorithms.

Next, we have used the AFRANCI Tool to draw a detailed circuit diagram that determines the communication among layers and control modules. The inner plasticity was carefully aimed with the study of the dynamic conscience that influences the decision-making at the actuators, and layers that comprise these components. In addition, the automatic training sequence demonstrated the most promising performance in execution phase, which reached a bidirectional route

of data. In the bidirectional flow of data, the lower layers send sensory inputs to the upper layers in order to solve a problem by specialised architecture layer; consequently, the upper layers send data back to the lower layers to perform actions by its actuators. In the end, our fourth-layered hybrid architecture provides control over behaviours in a unknown simulated environment. The harmonic heterogeneity among layers and components provides agents with their own and distinct kind of intention, such as trends to represent behaviours and methods and produce results efficiently.

7.3 Research Contributions

We have proposed an hybrid architecture of interconnected heterogeneous modules. Those modules may be constructed using Machine Learning algorithms to develop autonomous and adaptable agents. The major contributions of this thesis are the following.

Hybrid Architecture We have extended the definition of abstract control levels and coordination methodologies that support heterogeneous ML algorithms interchanging messages in different flow of control, and meta-architecture for agents that virtualise many architectures in the main architecture - a new method for Multi-Strategy Learning System.

Architecture for Cognitive Agents Resume and commented analysis of the main structural organisations with their examples in unpredictable, dynamic and dangerous environment in which the Autonomous is designed to work. In addition, classification of their most promising features and combinations towards our proposed architecture. Followed by the extension of the definition of abstract control levels and coordination methodologies that support heterogeneous ML algorithms interchanging messages in different flow of control, and meta-architecture for agents that virtualise many architectures in the main architecture - a new method for Multi-Strategy Learning System. Thus, the agent may modify its performance, which means short-term and long term-internal self-modification is possible.

Machine Learning to construct Agent modules Overview of the most relevant ML algorithms used in Artificial Intelligence presented a new tendency to define intelligent agents. In addition, decision-making problems were implemented to illustrate the strengths of learning methods. Consequently, advantages and disadvantages compose particular and overall conclusions. The ML algorithms used in this thesis are Artificial Neural Networks, Rule Induction, Decisions Trees, and Genetic Algorithms which are supervised learning methods.

Framework for Designing Consciousness Autonomous Agents The proposal of a modular architecture for autonomous and adaptable agents that implement tasks previously defined has elucidated the uncertain theories about mixing symbolic and sub-symbolic approaches in the agent structure. In this framework, it is possible to divide levels of complexity into incremental functionality. The architecture combined characteristics of adaptability, robustness and uniformity that are offered by Connectionist Approach with characteristics of representation, inference and universality, which are natives of Symbolic Approach. It supports heterogeneous abstractions levels arranged in a bidirectional flow of information able to solve inner conflicts and propose solutions.

Toolkit for Building Consciousness Autonomous Agent A graphical front-end tool was built for users to design, test, debug and analyse agent architectures. The AFRANCI tool was used extensively in to attribute new functions by modification in the agent structure, and develop organised layered architectures with interconnected ML modules of both symbolic-connectionist approaches described in this thesis.

Test-bed A test-bed was implemented to evaluate our approach, which offered fast and satisfactory responses, breaking AI paradigms, and achieving organisational levels that other architectures had not reached.

7.4 Limitations

Despite having reached satisfactory results in this thesis, the following limitations were detected:

- The Agent was limited to the main ML algorithms studied. Other ML algorithms could be tested and their performance compared;
- The modules become dependent due to hierarchy and specialisation;
- AFRANCI Tool does not support interconnected workspaces that deal with unfinished projects;
- Background knowledge is necessary to off-line training time, which can slow down installation and development;
- Decision tree methods use greedy algorithms. However, the greedy nature of these algorithms can overlook multivariate relationships that cannot be found when attributes are considered separately;

- The main tests of architecture happened in labyrinths RoboCup Rescue and CyberMouse, some critical opinions could suppose that the architecture is not generic but environment dependant. As we have presented in numerous times, this critic is not supported.

7.5 Future Work

Although many topics were addressed and covered, it is possible to identify some open questions in this work, as they follow:

- Up to now, a sort of hot swap long-term memory as load. Other step could be investigate other long-term memories to different situations and implement a meta-cognitive layer (meta-meta-management level) to solve eventual conflicts;
- Since the architecture has stored all agent activities along the execution in the environment, a topic to be investigate is the possibility of the agent to learn along the execution and do not forget the rules previously learned in off-line training:
- Auto-update and fine-tune of long-term memories load in cognitive level to new experiences without user intervention:
- New structures could be added over those previously existent or extended in the same layer to enhance the behaviour complexity;
- More experiments should be designed to evaluate the architecture and utilisation of Machine Learning algorithms, extending our study about inner state induction;
- Apply the architecture in other areas, such as Biologic Evolution, and Psychology.

The methodology used in this work focused on development of simple behaviour examples to prove and test the ideas presented. Thus, the final products of this thesis were tools and models that help architect/designers to implement the agents soul with high degree of autonomy and intelligence.

Bibliography

[1] A.K. Abbas. Imunologia Celular e Molecular. Revinter, 3o. edition, 2000.

[2] P.E. Agre and D. Chapman. What are plans for? In P. Maes, editor, Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back, pages 17–34. The MIT Press: Cambridge, MA, USA, 1990.

[3] H. Akin, C. Skinner, J. Habibi, T. Koto, and S. L. Casio. Robocup 2004 Rescue Simulation League Rules V1.01. The RoboCup Rescue Technical Committee, 2004.

[4] K. Ali and R. Arkin. Implementing schema-theoretic models of animal behavior in robotic systems, 1998.

[5] Ethem Alpaydin. Introduction to Machine Learning (Adaptive Computation and Machine Learning). The MIT Press, October 2004.

[6] L.K. Altman. Manual merck. distúrbios imunes transplante. `http://www.msd.brazil.com`, December 2006.

[7] J. B. Mota Alves. Fiction, reality and expectation of behavior-based robots. In Annals of 1st Brazilian Symposium of Intelligent Automation, pages 145–154. SBIA, Rio Claro, SP, 1993.

[8] C. Angle and R. Brooks. Small planetary rovers. In EEE/RSJ International Workshop on Intelligent Robots and Systems, pages 383–388, Ikabara, Japan, 1990.

[9] L.J. Antunes. Imunologia Geral. Atheneu, 1999.

[10] Associaç ao Brasileira de Normas Técnicas. Abnt web site. Publicado por htpp em http://www.abnt.org.br/default.asp, Abril 2007.

[11] Jo ao Certo, Nuno Cordeiro, Francisco Reinaldo, Luís Paulo Reis, and Nuno Lau. Fc portugal rescue home page. `http://www.fe.up.pt/~rescue`, Feb 2006.

[12] Jo ao Certo, Nuno Cordeiro, Francisco Reinaldo, Luís Paulo Reis, and Nuno Lau. Fcpx: A tool for evaluating teams's performance in robocup rescue simulation league. Research in Computer Science, 26:127–136, Nov 2006.

[13] J.C.C. Baptista-Silva. Angiologia e cirurgia vascular: guia ilustrado., chapter Transplante renal: cirurgia no receptor: adulto, page http://www.lava.med.br/livro. UNCISAL/ECMAL & LAVA, 2003.

[14] K. S. Barber, T. H. Liu, and D. C. Han. Agent-oriented design. In Francisco J. Garijo and Magnus Boman, editors, Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World : Multi-Agent System Engineering (MAAMAW-99), volume 1647, pages 28–40. Springer-Verlag: Heidelberg, Germany, Feb 1999.

[15] Jorge Muniz Barreto. Resi: Revista eletrónica de sistemas de informção. Publicado por http://www.inf.ufsc.br/resi, Abril 2007.

[16] Daria Alexandrovna Barteneva. Computational mind models for emotional behavioral multi-agent systems. Master's thesis, Faculdade de Engenharia da Universidade do Porto (FEUP), 2006.

[17] Len Bass, Paul Clements, and Rick Kazman. Software Architecture in Practice. Addison-Wesley Professional, 2003.

[18] B. Bauer. Agent-standardization and implementation. In: Postgraduate Course on Intelligent Agents, July 1999. EPFL, Lausanne.

[19] R. D. Beer. Intelligence as Adaptative Behavior: An Experiment in Computational Neurobiology. Academic Press, 1990.

[20] David Benyon. Task analysis and system design: The discipline of data. Interacting with Computers, 4(2):246–259, 1992.

[21] F.C. Berthoux, E.H. Jones, O. Mehls, and F. Valderrabano. Pre-emptive renal transplantation in adults aged over 15 years. Nephrology Dialysis Transplantation, 11:41–43, 1996.

[22] H. D. Block. The perceptron: A model for brain functioning. i. Rev. Mod. Phys., 34(1):123–135, Jan 1962.

[23] E.J.W. Boers, H. Kuiper, B.L.M. Happel, and I.G. Sprinkhuizen-Kuyper. Designing modular artificial neural networks. In H.A. Wijshoff, editor,

Proceedings of Computing Science in The Netherlands, pages 87–96, SION, Stichting Mathematisch Centrum, 1993.

[24] W. BOGGS and M. BOGGS. Mastering UML with rational rose 2002. SYBEX, London, 2002. ISBN: 0-7821-4017-3.

[25] Jeffrey M. Bradshaw. An introduction to software agents. In Jeffrey M. Bradshaw, editor, Software Agents, pages 3–46. AAAI Press - The MIT Press, 1997.

[26] Valentino Braitenberg. Vehicles: Experiments in Synthetic Psychology. Cambridge, Mass. : MIT Press, 1984.

[27] Leo Breiman, Jerome Friedman, Charles J. Stone, and R. A. Olshen. Classification and Regression Trees. Chapman & Hall-CRC, January 1984.

[28] Rodney A. Brooks. A robust layered control system for a mobile robot. In Robotics and Automation, Journal of IEEE, volume 2, pages 14–23, 1986.

[29] Rodney A. Brooks. A robot that walks; emergent behaviors from a carefully evolved network. Neural Comput., 1(2):253–262, 1989.

[30] Rodney A. Brooks. Elephants don't play chess. Robotics and Autonomous Systems, 6(1&2):3–15, June 1990.

[31] Rodney A. Brooks. Intelligence without reason. In John Myopoulos and Ray Reiter, editors, Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91), pages 569–595, Sydney, Australia, 1991. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

[32] Arthur Buchsbaum and Francisco Reinaldo. A tool for logicians. The PracTEX Journal, (3), 2007.

[33] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, Peter Sommerlad, and Michael Stal. Pattern-Oriented Software Architecture, Volume 1: A System of Patterns. John Wiley & Sons, August 1996.

[34] R. Calder, J. Smith, A. Courtenmanche, J. Mar, and A. Ceranowicz. Modsaf behavior simulation and control. In Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation, 1993.

[35] Pedro Luis Luque Calvo. Construction of tables latex multicolumn/color. Acessado por http em `http://www.informatica.us.es/~calvo/prog/in/p4.htm`, Abril 2004.

[36] Lance Carnes. The practex journal. Publicado por http://www.tug.org/pracjourn/, Abril 2007.

[37] Yves Chauvin and David E. Rumelhart, editors. Back-Propagation: Theory, Architecture, and Applications. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA, 1995.

[38] P. Clark and R. Boswell. Rule induction with CN2: Some recent improvements. In Proc. Fifth European Working Session on Learning, pages 151–163, Berlin, 1991. Springer.

[39] P. Clark and T. Niblett. Induction in noisy domains. In Progress in Machine Learning–Proceedings of EWSL 87: 2nd European Working Session on Learning, pages 11–30, Bled, Yogoslavia, 1987.

[40] Peter Clark. Knowledge representation in machine learning. In Yves Kodratoff and Alan Hutchinson, editors, Machine and Human Learning, advances in European Research, pages 35–49. Michael Horwood, London, 1989.

[41] Peter Clark and Tim Niblett. The cn2 induction algorithm. Mach. Learn., 3(4):261–283, 1989.

[42] Agent Communication. Foundation for intelligent physical agents fipa 97 specification part 2, 2003.

[43] Jonathan H. Connell. Sss: a hybrid architecture applied to robot navigation. In Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on, pages 2719–2724 vol.3, 1992. TY - CONF.

[44] Nuno Cordeiro, Francisco Reinaldo, Jo ao Certo, Luís Paulo Reis, and Nuno Lau. Fcpx home page. `http://www.fe.up.pt/~rescue/FCPx`, Feb 2006.

[45] F. D'Agostino, A. Farinelli, G. Grisetti, L. Iocchi, and D. Nardi. Monitoring and information fusion for search and rescue operations in large-scale disasters. In Information Fusion, pages 672–679, AnnaPolis, July 2002. IEEE.

[46] Portugal Dáirio. Ue quer fora de protecção civil. Coluna Poltica, maro 2004.

[47] C. Darwin. On the Origin of Species by Means of Natural Selection. John Murray, London, 1859.

[48] DELVE data for evaluating learning in valid experiments, 2003.

[49] Artur d'Avila Garcez, Krysia Broda, and Dov Gabbay. Neural-Symbolic Learning Systems: Foundations and Applications. Perspectives in Neural Computing. Springer-Verlag, 2002.

[50] Lawrence Davis. Mapping classifier systems into neural networks. In Proceedings of the Workshop on Neural Information Processing Systems 1, pages 49–56, 1988.

[51] Randall Davis. Applications of meta level knowledge to the construction, maintenance and use of large knowledge bases. Technical report, Stanford Artificial Intelligence Laboratory, 1976.

[52] Robert A. Day. How to Write and Publish a Scientific Paper. Cambridge University Press, 6 edition, 2006.

[53] Flávio de Almeida e Silva. Hierarchic neural nets for behavioural implementation in autonomous agents. Msc thesis. department of computer science, Federal University of Santa Catarina, Florianópolis, Brazil, 2001.

[54] Instituto Brasileiro de Informação em Ciência e Tecnologia. Manifesto brasileiro de apoio ao acesso livre informao cientfica. `http://www.ibict.br/openaccess/arquivos/manifesto.htm`, Dezembro 2005. Acesso em 18 de dezembro de 2005.

[55] H. M. Deitel and P. J. Deitel. C++ How To Program. Prentice Hall, Englewood Cliffs, New Jersey 07632, 3 edition, 2001.

[56] Harvey M. Deitel and Paul J. Deitel. Java How to Program. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[57] DF. Decreto federal nÂ$\frac{1}{4}$ 2.268, June, 30th 1997.

[58] C.L. Blake D.J. Newman, S. Hettich and C.J. Merz. UCI repository of machine learning databases, 1998.

[59] Richard C. Dorf and Robert H. Bishop. Modern control systems. Prentice Hall, 10 edition, 2004.

[60] W. Duch, R. Adamczak, and N. Jankowski. Initialization and optimization of multilayered perceptrons, 1997.

[61] W. Duch and N. Jankowski. Survey of neural transfer functions, 1999.

[62] R.J. Duquesnoy. Hlamatchmaker: a molecularly based algorithm for histocompatibility determination. Human Immunology, 63:339–352, 2002.

[63] David L. Elliott. A better activation function for artificial neural networks. Technical Report TR 93-8, Institute for Systems Research, Univ. of Maryland, College Pk., College Park, MD, 1993.

[64] D. Erdogmus, O. Fontenla-Romero, J.C. Principe, A. Alonso-Betanzos, E. Castillo, and R. Jenssen. Accurate initialization of neural network weights by backpropagation of the desired response. In Proceedings of the International Joint Conference on Neural Networks, volume 3, pages 2005–2010. IEEE, 2003.

[65] Laurene V. Fausett. Fundamentals of neural networks: architectures, algorithms, and applications. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.

[66] M. E. Fayad and D. C. Schimidt. Surveying current research in object-oriented design. Communications of the ACM, 40(10):32–38, 1997.

[67] J. A. Feldman and D. H. Ballard. Connectionist models and their properties. CogSci, 6:205–254, 1982.

[68] Jacques Ferber. Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[69] M.N. Fernando, T. Deuzeny, A.G. Claudio, T. Euclides, H. Sheila, M. Marcos, and R.M. Carlos. Hlamatchmaker: a molecularly based algorithm for histocompatibility determination. Revista Brasileira de Cirurgia Cardiovascular, 16(2), April-June 2001.

[70] L. C. Figueiredo and F. G. Jota. A switching time-varying and time invariant controller to stabilize nonholonomic systems. In XIV Congresso Brasileiro de Automática - CBA, Natal - RN, 2002.

[71] L. C. Figueiredo and R. A. Teixeira. Implementação de um controlador fuzzy em clp. In II Congresso Mineiro de Automação, V Simpósio Regional de Instrumentação, pages 73–79, Belo Horizonte - MG, 1998.

[72] Luiz Carlos Figueiredo, Gilcésar Ávila, Francisco Reinaldo, Rui Camacho, Demétrio R. Magalh aes, and Luís Paulo Reis. A tool for the development of robot control strategies. EJIS, Electronic Journal of Information Systems, 11(2), September 2007. ISSN 16773071.

[73] M.J.C. Figueiredo, L.C. & Justino. Robustez em medição: Filtro de kalman aplicado a veículos autónomos. Revista DOXA, 6:75–88, 2001.

[74] Terrence L. Fine. Feedforward Neural Network Methodology. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[75] R. J. Firby and M. Slack. Task execution: Interfacing to reactive skill networks. In AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents, pages 92–96, Stanford, CA, 1995.

[76] D.S. Fitzwater, B.H. Brouhard, D. Garred, R.J. Cunningham, A.C. Novick, and D. Steinmuller. The outcome of renal transplantation in children without prolonged pre-tranplant dialysis. Clinical Pediatrics, 30:148–152, 1991.

[77] M. Fowler and K. Scott. UML essencial: um breve guia para a linguagem padro de modelagem de objetos. Bookman, Porto Alegre, 2000.

[78] S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In Intelligent Agents III. Agent Theories, Architectures and Languages, volume 1193, Berlin, Germany, 1996. Springer-Verlag.

[79] J. Fredslund and M.J. Mataric. Robot formations using only local sensing and control. Computational Intelligence in Robotics and Automation, 2001. Proceedings 2001 IEEE International Symposium on, pages 308–313, 2001.

[80] Evandro César Freiberger and Ricardo Pereira e Silva. Suporte ao uso de frameworks orientados a objetos com base no histórico do desenvolvimento de aplicações. SBQS III, page 2, 2004.

[81] LiMin Fu. Neural Networks in Computer Intelligence. McGraw-Hill, Inc., New York, NY, USA, 1994.

[82] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Padres de projeto: soluções reutilizáveis de software orientado a objetos. Bookman, Porto Alegre, 2000.

[83] R. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[84] Michael R. Genesereth and Steven P. Ketchpel. Software agents. Communications of the ACM, 37(7):48–53, July 1994.

[85] Anne Geraci. IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries. Institute of Electrical and Electronics Engineers Inc., The, 1991.

[86] Andrew G. Glen and Lawrence M. Leemis. The arctangent survival distribution. Department of Mathematics, College of William & Mary. Williamsburg, VA., 1997.

[87] David E. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Professional, January 1989.

[88] Elizabeth Gordon and Brian Logan. A goal processing architecture for game agents. In AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems, pages 998–999, New York, NY, USA, 2003. ACM Press.

[89] S. J. Gould. Darwinism and the expansion of evolutionary theory. pages 308–387. Science, 216, 1982.

[90] J. J. Grefenstette. Genesis: a system for using genetic search procedures. In Proceedings of the 1984 Conference on Intelligent Systems and Machines, pages 161–165, 1984.

[91] Steven Alex Harp, Tariq Samad, and Aloke Guha. Towards the genetic synthesis of neural network. In Proceedings of the third international conference on Genetic algorithms, pages 360–369, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[92] Nicholas J. Higham. BibTeX: A versatile tool for LaTeX users. SIAM News, 27(1):10, 11, 19, January 1994.

[93] Pascal Hitzler, Steffen Hölldobler, and Anthony K. Seda. Logic programs and connectionist networks. Journal of Applied Logic, 3(2):245–272, 2004.

[94] Joel Hoff and George Bekey. An architecture for behavior coordination learning. In In IEEE International Conference on Neural Networks, volume 5, pages 2375–2380. IEEE, 1995.

[95] D. Hogg, F. Martin, and M. Resnick. Braitenberg creatures, 1991.

[96] J. H. Holland. Adaptation in Natural and Artificial Systems. University of Michigan Press, 1975.

[97] John H. Holland, Keith J. Holyoak, Richard E. Nisbett, and Paul R. Thagard. Induction. Series: Computational Models of Cognition and Perception. The MIT Press, Cambridge, 1986. Series Editors: Jerome A. Feldman, Patrick J. Hayes, and David E. Rumelhart. First MIT Press paperback edition, 1989.

[98] Kurt Hornik, Maxwell Stinchcombe, Halbert White, and Peter Auer. Degree of approximation results for feedforward networks approximating unknown mappings and their derivatives. Neural Computation, 6(6):1262–1275, 1994.

[99] IBGE. Normas de apresentação tabular. Rio de janeiro. 61 p., 1993.

[100] IBM-Taligent(ed.). Building object-oriented frameworks. White paper, Fev. 1994.

[101] K. Ito, A. Gofuku, Y. Imoto, and M. Takeshita. A study of reinforcement learning with knowledge sharing for distributed autonomous system. In International Symposium on Computational Intelligence in Robotics and Automation, volume 3, pages 1120–1125, Okayama Univ., Japan, July 2003. Dept. Syst. Eng.,, IEEE.

[102] J. Laird J. F. Lehman and P. Rosenbloom. A gentle introduction to soar, an architecture for human cognition on-line. Available via `www.eecs.umich.edu/~SOAR/sitemaker/docs/misc/Gentle.pdf`. accessed Jan 12, 2007.

[103] Marcela Jamett and Gonzalo Acuña. An interval approach for weight's initialization of feedforward neural networks. In Alexander F. Gelbukh and Carlos A. Reyes García, editors, MICAI, volume 4293 of Lecture Notes in Computer Science, pages 305–315. Springer, 2006.

[104] Nicholas R. Jennings, Katia P. Sycara, and Michael P. Georgeff. Editorial. Autonomous Agents and Multi-Agent Systems, 1(1):5, 1998.

[105] H. George John. Cross-validated c4.5: Using error estimation for automatic parameter selection. Technical note stan-cs-tn-94-12, Computer Science Department, Stanford University, California, October 1994.

[106] R. E. Johnson and B. Foote. Designing reusable classes. Journal of Object-Oriented Programming, 1(2):22–35, 1998.

[107] R. Jones, J. Laird, M. Tambe, and P. Rosenbloom. Generating behavior in response to interacting goals. In Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation. Orlando, Florida: Institute for Simulation and Training, University of Central Florida, 1994.

[108] K. A. JONG. An analisys of the behaviour of a class of genetic adaptative systems. PhD thesis, Michigan University, USA, 1975.

[109] K-TEAM. The khepera miniature mobile robot. `http://diwww.epfl.ch/lami/robots/K-family/Khepera.html`, 2004.

[110] Leslie Pack Kaelbling. A situated-automata approach to the design of embedded agents. SIGART Bull., 2(4):85–88, 1991.

[111] J. Kalbfleish. Probability and Statistical Inference, volume 2. Springer-Verlag, New York, 1979.

[112] H. Kitano, S. Tadokor, H. Noda, I. Matsubara, T. Takhasi, A. Shinjou, and S. Shimada. Robocup-rescue: Search and rescue for large scale disasters as a domain for multi-agent research. In In Proc. of the IEEE Conference on Systems, Men, and Cybernetics, volume 6, pages 739 – 743. IEEE, 1999.

[113] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawa, and Hitoshi Matsubara. Robocup: A challenge problem for ai and robotics. In RoboCup-97: Robot Soccer World Cup I, volume 1395/1998 of Lecture Notes in Computer Science, pages 1–19. Springer, 1998. http://www.springerlink.com/content/a3826g5q77706l87/.

[114] A. Kleiner and M. Göbelbecker. Rescue3D: Making rescue simulation attractive to the public. Technical Report 00229, Institut für Informatik, Universität Freiburg, 2004. http://www.informatik.uni-freiburg.de/∼kleiner.

[115] R. Kleiss and R. Pittau. Weight optimization in multichannel monte carlo. Computer Physics Communications, 83:141, 1994.

[116] Michael Knapik and Jay Johnson. Developing intelligent agents for distributed systems: exploring architecture, technologies, & applications. McGraw-Hill, Inc., New York, NY, USA, 1998.

[117] Donald E. Knuth. The TeX book, volume 1986a of Computers and Typesetting. Addison-Wesley, 1986.

[118] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In IJCAI, pages 1137–1145, 1995.

[119] Ron Kohavi. Wrappers for Performance Enhancement and Oblivious Decision Graphs. PhD thesis, Stanford University, 1995.

[120] Ron Kohavi and Dan Summerfield. Features subset selection using the wrapper method:overfitting and dynamic search space topology. In First International Conference on Knowledge Discovery and Data Mining (KDD-95), 1995.

[121] Bryan Kolb and Ian Q. Whishaw. An Introduction to Brain and Behavior. Worth Publishers Inc, New York, 2nd edition, 2005.

[122] I. Kolmanovsky and N. H. McClamroch. Developments in nonholonomic control problems. In IEEE Trans. On Control Systems, pages 20–36, 1995.

[123] I. Kononenko, I. Bratko, and R. Roskar. Experiments in automatic learning of medical diagnosticrules. Technical report, Faculty of Electircal Engineering, E. KardeljUniversity, Ljubljana, 1984.

[124] B. Kosko, editor. Neural Networks and Fuzzy Systems – A Dynamical Systems Approach to Machinne Intelligence. Prentice Hall, Englewood Cliffs, 1991. ISBN 0136114350.

[125] J. Laird and J. Duchi. Creating human-like sythetic characters with multiple skill levels: A case study using the soar quakebot. In AAAI Fall Symposium Series: Simulating Human Agents, 2000.

[126] J. E. Laird and Michael van Lent. Developing an artificial intelligence engine. In Proceedings of the Game Developers Conference, pages 577–588. San Jose, CA, March 16-18 1999.

[127] John E. Laird. It knows what you're going to do: adding anticipation to a quakebot. In AGENTS '01: Proceedings of the fifth international conference on Autonomous agents, pages 385–392, New York, NY, USA, 2001. ACM Press.

[128] John E. Laird. Using a computer game to develop advanced ai. Computer, 34(7):70–75, 2001.

[129] John E. Laird, Allen Newell, and Paul S. Rosenbloom. Soar: an architecture for general intelligence. Artificial Intelligence., 33(1):1–64, 1987.

[130] John E. Laird and Paul Rosenbloom. The evolution of the Soar cognitive architecture. In David M. Steier and Tom M. Mitchell, editors, Mind Matters: A Tribute to Allen Newell, pages 1–50. Lawrence Erlbaum Associates, Inc., Mahwah, New Jersey, 1996.

[131] John E. Laird, Paul S. Rosenbloom, and Allen Newell. Chunking in soar: The anatomy of a general learning mechanism. Machine Learning, 1(1):11–46, March 1986.

[132] Leslie Lamport. LaTeX: A Document Preparation System. Addison-Wesley, Massachussets, 2 edition, 1994. ISBN: 0201529831.

[133] C. Larman. Utilizando UML e padrões. Bookman, Porto Alegre, 2000.

[134] N. Lavrac. Computational logic and machine learning: a roadmap for inductive logic programming. Technical report, J. Stefan Institute, Ljubljana, Slovenia, 1998.

[135] Lego. Lego mindstorm hitachi h8: 3804 robotic invention system 2.0. `http://mindstorms.lego.com`, 2002.

[136] The IEEE Computer Society Digital Library. Ieee web site. Publicado por http em http://www.ieee.org/portal/site, Abril 2007.

[137] R. Liscano, A. Manz, E.R. Stuck, R.E. Fayek, and J.-Y. Tigli. Using a blackboard to integrate multiple activities and achieve strategic reasoning for mobile-robot navigation. Expert, IEEE [see also IEEE Intelligent Systems], 10(2):24–36, 1995. TY - JOUR.

[138] John Little and Loren Shure. Signal Processing Toolbox for use with MATLAB:User's Guide. The Mathworks, Cochituate Place, 24 Prime Park Way, Natick, MA, USA, 1988.

[139] P. Maes. How to do the right thing. Connection Science Journal, Special Issue on Hybrid Systems, 1, 1990.

[140] Pattie Maes. Situated agents can have goals. Special issue of journal of Robotics and Autonomous vehicle control, 6(1-2):49–70, 1990. TY - JOUR U1 - 90090535699 Compilation and indexing terms, Copyright 2004 Elsevier Engineering Information, Inc. U2 - Autonomous Agent Situated Agents Activation/Inhibition Dynamics Situation-Orientedness Goal-Orientedness Goal-Directed Behavior.

[141] Pattie Maes. Artificial life meets entertainment: lifelike autonomous agents. Communications of the ACM, 38(11):108–114, 1995.

[142] Danilo P. Mandic and Jonathon Chambers. Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability. John Wiley & Sons, Inc., New York, NY, USA, 2001.

[143] Maja J. Mataric. Behavioral synergy without explicit integration. SIGART Bulletin 2, 2(4):130–133, 1991.

[144] Maja J. Mataric. Behavior-based control: Main properties and implications. In Proceedings, IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems, pages 46–54, Nice, France., 1992.

[145] Maja J. Mataric. Interaction and Intelligent Behavior. Thesis. PhD thesis, Massachusetts Institute of Technology, 1994. ftp://publications.ai.mit.edu/ai-publications/pdf/AITR-1495.pdf.

[146] Maja J. Mataric. Learning to behave socially. Technical report, MIT Articial Intelligence Laboratory, April 10,1997 1997. http://citeseer.ist.psu.edu/144586.html.

[147] Mathworks. Matlab. Mathworks, Inc, Natick, MA, 1999.

[148] W. S. McCulloch and W. Pitts. A logical calculus of ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, 5:115–133, 1943.

[149] A. Menon, K. Mehrotra, C. K. Mohan, and S. Ranka. Characterization of a class of sigmoid functions with applications to neural networks. Neural Networks, 9(5):819–835, 1996.

[150] Z. Michalewicz. Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, New York, 1992.

[151] R.S. Michalski. A theory and methodology of inductive learning. In R. Michalski, J. Carbonnel, and T. Mitchell, editors, Machine Learning: An Artificial IntelligenceApproach, volume 1, pages 83–134. Morgan Kaufman, Palo Alto, CA, 1983.

[152] George Armitage Miller, Eugene Galanter, and Karl H. Pribram. Plans and the Structure of Behavior. Adams Bannister Cox Pubs, September 1986.

[153] Marvin Minsky. A framework for representing knowledge. Technical report, Cambridge, MA, USA, 1974.

[154] Marvin Minsky. Jokes and their relation to the cognitive unconscious. In Vaina and Hintikka, editors, Cognitive Constraints on Communication. Reidel, 1981. http://web.media.mit.edu/~minsky/papers/jokes.cognitive.txt.

[155] Marvin Minsky. The Society of Mind. Simon & Schuster, 1988.

[156] Marvin Minsky. Negative expertise. In International Journal of Expert Systems, volume 1, pages 13–19, 1994. http://web.media.mit.edu/~minsky/papers/NegExp.mss.txt.

[157] M.L. Minsky and S.A. Papert. Perceptrons. MIT Press, Cambridge, 1969.

[158] MIT and Matthew Wall. Galib: a library of genetic algorithm components. http://lancet.mit.edu/ga/, October 2006.

[159] Perry Moerland, Georg Thimm, and E. Fiesler. Results on the steepness in backpropagation neural networks. In Marc Aguilar, editor, Proceedings of the '94 SIPAR-Workshop on Parallel and Distributed Computing, pages 91–94, Institute of Informatics, University Pérolles, Fribourg, Switzerland, October 1994. SI Group for Parallel Systems.

[160] David J. Montana and Lawrence Davis. Training feedforward neural networks using genetic algorithms. In IJCAI'89: Proceedings of the 11th international joint conference on Artificial intelligence, pages 762–767, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc. http://dli.iiit.ac.in/ijcai/IJCAI-89-VOL1/PDF/122.pdf.

[161] Takeshi Morimoto. How to Develop a RoboCupRescue Agent for RoboCupRescue Simulation System version 0. The RoboCup Rescue Technical Committee, Nov. 2002.

[162] Takeshi Morimoto. Viewer for robocuprescue simulation system. `http://ne.cs.uec.ac.jp/morimoto/rescue/viewer/`, 2002.

[163] T. Mowbray. Essentials of object-oriented architecture. Object Magazine, pages 28–32, September 1995.

[164] R. M. Murray, Z. Li, and S. Sastry. A mathematical introduction to robotic manipulation. CRC Press LLC, 1994. ISBN 0849379814.

[165] Carl Nelson. A forum for fitting the task. Computer, 27(3):104–109, 1994.

[166] Fernando Moraes Neto, Deuzeny Tenório, Claudio A. Gomes, Euclides Tenório, Sheila Hazin, Marcos Magalh aes, and Carlos R. Moraes. Transplante cardíaco: a experiência do instituto do coração de pernambuco com 35 casos. Revista Brasileira de Cirurgia Cardiovascular, 16(2)(2):152–159, April - June 2001.

[167] Allen Newell. Unified theories of cognition. Harvard University Press, Cambridge, MA, USA, 1994.

[168] Allen Newell. Unified Theories of Cognition (The William James Lectures). Harvard University Press, October 2002.

[169] D. Nguyen and B. Widrow. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptative weights. In Neural Networks for Control, volume 3, pages 21–26, 1990.

[170] D. Nguyen and B. Widrow. The truck backer-upper: An example of self-learning in neural networks. In Miller W. T, Sutton R. S, and Werbos P. J., editors, Neural Networks for Control, pages 288–299, Cambridge, MA, 1990. MIT Press.

[171] Jakob Nielsen. The usability engineering life cycle. IEEE Computer, pages 12–22, March 1992.

[172] Paul E. Nielsen. Soar/ifor: intelligent agents for air simulation and control. In WSC '95: Proceedings of the 27th conference on Winter simulation, pages 620–625, Washington, DC, USA, 1995. IEEE Computer Society.

[173] Timothy Norman, Nick Jennings, Peyman Faratin, and Abe Mamdani. Designing and implementing a multi-agent architecture for business process management. In Jörg P. Müller, Michael J. Wooldridge, and Nicholas R. Jennings, editors, Proceedings of the ECAI'96 Workshop on Agent Theories,

Architectures, and Languages: Intelligent Agents III, volume 1193, pages 261–276. Springer-Verlag: Heidelberg, Germany, 12–13 1997.

[174] Irene L. Noronha, Agenor Spallini Ferraz, Álvaro Pacheco Silva Filho, David Saitovich, Deise de Boni Monteiro de Carvalho, Flávio Jota de Paula, Henry Campos, and Luiz Estevam Ianhez. Diretrizes em transplante renal. `http://www.sbn.org.br/Diretrizes/tx`, October 2006.

[175] Manas Ranjan Patra and Hrushikesha Mohanty. A formal framework to build software agents. In APSEC, pages 119–126, 2001.

[176] David W. Payton, J. Kenneth Rosenblatt, and David M. Keirsey. Plan guided reaction,. IEEE Transactions on Systems, Man and Cybernetics, No. 6, November/December 1990, 20:1370–1382, 1990.

[177] A.G. Pipe, Y. Jin, and A. Winfield. A hybrid adaptive heuristic critic architecture for learning in large static search spaces. In Intelligent Control, 1994., Proceedings of the 1994 IEEE International Symposium on, pages 237–242, 1994. TY - CONF.

[178] D.L. Prados. New learning algorithm for training multilayered neural networks that uses genetic-algorithm techniques. Electronics Letters, 28(16):1560–1561, 30 JUL 1992.

[179] W. Pree. Design patterns for object oriented software development. Addison-Wesley, 1994.

[180] J. Ross Quinlan. Internal consistency in plausible reasoning systems. New Generation Computing, 3(2):157–180, 1985.

[181] J.R. Quinlan. Learning efficient classification procedures and theirapplication to chess end games. In R. Michalski, J. Carbonnel, and T. Mitchell, editors, Machine Learning: An Artificial IntelligenceApproach. Tioga, Palo Alto, CA, 1983.

[182] Ross J. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann, January 1993.

[183] E.L. Milford R.A.Wolfe, V.B. Ashby. Comparison of mortality in all patients on dialysis awaiting transplantation, and recipients of a first cadaveric transplant. The New England Journal of Medicine, 314:1725–1730, 1999.

[184] Francisco Reinaldo. Pyramidnet tool project homepage. `http://www.inf.ufsc.br/~rei`, 2002.

[185] Francisco Reinaldo. Projecting a framework and programming a system for development of modular and heterogeneous artificial neural networks. Msc thesis. department of computer science, Federal University of Santa Catarina, Florianopolis, Brazil, Feb 2003.

[186] Francisco Reinaldo. Dezenas de sites interessantes para latex, 2005.

[187] Francisco Reinaldo. Afranci project homepage. `http://www.afranci.com`, 2006.

[188] Francisco Reinaldo. Os pacotes mais usados em latex, 2006.

[189] Francisco Reinaldo. From the editor:editorial: Tools for latex and tex users. The PracTEX Journal, (3), 2007.

[190] Francisco Reinaldo, Rui Camacho, Luís P. Reis, and Demétrio Renó Magalhaes. Fine-tune Artificial Neural Networks Automatically, volume 1 of Leacture Notes in Electrical Engineering, 27, chapter 5, pages 39–43. Springer Verlag, 2009.

[191] Francisco Reinaldo, Rui Camacho, and Luís Paulo Reis. Afranci: An architecture for learning agents. Phd report, FEUP, Porto, Portugal, August 2005.

[192] Francisco Reinaldo, Rui Camacho, and Luís Paulo Reis. Aplicando novos paradigmas biológicos para emergir comportamentos em um sistema autónomo. II Seminário de Investigadores e Estudantes Brasileiros em Portugal - II SIEBRAP, 2005. ISSN:1646-0936.

[193] Francisco Reinaldo, Joao Certo, Nuno Cordeiro, Luís Paulo Reis, Rui Camacho, and Nuno Lau. Applying biological paradigms to emerge behaviour in robocup rescue team. In Carlos Bento, Amílcar Cardoso, and Gaël Dias, editors, EPIA, volume 3808 of Lecture Notes in Computer Science, pages 422–434. Springer, 2005. ISSN: 03029743.

[194] Francisco Reinaldo, Eliane Pozzebon, Mauro Roisenberg, and Jorge Muniz Barreto. Biological answer generated by a robotic agent. In II Symposium Catarinense of Digital Image Processing, Florianópolis, Brasil, 2002.

[195] Francisco Reinaldo, Mauro Roisenberg, Jorge Muniz Barreto, Rui Camacho, and Luís Paulo Reis. A tool for fast development of modular and hierarchic neural network-based systems. In J. Manuel Feliz-Teixeira and A. E. Carvalho Brito, editors, Proceedings of the 2005 European Simulation and Modelling Conference, pages 161–163, Porto, PT, October 2005. EUROSIS-ETI. ISBN: 9077381228.

[196] Francisco Reinaldo, Mauro Roisenberg, Rui Camacho, and Luis Paulo Reis. A tool for fast development of modular and hierarchic neural network-based systems. EJIS, Electronic Journal of Information Systems, 8(2), September 2006. ISSN 16773071.

[197] Francisco Reinaldo and Marcus Siqueira. Cn2 for microsoft windows xp. `http://www.cs.utexas.edu/users/pclark/software/~cn2`, September 2006. Peter Clarck' WebSite.

[198] Francisco Reinaldo, Marcus Siqueira, Rui Camacho, and Luís Paulo Reis. Multi-strategy learning made easy. WSEAS Transactions On Systems, Greece, 5(10):2378–2384, July 2006. ISSN 1109-2777.

[199] Francisco Reinaldo, Marcus Siqueira, Rui Camacho, and Luís Paulo Reis. A tool for multi-strategy learning. Research in Computer Science, 26:51–60, Nov 2006.

[200] Luís Paulo Reis. Coordination in Multi-Agent Systems: Applications in University Management and Robotic Soccer. Doctoral dissertation, University of Porto, Porto, Portugal, 2003.

[201] Ken Richardson. Models of Cognitive Development. Psychology Press, 1998.

[202] Helge Ritter and Teuvo Kohonen. Self-organizing semantic maps. Biologic Cybernetics, 61(4):241–254, 1989.

[203] Ronald L. Rivest. Learning decision lists. Machine Learning, 2(3):229–246, 1987.

[204] M. Roisenberg. Emergency of the intelligence in autonomous agents through inspired models in the nature. Doctoral dissertation, Federal University of Santa Catarina, Florianopolis, Brazil, 1998.

[205] M. Roisenberg, J.M. Barreto, F.d.A. Silva, and R.C. Vieira. Pyramidnet: A modular and hierarchical neural network architecture for behavior based robotics. In Proceedings of International Symposium on Robotics and Automation - ISRA 2004, page 6, Queretaro, Mexico, August 25-27 2004. IEEE.

[206] Mauro Roisenberg. Biobots. Technical report, Federal University of Santa Catarina: Florianópolis. p. 12, 2001.

[207] F. Rosenblatt. The perception: a probabilistic model for information storage and organization in the brain. Psychological Review, 65(6):386–408, 1958.

[208] F. Rosenblatt. Principles of Neurodynamics. Spartan, New York, 1962.

[209] P. S. Rosenbloom, J. E. Laird, and A. Newell. Knowledge level learning in soar. pages 527–532, 1993.

[210] P. S. Rosenbloom and A. Newell. A preliminary analysis of the soar architecture as a basis for general intelligence. pages 860–896, 1993.

[211] Paul Rosenbloom. A Symbolic Goal-Oriented Perspective on Connectionism and SOAR, pages 245–263. Elsevier, New York, 1988.

[212] Stuart Russell and Peter Norvig. Artificial Intelligence: A Modern Approach. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[213] J. P. Sauv. Definies de frameworks. `http://www.dsc.ufpb.br/~jacques/cursos/1999.2/map/material/frame/deffw.htm`, Fev. 2002.

[214] Roger C. Schank and Robert P. Abelson. Scripts, Plans, Goals and Understanding: an Inquiry into Human Knowledge Structures. L. Erlbaum, Hillsdale, NJ, 1977.

[215] Michael Schillo, Hans-Jrgen Brckert, Klaus Fischer, and Matthias Klusch. Towards a definition of robustness for market-style open multi-agent systems. In Proceedings of the fifth international conference on Autonomous agents (Agents '01), Montreal, Quebec, Canada, 2001. ACM Press.

[216] A. Y. Shamseldin, E. N. Ahmed, and K. M. O'Connor. Comparison of different forms of the multi-layer feed-forward neural network method used for river flow forecast combination. Journal of Hydrology and Earth System Sciences, 6(4):671–684, 2002.

[217] Asaad Y. Shamseldin, Ahmed E. Nasr, and Kieran M. OConnor. Comparison of different forms of the multi-layer feed-forward neural network method used for river flow forecasting. Hydrology and Earth System Sciences (HESS), 6(4):671–684, 2002.

[218] A. Shapiro. The Role of Structured Induction in Expert Systems. Phd thesis, University of Edinburgh, Edinburgh, Scotland, 1983.

[219] Mary Shaw and David Garlan. Software architecture: a roadmap. In ICSE '00: Proceedings of the Conference on The Future of Software Engineering, pages 91–101, New York, NY, USA, 2000. ACM.

[220] R. P. Silva. Suporte ao desenvolvimento e uso de frameworks e componentes. Dissertação de doutorado em ciência da computação, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil, 2000.

[221] Herbert A. Simon. The Sciences of the Artificial. MIT Press, Cambridge, Massachusetts, first edition, 1969.

[222] Herbert A. Simon. The Sciences of the Artificial - 3rd Edition. The MIT Press, October 1996.

[223] Push Singh. Examining the society of mind. Computers and Artificial Intelligence, 22(6), 2003.

[224] Push Singh. EM-ONE: An Architecture for Reflective Commonsense Thinking. PhD thesis, Massachusetts Institute of Technology, 2005.

[225] Adel Smeda, Tahar Khammaci, and Mourad Oussalah. 1:454–460, 2005.

[226] David Canfield Smith, Allen Cypher, and Jim Spohrer. Kidsim: programming agents without a programming language. Communications of the ACM, 37(7):54–67, 1994.

[227] SNT. Sistema nacional de transplantes. `http://dtr2001.saude.gov.br/transplantes`, April 2007.

[228] Stephen W. Soliday. Programmable transfer functions for neural networks. In First Industry and Academy Symposium on Research for Future Supersonic and Hypersonic Vehicles, volume 1, pages 142–147, Dec 1994. ISBN: 0962745189.

[229] Eduardo D. Sontag. Feedforward nets for interpolation and classification. Journal of Computer and System Sciences, 45(1):20–48, 1992. http://www.math.rutgers.edu/∼sontag/FTP_DIR/jcss-sigmoids.pdf.

[230] SOSTeam. Sos homepage, amirkabir university of technology. `http://ce.aut.ac.ir/~sos/`, 2005.

[231] Springer. Springer verlag web site. Publicado por http em http://www.springer.com/east/home?SGWID=5-102-0-0-0, Abril 2007.

[232] M. Srinivas and Lalit M. Patnaik. Genetic algorithms: A survey. Computer, 27(6):17–26, June 1994.

[233] L. Steels. Exploiting analogical representations. In Pattie Maes, editor, Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back, pages 71–88. MIT Press, Cambridge, MA, USA, 1990.

[234] Peter Stone and Manuela M. Veloso. Layered learning. In Machine Learning: ECML 2000, 11th European Conference on Machine Learning, Barcelona, Catalonia, Spain, May 31 - June 2, 2000, Proceedings, volume 1810, pages 369–381. Springer, Berlin, 2000.

[235] SUN. http://java.sun.com/. Publicado por http., Abril 2007.

[236] Milind Tambe, W. Lewis Johnson, Randolph M. Jones, Frank V. Koss, John E. Laird, Paul S. Rosenbloom, and Karl Schwamb. Intelligent agents for interactive simulation environments. AI Magazine, 16(1):15–39, 1995.

[237] Arian Team. Official arian home page. `http://ce.sharif.edu/~arian/`, Nov 2005. Sharif University of Technology.

[238] A. M. Turing. Computing machinery and intelligence. Mind, 58:433–460, 1950.

[239] Jari Vaario. An Emergent Modeling Method for Artificial Neural Networks. PhD thesis, The University of Tokyo, 1993.

[240] Michael van Lent, John Laird, Josh Buckman, Joe Hartford, Steve Houchard, Kurt Steinkraus, and Russ Tedrake. Intelligent agents in computer games. In AAAI '99: Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence, pages 929–930, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.

[241] Sankar Virdhagriswaran. MuBot. (no longer available directly: quote from `http://www.msci.memphis.edu/~franklin/AgentProg.html`).

[242] Raul S. Wazlawick. Análise e Projeto de Sistemas de Informação Orientados a Objetos. Elsevier, 2004.

[243] Darrell L. Whitley. Cellular genetic algorithms. In Proceedings of the 5th International Conference on Genetic Algorithms, page 658, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[244] B. Widrow. Generalization and information storage in networks of adaline "neurons". In M.C. Yovits, G.T. Jacobi, and G.D. Goldstein, editors, Self-Organizing Systems 1962, pages 435–461, Washington, 1962. (Chicago 1962), Spartan.

[245] B. Widrow and M.E. Hoff. Adaptative switching circuits. IRE WESCON Convention Records, a:96–104, 1960.

[246] N. Wiener. Cybernetics: or Control and Communication in the Animal and the Machine. MIT Press, Cambridge, MA, 1948.

[247] M. Wiesmeyer. Soar i/o reference manual, version 2. Technical report, Department of EECS. University of Michigan, 1988.

[248] Rebecca J. Wirfs-Brock and Ralph E. Johnson. Surveying current research in object-oriented design. Communications of the ACM, 33(9):104–124, 1990.

[249] Ian Witten and Eibe Frank. Data Mining: Practical Machine Learning Tools and Techniques (Second Edition). Morgan Kauffmann, 2005.

[250] Ian H. Witten and Eibe Frank. Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann, San Francisco, 2nd. edition, 2005.

[251] Michael Wooldridge and Nicholas R. Jennings. Agent theories, architectures, and languages: a survey. In ECAI-94: Proceedings of the workshop on agent theories, architectures, and languages on Intelligent agents, pages 1–39, New York, NY, USA, 1995. Springer-Verlag New York, Inc.

[252] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. Knowledge Engineering Review, 10(2):115–152, 1995.

[253] Michael J. Wooldridge and Nick R Jennings. Intelligent agents: Theory and practice. In The Knowledge Engineering Review, volume 2, pages 115–152. 1995.

[254] Robert Wray, Ron Chong, Joseph Phillips, Seth Rogers, and Bill Walsh. A Survey of Cognitive and Agent Architectures. University of Michigan, 1994.

[255] Gül Yazici, Övünç Polat, and Tülay Yildirim. Genetic optimizations for radial basis function and general regression neural networks. In Alexander F. Gelbukh and Carlos A. Reyes García, editors, MICAI, volume 4293 of Lecture Notes in Computer Science, pages 348–356. Springer, 2006. 3-540-49026-4.

[256] A. Zell, N. Mache, T. Sommer, and T. Korb. Design of the snns neural network simulator. In H. Kaindl, editor, 7. Österreichische Artificial-Intelligence-Tagung, pages 93–102. Springer, Berlin, Heidelberg, 1991.

[257] Andreas Zell, Niels Mache, Ralf Huebner, Michael Schmalzl, Tilman Sommer, and Thomas Korb. Snns: Stuttgart neural network simulator. Technical report, University of Stuttgart, Institute for Parallel and Distributed High Performance Sytems(IPVR), Stuttgart, 1992.

[258] Gilad Zlotkin and Jeffrey S. Rosenschein. Negotiation and goal relaxation. In Y. Demazeau and J. P. Müller, editors, Decentralized A. I. 2, Proceedings of the Second European Workshop on Modeling Autonomous Agents in a Multi-Agent World, pages 273–286. Elsevier Science Publishers B.V./North-Holland, 1991.