



Nathalia Moraes do Nascimento

**FloT: An Agent-Based Framework for Self-Adaptive
and Self-Organizing Internet of Things Applications**

DISSERTAÇÃO DE MESTRADO

Dissertation presented to the Programa de Pós-graduação em Informática of the Departamento de Informática do Centro Técnico Científico da PUC-Rio as partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor: Prof. Carlos José Pereira de Lucena

Rio de Janeiro
August 2015



Nathalia Moraes do Nascimento

**FIoT: An Agent-Based Framework for Self-Adaptive
and Self-Organizing Internet of Things Applications**

Dissertation presented to the Programa de Pós-graduação em Informática of the Departamento de Informática do Centro Técnico Científico da PUC-Rio as partial fulfillment of the requirements for the degree of Mestre.

Prof. Carlos José Pereira de Lucena

Advisor

Departamento de Informática — PUC-Rio

Prof. Andrew Diniz da Costa

Departamento de Informática –PUC-Rio

Prof. Hugo Fuks

Departamento de Informática –PUC-Rio

Prof. Ruy Luiz Milidui

Departamento de Informática –PUC-Rio

Prof. José Eugênio Leal

Coordinator of the Centro Técnico Científico da PUC-Rio

Rio de Janeiro, August 31 th, 2015

All rights reserved.

Nathalia Moraes do Nascimento

B.Sc. in Computer Engineering, State University of Feira de Santana (UEFS), 2013. She is a researcher member of the Software Engineering Laboratory at the Pontifical Catholic of Rio de Janeiro since 2013. Her main studies are related to the area of software engineering, artificial intelligence, multi-agent systems, and internet of things.

Bibliographic data

Nascimento, Nathalia Moraes do

FloT: An Agent-Based Framework for Self-Adaptive and Self-Organizing Internet of Things Applications / Nathalia Moraes do Nascimento; advisor: Carlos José Pereira de Lucena. — 2015.

102 f. : il. (color); 30 cm

1. Dissertação (Mestrado em Informática) - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2015.

Inclui bibliografia.

1. Informática – Teses. 2. Sistema Multiagente. 3. Auto-Organização. 4. Autoadaptação. 5. Internet das Coisas. 6. Aprendizado de Máquina. I. Lucena, Carlos José Pereira de. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

Aknowledgments

Thank You, God, for putting the correct people in my life. I could not have done this work without them:

- My family - Nilza, Geraldo, Gabriela, and Júllia- who always give me all the support that I need. They are my greatest inspiration for becoming a better person;
- My advisor, Carlos Lucena, who is my biggest reference;
- My boyfriend, Juliano, who I admire so much and motivates me to be more studious and organized
- My friends and colleagues, specially my good friends Carolina Valadares and João Dutra. Carol convinced me to attend PUC-Rio and received me in her house for the first month in Rio, which was the most difficult. João helped me many times;
- The staff members of the Department of Informatics, specially professor Hugo Fuks, who shared with me some of his creativity, and Regina Zanon, who is so patient with the students;
- Professor Jean Pierre, who contributed to my research and helped me with the last reviews of this dissertation;
- My research group, specially Andrew, Marx, Davy, and Roberto, who shared with me some of their advanced academical experiences and helped me a lot.

This work was supported by the Laboratory of Software Engineering (LES) at PUC-Rio. Our thanks to CNPq, CAPES, FAPERJ and PUC-Rio for their financial support.

Abstract

Nascimento, Nathalia Moraes do; Lucena, Carlos José Pereira de (Advisor). **FIoT: An Agent-Based Framework for Self-Adaptive and Self-Organizing Internet of Things Applications**. Rio de Janeiro, 2015. 102p. MSc. Dissertation — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

The agreed fact about the Internet of Things (IoT) is that, within the coming years, billions of resources, such as cars, clothes and foods will be connected to the Internet. However, several challenging issues need to be addressed before the IoT vision becomes a reality. Some open problems are related to the need of building self-organizing and self-adaptive IoT systems. To create IoT applications with these features, this work presents a Framework for Internet of Things (FIoT). Our approach is based on concepts from Multi-Agent Systems (MAS) and Machine Learning Techniques, such as a neural network and evolutionary algorithms. An agent could have characteristics, such as autonomy and social ability, which makes MAS suitable for systems requiring self-organization (SO). Neural networks and algorithms of evolution have been commonly used in robotic studies to provide embodied agents (as robots and sensors) with autonomy and adaptive capabilities. To illustrate the use of FIoT, we derived two different instances from IoT applications: (i) Quantified Things and (ii) Smart Cities. We show how flexible points of our framework are instantiated to generate an application.

Keywords

Multi-Agent System; Self-Organization; Self-Adaptation; Internet of Things; Machine Learning.

Resumo

Nascimento, Nathalia Moraes do; Lucena, Carlos José Pereira de. **FIoT: Um Framework Baseado em Agentes para Aplicações Auto-Organizáveis e Autoadaptativas de Internet das Coisas**. Rio de Janeiro, 2015. 102p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A ideia principal da Internet das Coisas (IoT) é conectar bilhões de coisas à Internet nos próximos anos, a exemplo de carros, roupas e comidas. Entretanto, muitos problemas precisam ser resolvidos antes que essa ideia possa ser concretizada. Alguns desses problemas estão relacionados à necessidade de construir sistemas para IoT que sejam auto-organizáveis e autoadaptativos. Este trabalho, portanto, apresenta a elaboração do Framework para Internet das Coisas (FIoT), que oferece suporte ao desenvolvimento de aplicações para IoT com essas características. Ele é baseado nos paradigmas de Sistemas Multiagente (SMA) e algumas técnicas abordadas em Aprendizado de Máquina, a exemplo de redes neurais e algoritmos evolutivos. Um agente pode ter algumas características, como autonomia e sociabilidade, que tornam SMAs compatíveis com sistemas que requerem auto-organização. Redes neurais e algoritmos de evolução vêm sendo comumente usados nos estudos de robótica, no intuito de prover autonomia e adaptação à agentes físicos (ex.: robôs, sensores). Para demonstrar o uso do FIoT, dois grupos de problemas em IoT serão instanciados: (i) Cidades Inteligentes e (ii) Quantificação de Coisas.

Palavras-chave

Sistema Multiagente; Auto-Organização; Autoadaptação; Internet das Coisas; Aprendizado de Máquina.

Contents

1	Introduction	11
1.1	Problem Statement	13
1.2	Out of Scope	14
1.3	Objectives	14
1.4	Proposed Solution	15
1.5	Contributions	15
1.6	Dissertation Organization	16
2	Background and Related Work	17
2.1	The Internet of Things	17
2.2	Multi-agent System	18
2.3	Evolutionary Algorithms	23
2.4	Artificial Neural Network (ANN)	25
2.5	Evolutionary Robotics	31
2.6	Self-adapting and Self-Organizing Systems	33
2.7	MAS for IoT	37
3	FloT: Framework for Internet of Things	39
3.1	Domain Analysis	40
3.2	Agent-Based Model	41
3.3	Central Idea for the Framework Design	42
3.4	Details of FloT	49
3.5	How to instantiate FloT: Technical Details	51
4	Evaluation: Illustrative Examples	58
4.1	FloT's Instances	58
4.2	Quantified Things	59
4.3	Smart City	70
4.4	A Future Instance: Quantified US	83
4.5	How the generated applications adhere to FloT	85
5	Conclusion	89
5.1	Evaluation and Future Works	90
6	Bibliography	92

List of Figures

2.1	Arduino Yún (Arduino, 2014).	18
2.2	A Thing.	18
2.3	The diagram of a generic agent provided by authors in (Russell and Norvig, 1995) [p.32].	19
2.4	Modeling an Embodied Agent based on Figure 2.3 and the literature description provided in this section.	23
2.5	A generic diagram of an evolutionary algorithm.	24
2.6	Block diagram representation of nervous system (Haykin, 1994). Pg.28.	25
2.7	The model of a neuron (Haykin, 1994). Pg.33.	26
2.8	Multilayer feedforward network with two hidden layers (Haykin, 1994). Pg.181.	28
2.9	Recurrent network (Haykin, 1994). Pg.46.	29
2.10	Evolving a neural network. Adapted from Floreano and Mattiussi (2008, p.317).	31
2.11	Evolving Embodied Agents.	33
2.12	IBM Control Loop to generate autonomous elements (Jacob et al., 2004). Pg 24.	35
3.1	An agent-based model to generate IoT applications.	41
3.2	Activity diagram of GodAgent.	43
3.3	Control loop provided by framework in (Neto et al., 2009).	43
3.4	Control loop provided by the FloT framework.	44
3.5	Activity diagram of the Adaptive Agent.	45
3.6	Activity diagram of ObserverAgent.	46
3.7	Activity diagram of a Thing.	47
3.8	Use Case Diagram.	49
3.9	Packages of the FloT project.	50
3.10	Class diagram of FloT - Agents.	50
3.11	Class diagram of FloT - Behaviors.	51
3.12	Class diagram of FloT - Controllers.	51
3.13	Examples of controllers registered by the system developer via text files.	55
3.14	Configuring the execution of a genetic algorithm.	56
4.1	Different configurations to create FloT's instances.	59
4.2	The model for individual tracking in QT systems.	62
4.3	Bananas from the same bunch after being stoted in different conditions.	64
4.4	An instance of FloT to create "Quantified Things" Instances.	65
4.5	Set of sensors.	65
4.6	Example of scenarios.	66
4.7	Introducing the IoT paradigm for transportation system analysis (Möller, 2014). Pg. 104.	70

4.8	A FloT instance for creating a smart controller for homogeneous things.	71
4.9	Traffic elements.	72
4.10	Modeling a scenario as a graph.	72
4.11	Adaptive Agent's Neural Controller.	74
4.12	Performing an adaptive process to adjust the Traffic Neural Controller weights. Figure adapted from (Nolfi and Gigliotta, 2010). P.7.	75
4.13	Simulation Urban Road Network. Adapted from Waze (2014) (MOBILE, 2014).	76
4.14	The configuration file for the evolutionary algorithm.	77
4.15	Simulation Results - Best Fitness.	77
4.16	Comparison of the FloT approach and conventional systems in the first scenario.	79
4.17	The second scenario - Copacabana - RJ -BR. The car 9's route. Adapted from Waze (2015) (MOBILE, 2014).	79
4.18	Comparison of the FloT approach and conventional systems in the second scenario.	80
4.19	Agent's Neural Controller.	80
4.20	Comparison of the evolved agents approach and conventional systems in the first scenario.	81
4.21	Comparison of the evolved agents approach and conventional systems in the second scenario.	82
4.22	Relation between gases emitted and diagnoses.	84
4.23	Future Instance: QS model.	85
4.24	Future Instance: QU model.	86
4.25	How the generated applications fill the main variable parts of FloT.	87
4.26	A comparison between Quantified Self and Quantified Us instances' design.	88

List of Tables

3.1	How the model and FloT meet the IoT requirements.	48
4.1	Specific requirements for “Quantified Things” applications.	63
4.2	Experimental Description	66
4.3	Initial Database for neural network training.	67
4.4	Results of backpropagation execution	68
4.5	New data to also be used in backpropagation.	69
4.6	Instance I: Flexible Points	86
4.7	Instance II: Flexible Points	87

1

Introduction

Internet of Things (IoT) is a broad concept. In general, IoT refers to a global infrastructure of networked physical things interconnected through Internet (Rodrigues et al., 2012; Park et al., 2013). The central perspective is that, within the coming years, billions of resources, such as cars, lamps, foods and factory machinery will be connected to the Internet and share information about themselves and their environments. IoT will make it possible to develop a variety of application scenarios, such as smart homes and cities, e-health, environmental monitoring and many others. Smart traffic management is an example of a smart city application, which aims at providing intelligent transportation through real-time traffic information and path optimization (Gubbia et al., 2013).

According to the authors in (Atzori et al., 2010), the potentialities offered by IoT make it possible to develop a huge number of applications, a very small part of which is currently available to our society. Most IoT applications are not developed yet because they require scalability beyond millions of devices where centralized solutions could exceed their boundaries. Further, they cannot have fixed deployments or fixed systems configurations, as the environment is in continuous transition (Fortino and Trunfio, 2014). For example, an autonomous application for traffic management depends on the abilities of the traffic light controllers to adapt to changing traffic situations (Rochner et al., 2006). We can observe that traffic changes according to different time-scales. As the authors describe in (Rochner et al., 2006), a typical workday can be divided into several periods of different traffic situations, including two peak periods with high demands due to commuter traffic.

The truth is that several challenging issues still need to be addressed before the IoT vision becomes a reality (Velloso et al., 2010; Bandyopadhyay and Sen, 2011; Atzori et al., 2010). A possible further complication is the fact that the vast majority of present research in universities and industry is paying more attention to operational technology, in order to solve problems related to limited Internet traffic capacity, communication protocols, and network architecture (Lopez and Pérez, 2012). For example, the authors in (Gubbia

et al., 2013) discuss the open challenges and future directions in the Internet of Things. They present global addressing schemes, cloud storage, and wireless power as the key elements of the current IoT research. In their opinion, self-adaptive system of systems is an example of the key application outcomes that are only expected in the next decade.

In (Fortino and Trunfio, 2014), one of the few works that bring a non-operational IoT approach, the authors relate some open issues which are needed to build elements capable of coping with the changing environments and taking appropriate decisions autonomously. Therefore, they discuss about the importance of creating autonomous and adaptive IoT systems. In an effort to call attention to these issues, a new terminology associated with IoT is emerging: Smart Objects (SOs) or Smart Things. They represent loosely coupled and decentralized systems of cooperating objects. Editors in (Fortino and Trunfio, 2014) discuss smart objects and define them as an autonomous, physical digital object augmented with sensing/actuating, processing, interpretation, storing, and networking capabilities.

IoT is a new and exciting approach, and will soon be adopted by the market (Gubbia et al., 2013). To define new frameworks/middlewares (Beydeda et al., 2005; Sommerville, 2004) for the rapid prototyping, there is a need to facilitate the development process of SOs. Frameworks are general software systems (i.e. systems that consist of abstract and concrete classes), which can be adapted or extended to create more specific applications. According to Ian Sommerville (Sommerville, 2004), “the sub-system is implemented by adding components to fill in parts of the design and by instantiating the abstract classes in the framework.”

Meanwhile, a few framework/middleware approaches have been proposed to support the creation of a SO-based IoT infrastructure (Fortino et al., 2012). The authors in (Fortino et al., 2014) analyze the existing approaches and discuss their limit in the management of a vast number of cooperative SOs – none of them presented the design of any complex application scenario (i.e. using a vast number of cooperative things, such as traffic scenarios). For example, the authors in (Fortino et al., 2013) developed a middleware for smart objects and affirm that the support of distributed computing entities is the key and novel feature of their approach. Nonetheless, to illustrate the use of their architecture, they present a simple case study, which refers to a smart office environment constituted by only two cooperating things. These works do not show efficiency in complex scenarios, where things must cope with a changing environment and where a sophisticated organization system is required.

According to the authors in (Fortino et al., 2014), to develop SO-based

IoT systems, novel software engineering methodologies for dynamic systems need to be defined. Due to proliferation of the Internet and the spread of mobile devices, an increasing number of applications are suitable for using self-organization to fulfill their goals, such as manufacturing control and traffic management (Serugendo et al., 2005). Accordingly, we aim at facilitating the development of smart systems within the Internet of Things domain, providing them with more autonomy, self-adaptive and self-organizing properties. In order to do this, we first propose a model to create smart things. Thus, we present a framework, named “Framework for Internet of Things” (FIoT), as an effort to facilitate the process of creation of smart things.

The objective of FIoT is to facilitate the creation of diverse applications, such as controllers for car traffic, factory machines, public lighting, and smart homes. Hence, the framework allows the creation of autonomous controllers for groups of homogeneous things which can be connected to the Internet. To illustrate the use of FIoT, we will present examples from two of the Internet of Things applications: (i) Quantified Things and (ii) Smart Cities. The next subsections of this chapter aim at presenting our work motivation and objectives, besides describing how this document is organized.

1.1

Problem Statement

A few years ago, the authors in (Kephart and Chess, 2003) called the global goal to connect trillions of computing devices to the Internet the nightmare of ubiquitous computing (Lyytinen and Yoo, 2002). The reason for that is that to reach this global goal requires a lot of skilled Information Technology (IT) professionals to create millions of lines of code, and install, configure, tune, and maintain these devices. According to the author in (Kephart, 2005), in few years, IT environments will be impossible to be administered, even by the most skilled IT professionals.

Predicting the emergence of this problem, in 2001 the IBM company suggested the creation of autonomic computing (Horn, 2001). IBM recognized that the only viable solution to resolve this problem was to endow systems and the components that comprise them with the ability to manage themselves in accordance with high-level objectives specified by humans (Kephart, 2005). Therefore, IBM proposed systems with self-developed capabilities. The company emphasized the need of automating IT key tasks, such as coding, configuring, and maintaining systems, based on the progress observed in the automation of agriculture’s manual tasks.

Other IT companies agreed with IBM and then generated their own man-

ifests, such as Hewlett-Packard (HP, 2003) and Microsoft (Microsoft, 2004). However, the IT Industry interest in the development of self-management devices is not evident yet. As a result, not only the goal of the Internet of Things to connect billions of devices to the Internet has not been reached, but we have also been experiencing the problems previously listed by the authors in (Kephart and Chess, 2003).

The truth is that companies and researchers are so busy competing to define the official protocol and architecture for the Internet of things, that very few researches to provide a sophisticated control to manage all these billions of things have been developed. As a result, there is a lack of software to support the development of a huge number of different IoT applications.

1.2 Out of Scope

As the development of smart objects is part of a broader context, a set of related aspects will be left out of the scope of this work. Thus, the following approaches are not directly addressed by this work:

- Security
- Ontology
- Protocols
- Scalability

1.3 Objectives

Our objective in this dissertation is to propose a feasible solution through a general software system for providing self-organizing and self-adaptive behaviors for Internet of Things applications. To this end, our approach consists in:

- Developing Autonomous things:
 - Things that are able to cooperate and execute complex behavior without the need for centralized control to manage their interaction.
 - Things that are able to have behavior assigned at design-time and/or at run-time.
- Providing mechanisms to automatically recognize and control autonomous things in the environment;
- Providing mechanisms to allow things to self-adapt and to improve their own behavior;

1.4

Proposed Solution

To reach these objectives, we propose the following directions:

1. To make a generic software basis for IoT, we propose the development of a framework. The framework approach can be used to rise the common requirements among IoT applications and implement a reusable architecture (Markiewicz and Lucena, 2001).
2. To create autonomous things and a distributed control, we decided to model the framework based on a multi-agent approach. According to the authors in (Cetnarowicz et al., 1996), the active agent was invented as a basic element from which distributed and decentralized systems could be built. In our approach, we consider the use of embodied agents, which is typically used to model and control autonomous physical objects that are situated in actual and complex environments (Steels, 2004).
3. To control the things, we choose a control architecture based on artificial neural networks. A neural network is a well known approach to provide dynamically responses and automatically create a mapping of input-output relations (Haykin, 1994). In addition, it is commonly used as an internal controller of embodied agents (Marocco and Nolfi, 2007).
4. To make things self-adaptive, we propose using various Machine Learning (ML) techniques, notably supervised learning and evolutionary algorithms.

1.5

Contributions

Our contributions are multifold:

1. A multi-agent software architecture (a framework) to construct IoT applications with self-adaptive agents monitoring and controlling the things and their interactions.
2. Instances of this framework to two types of IoT applications:
 - (a) Prediction application, where things may self predict its condition. For this purpose, we proposed a new approach of IoT applications in this work, named “Quantified Things”. The overall idea of Quantified Things applications is to deliver a way of incorporating the small data generated by each thing into larger datasets to get

more meaning out of these data. A thing, equipped with sensors, can be monitored and can record specific features regarding its behavior. Then it can populate a cloud database in order to provide individual- and collective-level analyses.

- (b) Control application, a decentralized control for a collection of homogeneous things. For a simulated car traffic application, we show how to instantiate the framework to create autonomous and dynamic controllers for traffic lights.
3. Design of an adaptive agent based on a generic neural network architecture which accepts two types of adaptation:
- (a) Supervised learning, via back-propagation algorithm, to support classification and prediction (used in 2a application)
 - (b) Evolutionary algorithm to discover control policies (used in 2b application)

1.6

Dissertation Organization

In addition to the Introduction, this dissertation contains four more chapters, as follows:

Chapter 2 provides a background of the main concepts used in our work. Chapter 3 describes the proposed framework, FIoT and its development. In Chapter 4, we present the experiments setup. Finally, the chapter 5 provides our conclusions and future works.

2

Background and Related Work

In this chapter, we first provide a brief introduction to the Internet of Things, which is the domain of this research project. Next, we define some concepts involved in our proposed solution, explaining our main motivations for using them. In particular, we briefly describe some definitions and advantages of (i) Multi-agent System (MAS), (ii) Self-Adaptive and Self-organizing systems (SASO), and some artificial intelligence techniques, such as (iii) neural networks and (iv) evolutionary algorithms.

Finally, we provide an overview of frameworks in the literature while addressing the idea of mixing Artificial Intelligence and Internet of Things concepts, especially those with focus on multi-agent systems.

2.1

The Internet of Things

The Gartner Company (Stamford, 2014) promotes annually a research to investigate the technology trends in the world. According to recent research results, the Internet of Things is one of the greatest technological revolutions in recent years. In the next years, there will be more and more objects connected to the Internet, which can be monitored and controlled independently. As a result, the tendency is to substitute centralized computing for distributed computing.

Different kinds of low-cost microelectronics with high potential to connect to the Internet have been quickly emerging. They are the key to the Internet of Things, which makes it possible for the physical world to interface with the Internet (Pfister, 2011). We can also notice a popularization of these microelectronics - they are not only cheaper, but also easier to handle. The Arduino (Arduino, 2014) is one of the electronic platforms that has been allowing microcontroller boards to be handled by anyone. It promotes the development of electronic platforms based on the Open-Source Hardware (referring to the principles of the Open-Source Software - for more information, see (Jaeger and Metzger, 2002)). Figure 2.1 illustrates the Arduino Yún, which is the Arduino model used in our work. See (Arduino, 2014) for more

information about Arduino boards.

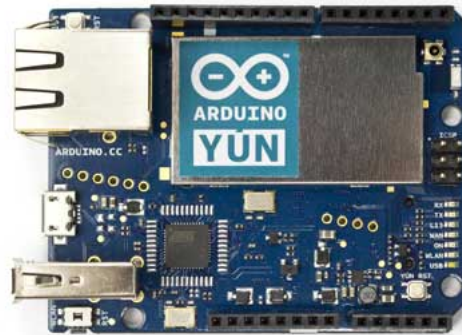


Figure 2.1: Arduino Yún (Arduino, 2014).

In the IoT context, a thing is an object provided with an IP address and the ability to transfer data over a network. For example, it can be a person wearing a heart monitor implant or a farm animal carrying a biochip (Vaidya et al.; Vimala and Rajaram, 2014). Accordingly, Figure 2.2 illustrates the concept of a “Thing” that we have been used in this dissertation. In this work, a thing consists of a device linked to an object with the purpose of monitoring or controlling it. A device makes it possible that an object in the environment to interface with the Internet and an application system. A device consists of sensors, actuators and a microcontroller.

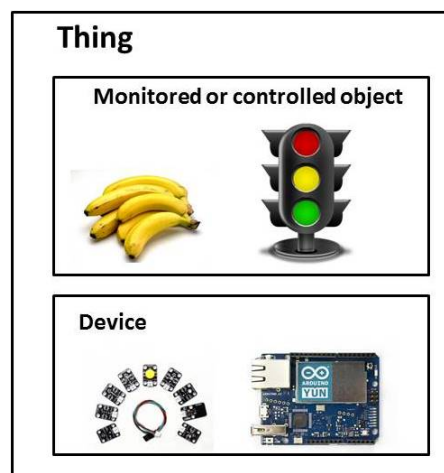


Figure 2.2: A Thing.

2.2 Multi-agent System

The aim of this section is to provide an understanding of what agents are, and some of the properties associated with their usability. The authors

in (Russell and Norvig, 1995) define “**agent**” as “anything that can be viewed as **perceiving** its environment through **sensors** and **acting** upon that environment through **effectors**.” Accordingly, they describe the sensors and actuators of different agent types. For example, a human agent has eyes, ears, and other organs for sensors, and hands, legs, mouth, and other body parts for effectors. A robotic agent uses cameras and infrared range finders as sensors and various motors as effectors. A software agent has “strings encoded” as its sensors and actuators. They provide an abstract view of an agent, which is illustrated in Figure 2.3. It shows the action output generated by the agent in order to affect its environment.

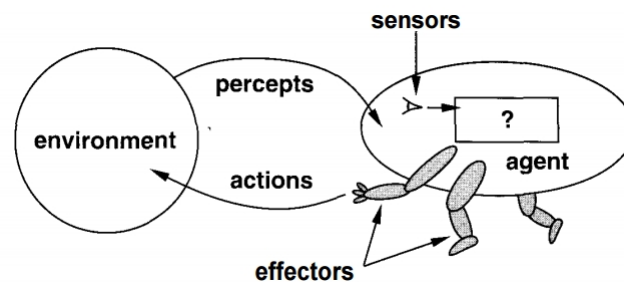


Figure 2.3: The diagram of a generic agent provided by authors in (Russell and Norvig, 1995) [p.32].

An agent can inhabit a real or simulated environment. The environments can differ taking other properties into account. For example, if the environment can change while an agent is deliberating, then we say the environment is dynamic for that agent; otherwise it is static. Different environment types require different agent in order to deal with them effectively (Russell and Norvig, 1995). The most general properties of an agent are (Wooldridge and Jennings, 1995):

- Autonomy: an agent functions without intervention. It has the control over own actions and internal state;
- Reactivity: it perceives its environments and responds to changes;
- Pro-activeness: it acts in anticipation of future goals (goal-directed behavior);
- Social ability: it interacts with other agents (and even humans).

Various other attributes are discussed in the context of agency (Wooldridge and Jennings, 1995). For example:

- Mobile: an agent has the ability to move around an electronic network (White, 1994);

- Situated: an agent experiences the environment using sensors and acts using effectors (Russell and Norvig, 1995);
- Embodied: an agent has a body and experiences the world directly (Brooks, 1995). According to the authors in (Brooks, 1995), disembodied systems concentrate on programming intellectual activities like chess, while the embodied approach aims at equipping a digital computer with the best sense organs (e.g. “organs” to move, talk, hear, touch, and see). We describe it better in Section 2.2.2;
- Awareness: an agent has the ability of sensing its environment in different ways, and take decisions accordingly;

According to the authors in (Wooldridge, 2009), a collection of interacting agents is a Multi-Agent System (MAS). Therefore, multi-agent systems can be used to model complex and dynamic real-world environments, which involves a vast number of entities (e.g. simulation of societies) (Poslad, 2007). It is a useful paradigm for managing large distributed information handling systems (Marzo et al., 2004).

Multi-agent systems have been applied to a wide range of application types, including e-commerce, human-computer interfaces, network control, air traffic control, and diagnosis (Lucena, 2004; Pěchouček and Mařík, 2008).

2.2.1 JADE

The Java Agent DEvelopment Framework (JADE) is a Java software framework implemented to facilitate the development of multi-agent systems (Telecom, 2015). According to the authors in (Pěchouček and Mařík, 2008), JADE is a leading open-source agent development environment on the market and some of the existing MAS applications and prototype systems use it.

JADE implements the Foundation for Intelligent Physical Agents (FIPA) specifications that represent a collection of standards for the development of agent-based systems (FIPA, 2015). One of these standards is the Agent Communication Language (ACL) (FIPA, 2015), a protocol for agent communication. It allows the development of an interoperability communication structure, which agents can execute on different platforms and exchange information (Bellifemine et al., 2007; Bellifemine et al., 2010).

However, JADE can only be used to implement agents to execute in Java-compatible systems (e.g. Windows, Android) (Bellifemine et al., 2010). Thus, a JADE agent cannot be directly deployed in an Arduino microcontroller board, for example, since it gives support only to the development of C-language

programs (Arduino, 2014). Probably, in the future will be possible to install the Java Virtual Machine (JVM) in a microcontroller board as an Arduino. Another solution is to investigate FIPA platforms for the development of C++ agents, such as Mobile-C (Chen et al., 2006). Thus, an agent could be deployed in an Arduino and communicate with a JADE software agent through ACL messages.

As we decided to work only with JADE, that is a well-known framework for agent development, we created a software layer to make it possible for a JADE multi-agent system to interface with Arduino boards via the Internet.

2.2.2

Embodied Agents

There is a growing acknowledgment in the artificial life and artificial intelligence communities about the importance of investigating systems' bodies before understanding and modeling their cognition (Pfeifer and Bongard, 2006). Imagine you want to build a smart fireplace that can decide whether to light itself or not. First, you need to define the sensors and actuators of this fireplace. Will it have a temperature sensor? And what about a presence sensor? Will it have actuators to control the light intensity? The type of sensors and actuators that your fireplace received directly affects the behaviors that it can have.

According to the authors in (Brooks, 1995), “only an embodied intelligent agent is fully validated as one that can deal with the real world.” The embodied agents have a body and are physically situated, that is, they are physical agents interacting not only among themselves but also with the physical environment. They can communicate among themselves and also with human users. Robots, wireless devices and ubiquitous computing are examples of embodied agents (Steels, 2004). According to the authors in (Steels, 2004), a robot can be seen as a software agent controlling a physical body. For example, the author in (Wooldridge, 2009) describes the robot Stanley (i.e. an unmanned ground vehicle navigation developed by authors in (Thrun et al., 2007)) as “**an autonomous agent embodied in a car.**” The software architecture of Stanley's control system is organized into six layers. We provide a briefly description of the main layers below (Wooldridge, 2009):

- Sensor interface layer: Stanley is equipped with a wide range of sensors apparatus;
- Perception layer: Stanley's collected data is translated into the internal models used to represent Stanley's environment. The state of the ve-

hicle is modeled in terms of 15 variables, relating to position, velocity, orientation, accelerometer, and gyro;

- Planning and control layer: path planning, steering, and brake control;
- Vehicle interface layer: the interface between the control system and the actuator controls.

The author in (Wooldridge, 2009) says that the most of effort in building Stanley went into the perception layer, the layer responsible for making the relation between the sensors and the associated techniques to interpret sensor data. This complexity is not a particular problem of physical agents. The authors in (Russell and Norvig, 1995), for example, supports the idea of specifying which action an agent ought to take in response to any given percept sequence to provide a design for an ideal agent. However, they argue that to create this specifications could provide an infinite list for most agents.

In order to face the problem of perception, a strategy that has been commonly applied on the development of physical agents by robotic researchers is the use of artificial neural networks (i.e. an artificial intelligence technique to encode a mathematical function that establishes a relation between a set of inputs and a set of outputs. See a detailed description in the section 2.4) (Marocco and Nolfi, 2007; Nolfi and Floreano, 2000; Floreano and Mattiussi, 2008). In accordance with the benefits of using neural networks, the Laboratory of Artificial Life and Robotics (Parisi and Nolfi, 1994) (Nolfi, 1995) defines embodied agents as agents that have a body and are controlled by an artificial neural network. In order to provide self-organizing capabilities, these agents use adaptive techniques (i.e. evolutionary techniques) to adapt to their task environment.

Furthermore, according to the authors in (Zahedi and Ay, 2013) and in (Polani, 2011), the embodiment can be seen as a a two-way filter layer between the brain and the environment. First, the embodiment filters the external world and determines how the brain perceives it. Second, the embodiment translates commands emitted by the brain and expresses them as observable behaviors.

In Figure 2.4, we illustrated an embodied agent to be used in our applications, following the above description. According to this illustration, the body is a microcontroller connected to the Internet; it also contains some sensors and actuators. While the former allows the collection of environmental information, the latter allows the performance of some actions, such as to activate a specific environmental component and emit communication signals.

Through the Internet, the “brain” and body can communicate. A software agent contains the “brain” that is an encoded neural network. The neural

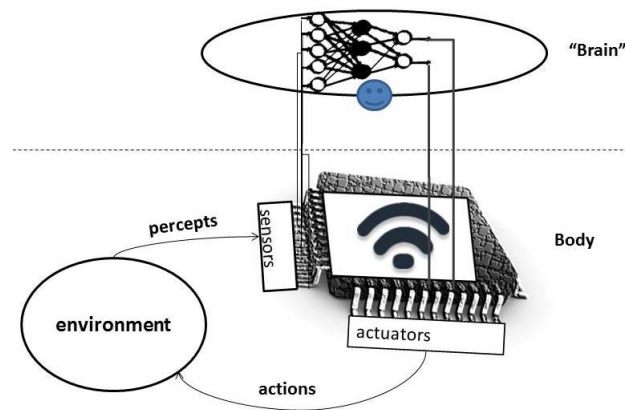


Figure 2.4: Modeling an Embodied Agent based on Figure 2.3 and the literature description provided in this section.

network outputs define the values to be set in the actuators. These outputs are calculated according to the data collected by sensors and the encoded configuration that the neural network is using.

2.3 Evolutionary Algorithms

Faced with the sophistication and adaptability provided by evolutionary biological systems, computer scientists and engineers have felt motivated to try to replicate these systems in the development of artificial systems. They have been seeking to solve problems that are difficult to be solved by analytical methods (Floreano and Mattiussi, 2008).

The artificial evolutionary algorithm is briefly defined as a collection of individuals in a search space, where each is a different solution to a given problem. A chromosome represents the individual, and the goal of the search is to identify the one with the best genetic material. We measure the quality (fitness) of each by a given fitness function, which measures how good that particular individual is among the ability of the entire population. The fittest individuals will have greater ability to reproduce and could thus result in the reduction of the least fit individuals.

The different individual sequence of genes can be in various formats as binary, character strings, numeric values, and others. We represent the genetic material of an individual, of that search space, by a vector m with p positions:

$p = x_1, x_2, x_3, \dots, x_m$, where each x represents a gene also known as solution variable.

Figure 2.5 illustrates a generic diagram of an evolutionary algorithm. First, it is necessary to form an initial search space, represented by the first generation.

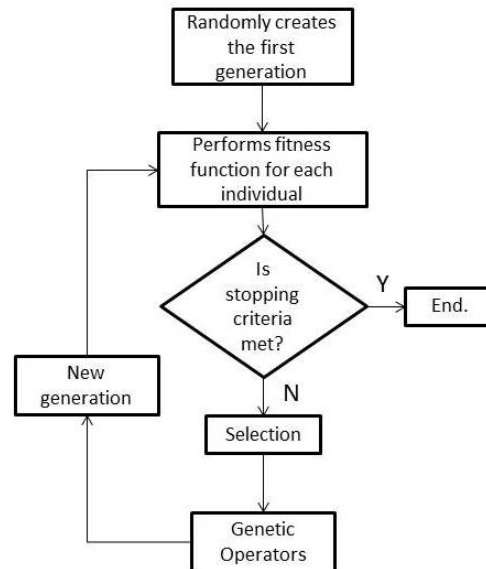


Figure 2.5: A generic diagram of an evolutionary algorithm.

According to the authors in (Floreano and Mattiussi, 2008), the size of the population for experiments that require interaction with a real environment normally is less than one hundred. Typically, the genotype of the first individuals are created randomly. The goal of this process is to ensure diversity for the initial generation. In the case of binary representation, for example, each genotype is created by a random sequence of 1s and 0s. The quality of the generation is measured based on an individual fitness average.

While the algorithm does not find a generation to meet the stopping criterion, new generations are being formed. The stopping criterion can be an individual performance criteria, by fitness function, the average fitness of all individuals, or the range of the maximum number of generations formed during the evolutionary process.

Natural selection preserves the best-adapted individuals of a generation, giving them a greater chance to procreate. Following this concept, the algorithm should select the best parents of the current generation, so that they can transfer their genetic material to the next generation. Thus, the first step of the reproduction process is the selection. There are several ways to perform this selection process. Among the best known are the roulette, selection by a tournament, and by ranking (see (Floreano and Mattiussi, 2008)).

To avoid the loss of the best solution during evolution, one can use the popular strategy of elitism. Elitism is a strategy of a composition of a new population. The strategy maintains the n best-selected individuals in the new generation, so that, one of their children have a copy of their genetic material.

When a generation does not meet the stopping criteria, among all possible

solutions to the problem it addresses, probably the best of their solutions is not enough to solve it. Thus, it is necessary to apply genetic operators, so that it is possible to introduce diversity in the population, and thus can insert new solutions that search space. One of the operators is the crossover, which generates a new individual from the combination of genetic material from two or more individuals parents. The other is the mutation that generates new individuals with a slightly changed genetic code from small random changes caused by their genotype. For more details about the different types of selection and genetic operators, see (Floreano and Mattiussi, 2008).

2.4 Artificial Neural Network (ANN)

There are some living organisms, such as the paramecium, that does not have a neural network. They can eat, move toward the light and escape from predators (as a simple reactive agent)(Floreano and Mattiussi, 2008). However, bodies with a neural network have at least two advantages: selective transmission of signals among body parts (input-output mapping (Haykin, 1994)), and adaptation through synaptic plasticity (Floreano and Mattiussi, 2008). The block diagram of Figure 2.6 illustrates the input-output mapping process of a human nervous system. It receives information from the environment, perceives it, and makes appropriate decisions (Haykin, 1994). According to the authors in (Haykin, 1994), “plasticity permits the developing nervous system to adapt to its surrounding environment.” As a result, the interest of modeling the operation of the neural system to build intelligent machines has been increasing.

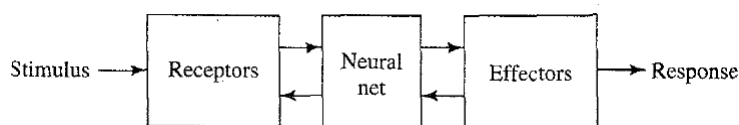


Figure 2.6: Block diagram representation of nervous system (Haykin, 1994). Pg.28.

The authors in (Haykin, 1994) provide the following definition of a neural network viewed as an adaptive machine:

A neural network is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

- Knowledge is acquired by the network from its environment through a learning process.
- Inter-neuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.

Systems that use an Artificial Neural Networks (ANN) have been presenting other properties beyond adaptivity, such as pattern recognition, perception, and motor control (Haykin, 1994). An ANN consists of neurons and connections (as well as in a biological neural network). To understand how a neural network functions is necessary to understand these elements.

2.4.1 Artificial Neuron

According to the author in (Haykin, 1994), a neuron is an information-processing unit that is fundamental to the operation of a neural network. Figure 2.7 shows the model of a neuron, which forms the basis for designing artificial neural networks.

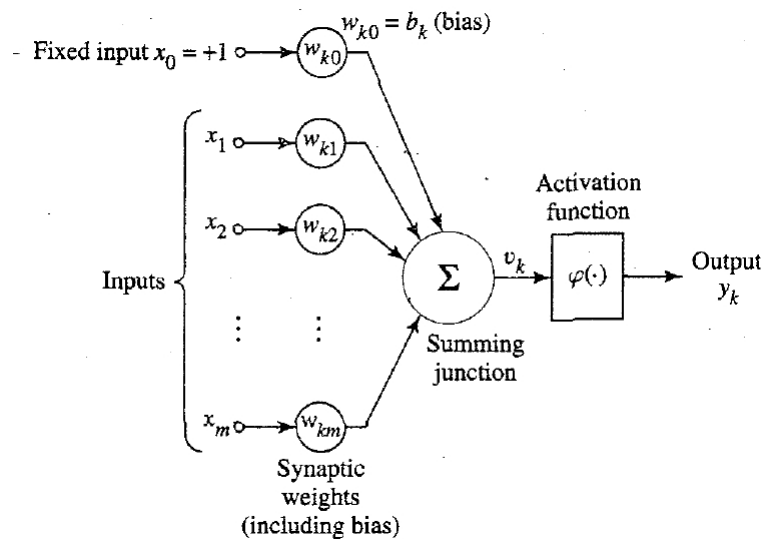


Figure 2.7: The model of a neuron (Haykin, 1994). Pg.33.

This model identifies some basic properties of a neuron, as shown (Haykin, 1994):

- A set of **synapses** or connecting links. Specifically, a signal x_m at the input of synapse m connected to neuron k is multiplied by the synaptic weight w_{km} . Unlike a synapse in the brain, the synaptic weight of an artificial neuron may lie in a range that includes negative as well as positive values.

- An **adder** for summing the input signals, weighted by the respective synapses of the neuron. The adder can add excitatory inputs or subtract inhibitory inputs from other neurons connection (McCulloch and Pitts, 1943).
- An **activation function** for limiting the amplitude of the output of a neuron. Typically, the amplitude range of the output of a neuron is normalized and written as a unit closed interval $[0,1]$ or alternatively $[-1,1]$.
- An externally applied **bias**, denoted by b_k . The bias b_k has the effect of increasing or lowering the net input of the activation function, depending on whether it is positive or negative, respectively. As shown in Figure 2.7, the effect of the bias is accounted by adding a new input signal fixed at $+1$, and adding a new synaptic weight equal to the bias b_k .

In mathematical terms, Haykin (1994, pg.33) proposes the following equations to describe a neuron k :

$$u_k = \sum_{j=1}^m w_{kj} \times x_j \quad (2-1)$$

$$v_k = u_k + b_k; \quad (2-2)$$

$$y_k = f(v_k); \quad (2-3)$$

Where x_1, x_2, \dots, x_m are the input signals; $w_{k1}, w_{k2}, \dots, w_{km}$ are the synaptic weights of neuron k ; u_k is the linear combiner output due to the input signals; b_k is the bias; $f(\cdot)$ is the activation function; and y_k is the output signal of the neuron.

The activation function, denoted by $f(\cdot)$, defines the output y of a neuron k in terms of the induced local field v . According to Haykin (1994), the sigmoid function is by far the most common form of activation function used in the construction of artificial neural networks. The equation 2-4 presents an example of a sigmoid function.

$$y_k = \frac{1}{(1 + e^{-v_k})}; \quad (2-4)$$

See Haykin (1994, pg. 34) for more information about different types of activation functions.

2.4.2 Network Architectures

An ANN is an arrangement of neurons. There are different classes of network architectures (e.g. competitive networks, which can make use of a

Winner-Takes-All algorithm(WTA), recurrent, feedforward, etc. See a discussion about them in (Haykin, 1994)). A widely used type is the feedforward network. They contains an input layer and an output layer. The first consists of sensory neurons that receive environmental stimuli. The second consists of motor neurons, which are responsible for producing the network response (Floreano and Mattiussi, 2008). A network may have one or more hidden layers, which are composed of hidden neurons. The function of hidden neurons is to intervene between the external input and the network output in some useful manner (Haykin, 1994). These neural networks are commonly referred to as multilayer perceptron (Haykin, 1994). Figure 2.8 illustrates a layout of a multilayer perceptron with two hidden layers.

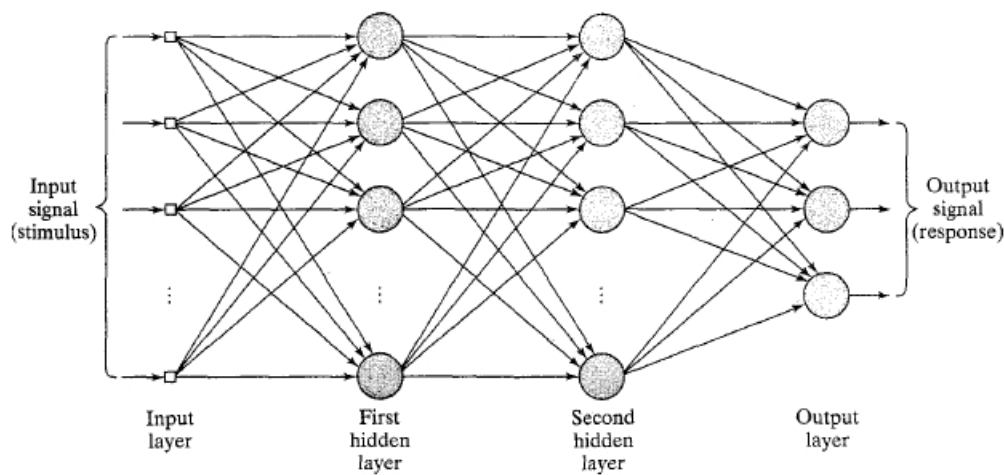


Figure 2.8: Multilayer feedforward network with two hidden layers (Haykin, 1994). Pg.181.

To determine the number and the size of the hidden layers is mostly a matter of trial and error. However, there are heuristic techniques to establish an optimal number of hidden neurons (Haykin, 1994). For example, the author in (Kuurkova, 1992) proposes a technique to derive estimates of numbers of hidden units based on properties of the function being approximated and the accuracy of its approximation. Alternatively, some researches (Miller et al., 1989; Belew et al., 1990) have been using evolutionary algorithms to design the network topology automatically.

Another type of network class is the recurrent network. Recurrent networks distinguish themselves from feedforward networks in that they have at least one feedback loop. Feedback refers to a dynamic system whenever the output of an element in the system influences in part the input applied to that particular element (Haykin, 1994), as shown in Figure 2.9.

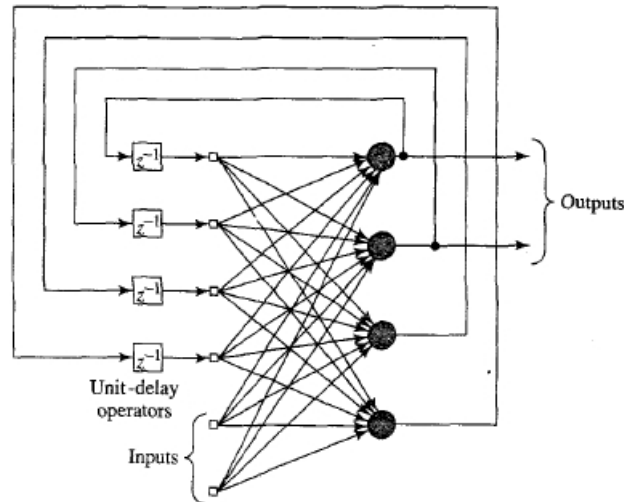


Figure 2.9: Recurrent network (Haykin, 1994). Pg.46.

2.4.3 Adaptive Process

According to the authors in (Flozano and Mattiussi, 2008), adaptation is a major feature of the nervous system. This allows the body to modify and develop behaviors in order to maintain or improve their probability to survive in dynamic and partially unknown environments. According to the authors in (Yao, 1999), learning and evolution are two fundamental forms of adaptation. There has been a great interest in combining learning and evolution with artificial neural networks (ANNs) in recent years.

2.4.3.1 Learning Algorithm

According to Haykin (1995, p.46), “a major task for a neural network is to learn a model of the environment in which it is embedded.” The learning algorithm used to train the network is directly linked with the neural network structure. After the learning step, the knowledge representation of the surrounding environment is defined by the values taken on by the free parameters (i.e., synaptic weights and biases) of the network (Haykin, 1994).

A learning algorithm can be supervised or unsupervised. In supervised algorithms, the neural training process is performed using labeled examples. In such cases, each example representing an input signal is paired with a corresponding desired response. Algorithms for unsupervised learning, or self-organized learning, do not provide a set of input-output pairs (Haykin, 1994). According to Haykin (1994, pg. 414), “the purpose of an algorithm for self-organized learning is to discover significant patterns or features in the input data, and to do the discovery without a teacher.” According to the authors

in (Floreano and Mattiussi, 2008), the use of an evolutionary algorithm is an alternative or complementary technique to unsupervised learning algorithms for adapting a neural network. We describe a particular evolutionary self-organized algorithm in section 2.4.3.

An algorithm that is commonly used to train multilayer perceptrons is the error back-propagation algorithm (Haykin, 1994). Basically, error back-propagation learning consists of two steps: a forward and a backward steps. In the first one, an activity pattern (input vector) is applied to the sensory nodes of the network. As a result, a set of outputs is produced. During the forward step the synaptic weights of the networks are all fixed. During the backward step, the synaptic weights are all adjusted in accordance with an error-correction rule (Haykin, 1994). The adjustment of the synaptic weights of the neurons in accordance with the error signal leads to an adaptive process (Haykin, 1994). See a detailed description about back-propagation algorithm in Haykin (1994, pg. 183).

According to Haykin (1994, pg. 200), there are some methods to improve the back-propagation algorithm's performance, such as to normalize the inputs. However, he agrees with the following statement: "the design of a neural network using the back-propagation algorithm is more of an art than a science in the sense that many of the numerous factors involved in the design are the results of one's own personal experience."

2.4.3.2 Evolutionary Neural Networks

Similar to unsupervised learning algorithms, evolutionary algorithms have been commonly used for adapting neural networks without a teacher (Floreano and Mattiussi, 2008; Floreano and Urzelai, 2000). According to the authors in (Floreano and Mattiussi, 2008), there are at least two reasons for using a evolutionary algorithm instead of a learning algorithm: 1) there are no restrictions on the type of architecture; and 2) it is not necessary to take an input-output mapping into account.

Evolutionary algorithms (EA) have been used to train neural networks, to select features, and to determine the topology of the network. For the evolution of a neural network, its characteristics are encoded in artificial genomes. A genome is usually represented as a string of real or binary values, and evolved according to a performance criterion. If the goal is only to train the neural network, the genotype will encode only the value of synaptic weights (Floreano and Mattiussi, 2008). The interested reader may consult more extensive papers (Miller et al., 1989; Belew et al., 1990; Yao, 1999).

By using an evolutionary algorithm, a weight sequence represents the

genotype of an individual. One generation consists of a pool of individuals that represent different network configurations (see the description of evolutionary algorithms in section 2.3). Figure 2.10 illustrates a multilayer feedforward network and two candidates (individuals) for its weights sequence. We are supposing that the weights are real values and can be written as a unit closed interval $[-3,3]$.

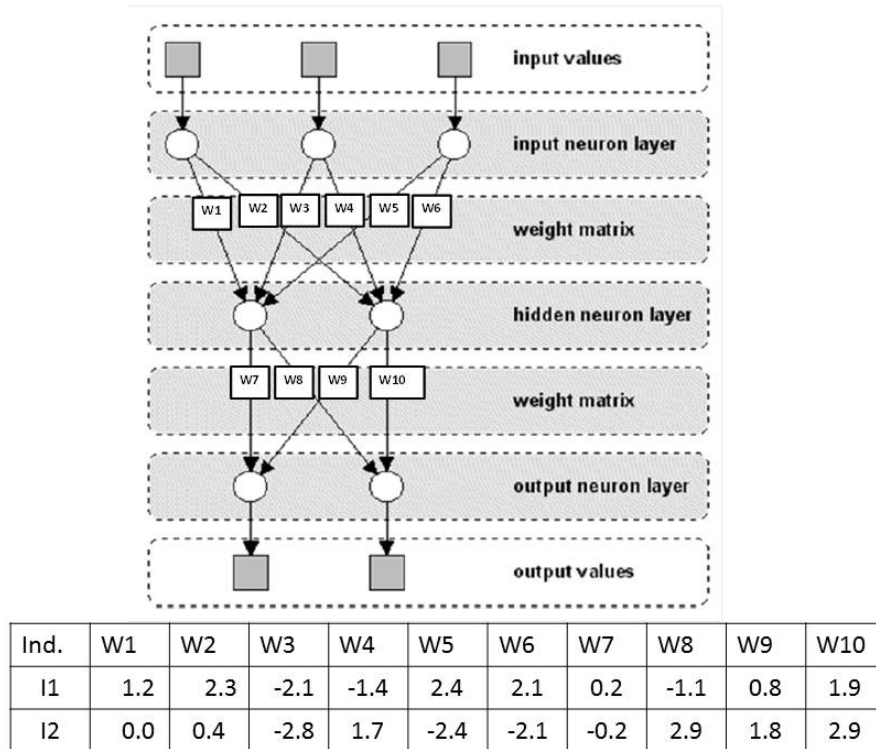


Figure 2.10: Evolving a neural network. Adapted from Floreano and Mattiussi (2008, p.317).

For each weight sequence candidate, the algorithm evaluates the network performance. The better individuals are selected to reproduce and create the next generation.

2.5 Evolutionary Robotics

The use of evolutionary algorithms to train neural networks is commonly applied to develop self-organizing robot swarms that are examples of embodied agents (Floreano and Urzelai, 2000). By the Robotic Agents Literature, this approach is known as Evolutionary Robotics (ER) (Marocco and Nolfi, 2007; Nolfi and Floreano, 2000; Floreano and Mattiussi, 2008; Massera et al., 2013; Nolfi and Parisi, 1997; Nolfi and Floreano, 1998; Floreano et al., 2007). The primary goal of ER is to provide ways of automatically synthesizing intelligent

autonomous robot systems (Nolfi and Floreano, 2000). According to the authors in (Nelson et al., 2007), the evolutionary robotic has the potential to lead to the development of robots that can adapt to uncharacterized environments, which may be able to perform tasks that human designers do not thoroughly understand.

Most of these experiments presents a population of agents equipped with a sensory-motor system, including wireless sensors, and a cognitive apparatus, which evolves a self-organizing communication system and use their communication abilities to solve a given problem (VIDE and Nolfi, 2006). An example is the work by Marocco and Nolfi entitled “Emergence of Communication in Embodied Agents Evolved for the Ability to Solve a Collective Navigation Problem” (Marocco and Nolfi, 2007). Similarly, authors in (Floreano et al., 2007) established an experimental system with colonies of ten robots that could forage in an environment containing a food and a poison source. This study showed that ER has a potentially critical role in evolving a sophisticated communication systems in collections of embodied agents, and in designing efficient groups of cooperative agents.

2.5.1 Evolving Embodied Agents

Based on the description of Evolutionary Robotics algorithm provided by authors in (Nolfi and Floreano, 2000), we prepared a diagram to illustrate the process of evolving embodied agents through the ER method, as depicted in Figure 2.11.

There are two ways of evolving agents’ neural network controller at runtime: embodied evolution and evolution in simulation with transfer to reality (Nelson et al., 2007). The former uses physical devices during the evolutionary process: the encoded configurations of neural network are loaded into embodied agents’ microcomputers, they are tested, and the associated candidates’ fitness are evaluated based on the performance of the real devices. By using physical devices during the evolutionary process could be so slow in real-time; otherwise this procedure insures that the controllers can function in real devices. In addition, the evolutionary training process can produce bad configurations for the neural network, and consequently generate a serious problem in particular cases such as a traffic urban controller. An alternative to embodied evolution is to evolve the controllers in simulated agents living in simulation environments (Nelson et al., 2007), and then transfer the evolved neural network to physical agents that are situated in a real world.

The ER method provides the emergence of features that have not been de-

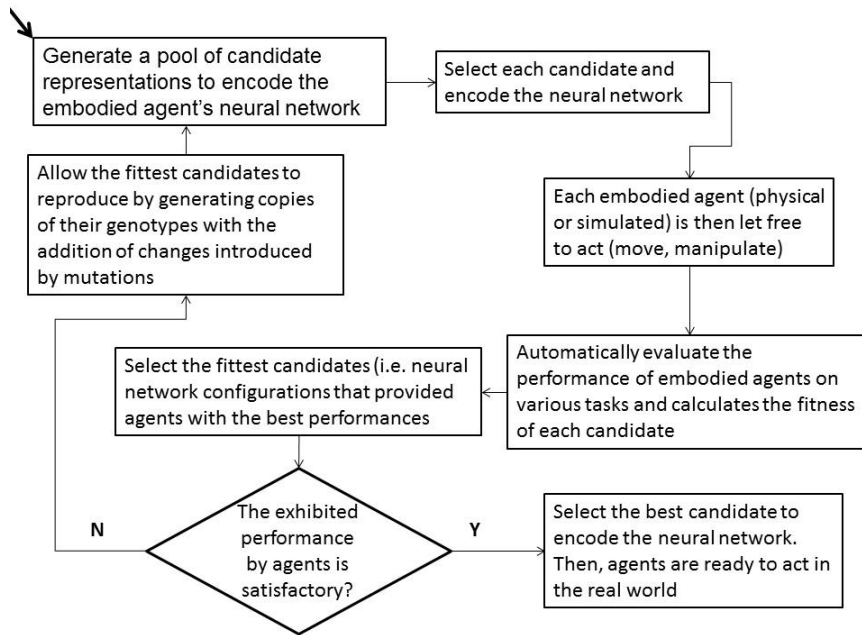


Figure 2.11: Evolving Embodied Agents.

fined at design-time, including a sophisticated self-organizing communication system. While in traditional approaches the desired behaviors are accomplished intuitively by the designer, in evolutionary robotics this is often the result of an adaptation process that usually involves a larger number of interactions between the system and the environment (Nolfi and Floreano, 2000).

2.6 Self-adapting and Self-Organizing Systems

The increasingly computerization of our environment has led software engineering to look for inspiration in diverse fields, such as robotics, artificial intelligence, or biology to find new ways of designing and managing systems (Babaoglu and Shrobe, 2007). As a result, self-adaptation (SA) and self-organization (SO) have emerged as two promising interrelated approaches (Babaoglu and Shrobe, 2007; Babaoglu and Shrobe, 2015). The goal of self-organizing and self-adaptive systems (SASO) is to reduce operation and design cost in the development of dynamical systems (Hudson and Denzinger, 2014).

There is a growing conference series to investigate self-organizing and self-adaptive systems, which is in its ninth edition (Babaoglu and Shrobe, 2015). We consider here the definitions of self-adaptive and self-organizing systems that have been used by this conference editors, as follows (Babaoglu and Shrobe, 2007; Babaoglu and Shrobe, 2015):

Self-adaptive systems work in a top-down manner. They evaluate their own global behavior and change it when the evaluation

indicates that they are not accomplishing what they were intended to do, or when better functionality or performance is possible. Such systems typically operate with an explicit internal representation of themselves and their global goals.

Self-organizing systems work bottom-up. They are composed of a large number of components that interact according to simple and local rules. The global behavior of the system emerges from these local interactions, and it is difficult to deduce properties of the global system by studying only the local properties of its parts. Such systems do not use internal representations of global properties or goals; they are often inspired by biological or sociological phenomena.

As explained by the editors in (Babaoglu and Shrobe, 2007), self-adaptive systems are top-down and self-organizing are bottom-up. Thus, these systems seem to be different and the creation of self-adaptive and self-organizing (SASO) systems resulted in new challenges (Hudson and Denzinger, 2014). For example, “how are intermediate-level structures formed which leverage micro-level behavior to achieve desirable macro-level outcomes?” (Babaoglu and Shrobe, 2015).

The authors in (Serugendo et al., 2007) discuss a generic framework for the development of self-adaptive and self-organizing systems. They listed some key requirements to provide a system with self-* properties and behaviors (e.g., self-organizing, self-adapting, etc. See the description of more self-* properties in (Horn, 2001)). We selected some of these requirements to use as a basis for the development of our model, as follows (Serugendo et al., 2007):

- Autonomous individual components: self-* behaviors arise from autonomous system components’ interaction. In SO systems, such components can be agents, peers or cars. In SA systems, such components can be autonomic managers and any element of the supporting infrastructure.
- Self-awareness: Self-* properties arise from the capability of the system or individual components to sensor an environment and identify by themselves any new condition, failure or problem. Self-awareness requires “sensing” capabilities and triggers “reasoning” and “acting.” SO systems sense their environment in different ways (e.g., configurations and neighbours) and take decisions accordingly (e.g., changing role or directions). SA systems are provided with monitoring, planning and plan execution capabilities.

- Behavior guiding and bounding: SO systems have their own local rules. SA systems have both local and global rules. For guiding the system towards optimal functioning, it is important to limit the system actions, but freely allowing decentralized adaptive behavior of individual components inside the boundaries.

To meet these requirements and develop a self-adaptive and self-organizing system, the authors in (Serugendo et al., 2007) propose the use of a set of technologies and software architecture infrastructures (see (Serugendo et al., 2007)). In particular, we are interested in the proposal of integrating a **control and feedback loop** and a **Coordination** modules. The former is based on the IBM control loop (2004), which is an architecture to provide systems with autonomy and self-awareness (Jacob et al., 2004), as shown in Figure 2.12.

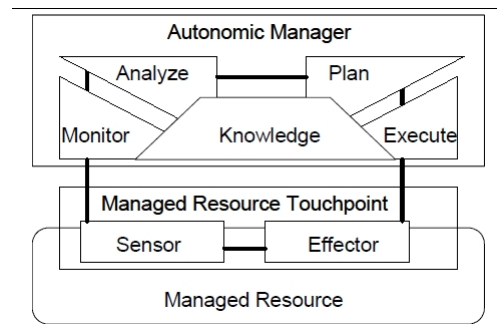


Figure 2.12: IBM Control Loop to generate autonomous elements (Jacob et al., 2004). Pg 24.

Based on notions of (M)onitoring, (A)nalyzing, (P)lanning (also Decision Making), (E)xecuting decision and a (K)nowledge base, the IBM control loop is known as MAPE-K. It has become a base model from which it is possible to extend self-adaptive (Neto et al., 2009) and/or self-organizing systems (Valadares et al., 2013). The management tasks involved in the MAPE-K model are related to: (i) monitoring of the managed components; (ii) analyzing the collected data, translating and aggregating the unstructured data into local knowledge; (iii) planning adaptation actions depending on the environment variation; and (iv) executing the decided plans upon the environment. An autonomic element can interact with other autonomic elements via sensors and effectors.

The proposal of IBM is to create elements completely autonomic and without the need of any external administrator intervention (Valadares et al., 2013). Meanwhile, the authors in (Müller-Schloer, 2004) affirm that to design self-organizing systems with pure decentralized control may become unmanageable. In order to provide these systems with the capability of

adapting at run-time, and preventing or healing from the occurrence of undesired emergent behavior, they also require to being open for intervention (i.e. an administrator access).

Thus, the Coordination module proposed by the authors in (Serugendo et al., 2007) provides these systems with a behavior guiding and bounding policies feature. They consider that all elements share metadata and policies, which results in a form of overall control.

This module is based on the “Observer-Controller” paradigm (Müller-Schloer, 2004). The key role of an “Observer” component is to monitor events and states. A “Controller” component is responsible to take actions whenever the observer part results let it consider appropriate (Müller-Schloer, 2004). As a result, the control and feedback loop can dynamically reconfigure the system components and change their perception of the collected data.

To illustrate the use of their architecture (Serugendo et al., 2007), the authors present two proof-of-concept applications. First, they present an application to a self-adaptive system. The goal of this system is to dynamically allocate resources between components. Second, they consider the application of their framework to a self-organizing system. They present a traffic light controller as an example of a bottom-up system with several independent components communicating and taking decisions on the basis of individual collected data. They aim at showing how the global property (to maximize traffic throughput) is broken down and implemented into local rules.

In this dissertation, we propose to combine MAS, neural networks, IBM control loop, and an instance of the “Observer-controller” paradigm in a architecture to address the key self-organizing and self-adaptive systems requirements listed above.

MAS are widely used to model self-organizing and/or self-adaptive systems (Hudson and Denzinger, 2014; Weiss and Sen, 1995; Neto et al., 2009; Briot et al., 2006). An agent has some characteristics, such as autonomy and social ability, which make MAS suitable for the development of SASO systems. To provide agents with adaptation capacity, we create a control based on the IBM control loop. However, we propose to substitute the analyze and plan modules of the IBM control loop for a neural network. As we will see, neural networks have a great generality and may be used for classification problems as well as for control problems.

Finally, we provide an Observer agent to store the neural network, which represents the common knowledge representation, and perform the adaptation process. Chapter 3 provides a more detailed description about our proposed architecture.

2.7

MAS for IoT

We first present an overview of frameworks in the literature while addressing the idea of mixing Artificial Intelligence and Internet of Things concepts, especially those with focus on multi-agent systems. We also describe a framework for physical agents that not have focus on IoT, but which provides physical systems with self-organizing and self-adaptive properties.

We know of few research efforts in the literature about smart objects for Internet of Things applications. The authors in (Fortino et al., 2014) aim at providing a clear picture of the suitability of middlewares to support the development of Smart Objects-based Internet of Things systems. The main SO middlewares have been described and compared in their paper based on a set of requirements for smart environments and objects.

Finally such authors listed four middlewares which provide efficient management to develop and deploy SOs: ACOSO (Agent-based Cooperating Smart Objects) (Fortino et al., 2012; Fortino et al., 2013), FedNet (Kawsar et al., 2010), Ubicomp (Goumopoulos and Kameas, 2009) and Smart Products (Mühlhäuser, 2008). They make use of different architectural models: the ACOSO is agent-oriented and event-driven, the FedNet is service-oriented, while Ubicomp and Smart Products are component-based.

However, authors in (Fortino et al., 2014) also discuss the limit of these middlewares in the management of a vast number of cooperative SOs, since none of them presented a case study to demonstrate its efficiency in wide scenarios. According to the authors, to develop SO-based IoT systems, novel software engineering methodologies for extreme-scale dynamic systems need to be defined. It is also necessary to include specific abstractions able to deal with system/component evolution that is a typical property of SO systems. The authors argue that agent-oriented methodologies could be exploited by engineers as the basis for formalizing such an effective development method for SOs. Multiagent Systems were widely employed to cope with the main requirements for IoT systems: interoperability, abstraction, collective intelligence and experience-based learning.

The work performed in (Lopez and Pérez, 2012) also proposes a framework for the IoT based on a multiagent System Paradigm. In this sense, the authors listed some requirements needed for developing IoT applications. This list gives support to the domain analysis of our proposed framework. According to authors, requirements are the acquisition of measurements and data from devices, its processing and translation of a context of useful information, and actuation over the environment. Moreover, the approach showed that agents

have characteristics that are suitable for those requirements, such as perception, autonomy and social ability. Despite the fact that the paper presents a real motivation to our approach, it only offers a brief description of a framework components. The authors have also mentioned that there is still the need for detailing every component and give them the intelligent characteristics. Our approach provides intelligence components to develop IoT applications through adaptation and organization algorithms, since we agree that it is crucial to model this type of application.

A framework developed by the Italian Institute of Cognitive Science and Technologies (Pezzulo et al., 2011), and used to support FIoT is the Framework for Autonomous Robotics Simulation and Analysis (FARSA) (Massera et al., 2013). This framework was created to assist research in the area of embodied cognition, adaptive behaviour, language and action. A set of works on Evolutionary Robotics (Marocco and Nolfi, 2007; Nolfi and Parisi, 1997; Nolfi and Floreano, 1998; Massera et al., 2013) were developed using FARSA or related software. Most of these experiments presents a group of embodied agents that evolves for the ability to solve a collective problem.

Another related work to our approach is the framework presented in (Sobe et al., 2012), Framework for Evolutionary Design (FREVO). The authors in (Sobe et al., 2012) presents Frevo as a multi-agent tool for evolving and evaluating self-organizing simulated systems. The authors affirm that Frevo allows a framework user to select a target problem evaluation, controller representation and an optimization method. However, it concentrates only on evolutionary methods for agent controllers. As a result, this tool can only provide offline adaptations and evolve only simulated environments. In addition, it is often applied in the creation of autonomous robots.

Unfortunately, we can not reuse these referred platforms to control smart objects since it they are very oriented towards the simulation of robotic agents. Furthermore, these platforms have limited communication structure since they do not give support to heterogeneous platforms required by current networks, such as desktop, web, mobile and microcontroller boards.

3

FloT: Framework for Internet of Things

In this chapter, we first perform a survey of Internet of Things requirements taking previously published and personal experiences into account. Then we describe our proposed agent-based model to create IoT systems and we show how this model meets these requirements. Our proposed model consists of three layers: (i) physical, (ii) communication, and (iii) application. To facilitate the development process of the communication and application layers of an IoT system, we developed the Framework for Internet of Things (FIoT). Therefore, we also present the FIoT in this chapter.

During the development of a framework, three stages must be considered: (i) domain analysis, (ii) framework design, and (iii) framework instantiation (Markiewicz and Lucena, 2001). A domain analysis stage provides a survey of domain requirements. In the framework design stage, we used the Unified Modeling Language (UML) diagrams (Beydeda et al., 2005) and the Astah Tool (Vision, 2011) to specify FIoT structure, behavior, and architecture. UML use case (Maßen and Lichter, 2002) and UML activity diagrams (Dumas and Hofstede, 2001) are used to assist the description of the main idea of FIoT. In addition, we present the FIoT UML class diagram (Markiewicz and Lucena, 2001), followed by the analysis of its kernel (“frozen-spots”) and flexible points (“hot-spots”) (Fayad et al., 1999). “Frozen-spots” are immutable and must be part of each framework instance. “Hot-spots” represent the flexible points of a system, which must be customized in order to generate a specific application (Markiewicz and Lucena, 2001).

According to the authors in (Markiewicz and Lucena, 2001), the abusive use of hot spots in a framework design will inevitably lead to complex software systems. Therefore, the framework designer has to choose the hot spots carefully, neither exaggerating nor creating a far too generic framework.

The instantiation stage is presented in the Chapter 4, performing the generation of new instances through implementation of the FIoT’s hot spots.

3.1 Domain Analysis

As we emphasized in the chapters 1 and 2, we used the works (Fortino and Trunfio, 2014) and (Lopez and Pérez, 2012) as basis of our domain analysis. We also consider the requirements for the development of self-organizing and self-adaptive applications proposed by the authors in (Serugendo et al., 2007).

From an engineering perspective, IoT systems are distributed systems consisting of components (things) that may be physical devices, animals or people. As we aim at giving support to the development of smart things, these components have to autonomously collect data about themselves and their environments and take actions (Kuniavsky, 2010). A smart IoT system can make decisions based on sensed data and use dynamic reconfiguration to improve its performance.

All IoT applications share common features, such as to connect and collect data; but they have different features that vary according to specific application scenarios. To assist the development of self-organizing and self-adaptive IoT applications, we performed a discovery domain requirement (R), as follows:

- R1. Design-time description (problem domain):
 - R1.1 To analyze environmental conditions that are associated with the problem goal (e.g., temperature, gases)
 - R1.2 To define how to collect environmental conditions (e.g., a microcontroller board and sensors)
- R2. Decentralization and Interoperability.
- R3. Autonomous things:
 - R3.1 Things should be capable of autonomously sensing/monitoring themselves and their environments
 - R3.2 Actuation over the environment
- R4. Self-adapting capability:
 - R4.1 The individual components or the whole system should be capable of identifying any new condition, failure, or problem by themselves/itself
 - R4.2 Run-time capability of reasoning and of acting/adapting
- R5. To design a software to allow the system:
 - R5.1 To recognize devices in the environment;

- R5.2 To acquire the data from devices that are collecting environmental data;
- R5.3 To interface with device sensors; and
- R5.4 To process and translate the data to a context of useful information.

In the next subsections, we show how our proposed model and framework meet the requirements listed above.

3.2 Agent-Based Model

We developed an agent-based model to be used as a basis for generating different kinds of applications for Internet of Things. Our approach is completely based on Artificial Intelligence paradigms, such as multi-agent systems, neural networks and evolutionary algorithms. We aim at providing mechanisms to automatically recognize and manage things in the environment.

As depicted in Figure 3.1, our model consists of the design of three layers: (L1) physical, (L2) communication, and (L3) application. Each device in the environment (physical layer) can be recognized and controlled by agents in the application layer.

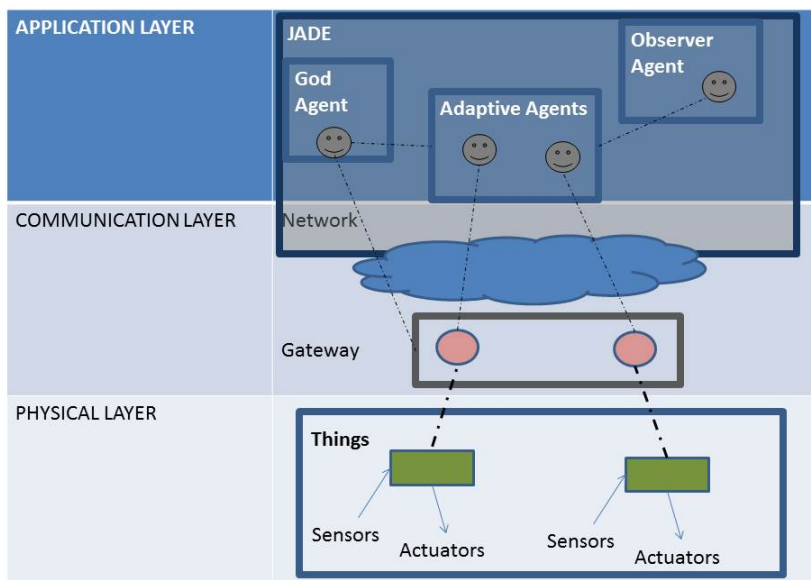


Figure 3.1: An agent-based model to generate IoT applications.

The physical layer consists of simulated or real devices (also named as things/objects) and environments. In order to model the physical layer, the project designer has to define the features of things as well as the features of the environment in which these things are situated. He must raise the

environmental conditions that need to be monitored (e.g. temperature, relative humidity and car flow). Then he can make specifications for devices. For example, to set their sensors and actuators (i.e. the necessary technology to collect data or make changes on the environment).

The communication layer defines that communication among agents on the application layer and devices must occur using the Internet. Each thing has one address, so an agent can access this address to get and set the necessary information to control the device. We suggest the Java Agent Development Framework (JADE) (Bellifemine et al., 2007) (Bellifemine et al., 2010) to implement the communication among agents and things. JADE implements the Foundation for Intelligent Physical Agents (FIPA) protocol for agent communication. It allows the development of an interoperability communication structure, which agents can execute on different platforms and exchange information (see Section 2.2.1).

The application layer uses a Multi-Agent System (MAS) to provide services, such as collecting, analyzing and transmitting data from several sensors to the Internet and back again. We aim at providing decentralization, autonomy and self-organizing features to applications through MAS. In addition, we provide the capacity of creating physical agents capable of interacting dynamically with complex environments by using approaches from Evolutionary Robotics (section 2.5).

We suggest developing controllers at the application layer to allow autonomous management of devices in the physical layer.

3.3

Central Idea for the Framework Design

A FIoT application consists of the development of three kinds of agents: (i) God Agents (Riel, 1996); (ii) Adaptive Agents; and (iii) Observer Agents (Müller-Schloer, 2004). The activity diagrams of these agents will be presented in this section in order to specify agents workflows. We used darker blocks in activity diagrams to represent the activities that vary from application to application. The other activities are performed independently of target applications. This analysis was performed to facilitate the further identification of hot and frozen spots at the class diagram design-time.

The activities of the God Agent are presented on the diagram depicted in Figure 3.2. The primary role of the God Agent is to detect new Things that are trying to connect to the system. Thus, the God Agent allows the automatic connection of new devices to the system, as a “plug and play” connection. The “plug” in this example, means that the device needs to send a message to

the God Agent's IP address, excluding manual settings. For each connected device, the God Agent creates an Adaptive Agent to control it. An Adaptive Agent is an agent embodied in a Thing, according to the description provided in Section 2.2.2. While a device represents its body, a JADE software agent contains its controller. The God Agent sets the controller (i.e. the “brain” of the agent) for the Adaptive Agent according to the type of its device (e.g. the number of sensors and actuators). Therefore, the controller creation is a flexible point on FIoT system implementation.

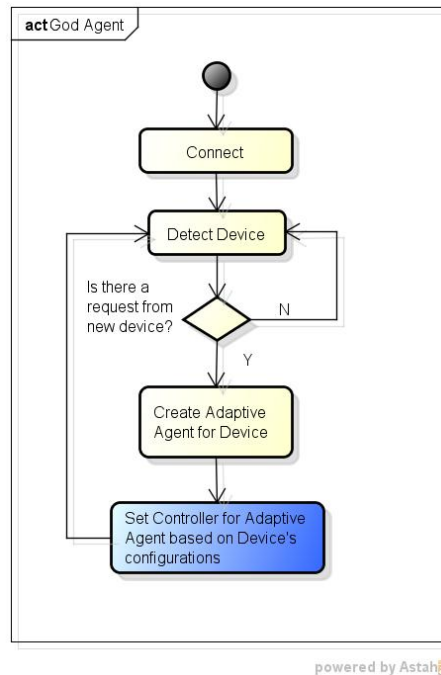


Figure 3.2: Activity diagram of GodAgent.

Authors in (Neto et al., 2009) developed a framework to implement self-adaptive software agents based on the autonomic computing principles proposed by IBM (section 2.6). In order to create adaptive agents, they provided a control loop composed of four activities: collect, analyze, plan and execute (see Figure 3.3).

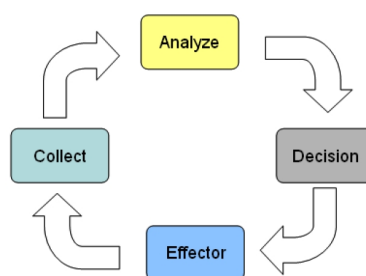


Figure 3.3: Control loop provided by framework in (Neto et al., 2009).

What follows is a brief description of each of these four activities:

- **Collect:** to collect application data;
- **Analyze:** to analyze those data by trying to detect problems;
- **Plan:** to decide what should be done in the case of problems; and
- **Execute:** to change the application due to executed actions.

We customized the control loop used in their work to define the behaviors of the FIoT's Adaptive Agents. Instead of executing the analyze and plan activities, the FIoT's Adaptive Agents make decisions based on a controller, which can be, for example, a finite state machine (FSM) or an artificial neural network (ANN), as shown in Figure 3.4.

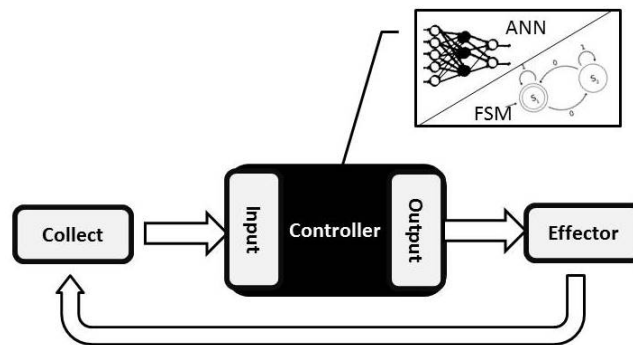


Figure 3.4: Control loop provided by the FIoT framework.

Therefore, our Adaptive Agent must execute a sequence of three key activities: (i) collect data from the thing; (ii) make decisions; and (iii) take actions. The task of data collection focuses on processing information coming from devices, such as reading data from input sensors. These collected data are used to set the inputs of the agent's controller. Then, the controller processes a decision to be taken by the agent.

Adaptive Agents act according to the controller output. An action (effector activity) can be to interact with other agents, to send messages, or to set actuators data of devices, allowing them to make changes to the environment. As shown in Figure 3.5, the developer does not need to make changes to the implementation of an Adaptive Agent since it behaves independently of the device it is monitoring.

The Observer Agent aims at allowing Adaptive Agents to cope with the changing (dynamic) environments, and at making them capable of adapting to the unexpected. By using an adaptive approach, we expect the emergence of features that have not been defined at design-time, including a sophisticated self-organizing system.

Some researchers (Floreano and Mattiussi, 2008; Nolfi and Floreano, 2000; Dorigo et al., 2004; Trianni and Nolfi, 2011; Quinn et al., 2003; Panait

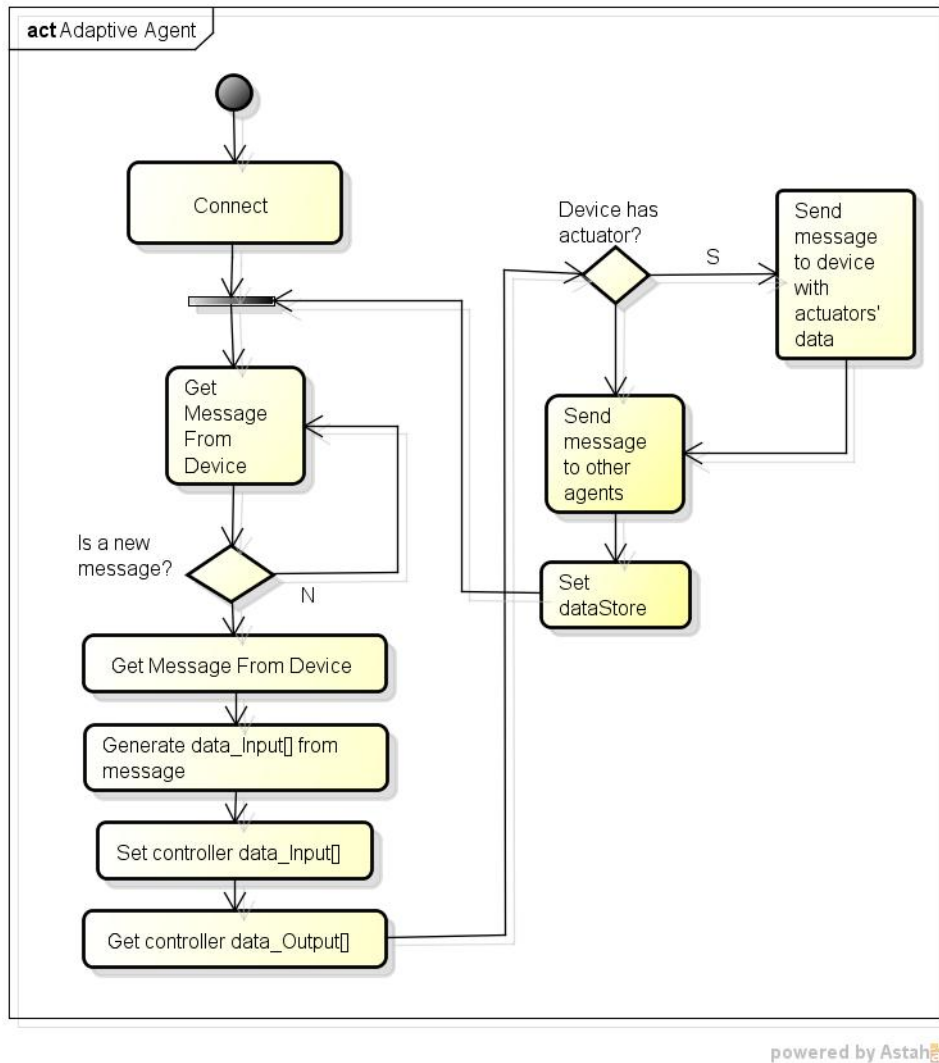


Figure 3.5: Activity diagram of the Adaptive Agent.

and Luke, 2005) investigate the emergence of cooperative or competitive self-organizing multi-agent systems. One of the specifications to generate a cooperative self-organizing multi-agent system is to conduct the adaptive process according to collective evaluations. Self-organizing systems have global goals. Thus, we aim at investigating during the adaption process if a collection of agents are attending together the global goal or not. If the system needs to adapt, the adaptation is performed for the whole multi-agent system. If we conduct the adaptive process according to individual evaluations, the agents may compete with each other. This is the main reason that we provide an Observer Agent to evaluate the global behavior of the collection of Adaptive Agents and to conduct the adaption process of the whole system. Therefore, its main goal is to verify if the Adaptive Agents need to adapt or not. When the actions of agents are far from what it expects, the Observer Agent executes a supervised or unsupervised learning method, such as backpropagation or

genetic algorithm.

The process of adaptation consists of generating new configurations for Adaptive Agents' controller and test how agents will behave in the environment. The Observer Agent selects the configuration used when the collection of Adaptive Agents shows a desired global action to set their controller. While the Observer Agent looks for the new controller configuration, Adaptive Agents continue their execution normally.

As shown in Figure 3.6, the Observer Agent is tightly coupled to the application being developed. The evaluation process has to be implemented according to the expected global solution. Another variable activity is the generation of new configurations for controllers. It depends on the applied adaptive technique.

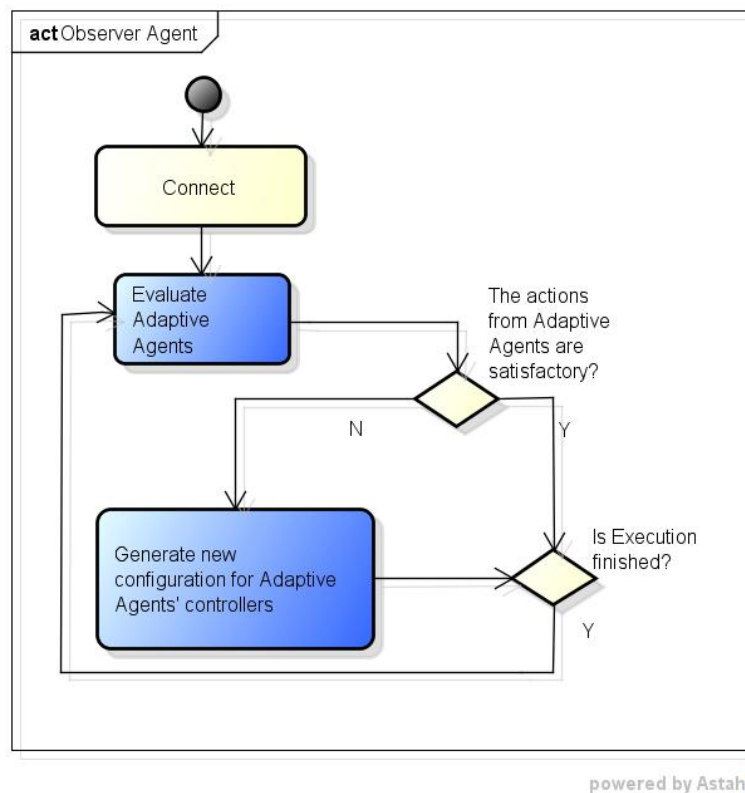


Figure 3.6: Activity diagram of ObserverAgent.

As agents execute specific activities to virtualize things and communicate with them at the physical layer, we also created an activity diagram for devices, as shown in Figure 3.7.

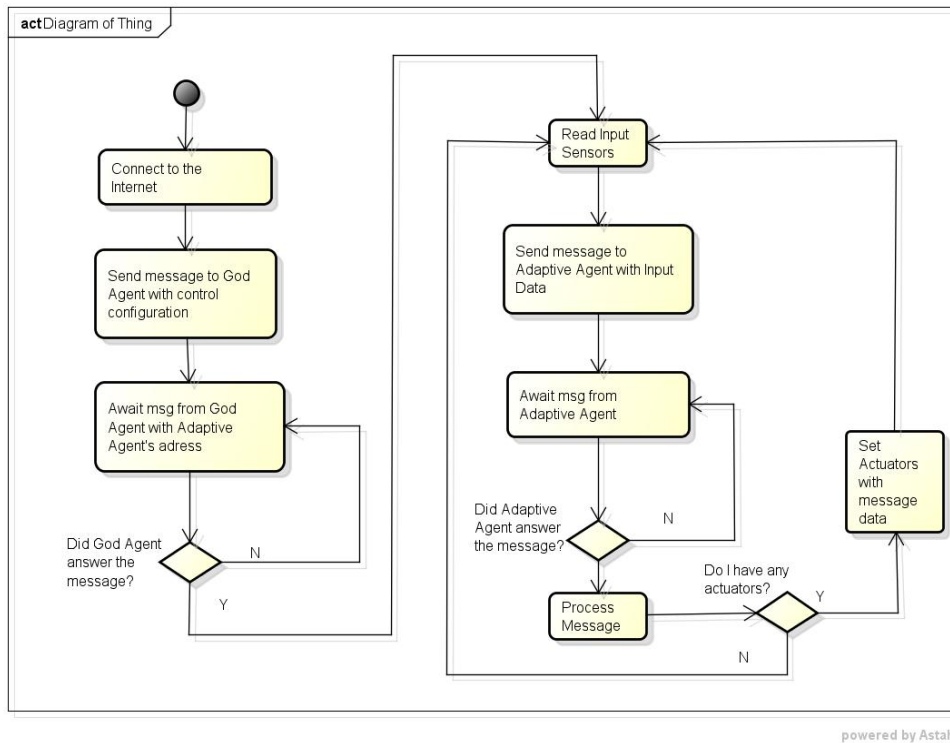


Figure 3.7: Activity diagram of a Thing.

Things in the physical layer must execute the following sequential activities:

- Connect to the Internet
- Send message to the GodAgent, reporting your type of controller. The GodAgent has some controllers already registered. Thus, the type of controller indicates the characteristics of a device, such as the list of sensors and actuators.
- Wait message from GodAgent containing the address of its AdaptiveAgent. Then, the thing will use this address to send and receive the next messages in a cycle:
 - Send message with data sensors
 - Wait message with data to set its actuators.

Table 3.1 summarizes the model and framework description in this section, and presents how they meet the requirements listed in Section 3.1, according to their layers. FlOT meets the requirements associated with the layers of communication (L2) and application (L3).

Table 3.1: How the model and FIoT meet the IoT requirements.

Req.	Layer	Description
R1	L1	The physical layer design defines the problem domain.
R2	L2 and L3	The application layer uses a Multi-Agent System (MAS) to provide services. The interoperability can be supported by the JADE Framework.
R3	L3	Adaptive agents autonomously manage devices, without the need of a human administrator.
R4.1	L3	The Observer Agent can evaluate the whole system, groups of Adaptive Agents, or each individual.
R4.2	L3	The adaptive process can be acquired through the execution of a supervised or an unsupervised learning algorithm at run-time. If the Adaptive Agents are not performing a desired behavior, the Observer Agent can execute a learning algorithm at run-time to adjust the parameters of the Adaptive Agents' controllers.
R5.1	L2 and L3	The God Agent automatically identifies things that are trying to connect to the system.
R5.2	L2 and L3	Adaptive Agents collect data from the things at the physical layer
R5.3	L1 and L3	Adaptive Agents have access to a set of sensors and actuators previously registered. The things at the physical layer inform of which type they are. Then, the Adaptive Agents know how to process the data sent by the things.
R5.4	L3	Adaptive Agents are intelligent agents that make use of a controller to process the data coming from the devices

3.3.1 Use Case Diagram

This subsection presents the Use Case Diagram of FIoT (Eriksson, 2006), as depicted in Figure 3.8. Our objective is to provide an initial design of the framework. The previous sections explained the roles of each actor and presented activity diagrams to describe the use case scenarios.

The microcontroller package is not part of the FIoT system; otherwise it represents the interaction between a thing and the application controller. A device can collect data from the environment (use case "Read Input") and change the value of its actuators (use case "Set Output"). A thing has to be capable of sending data to and reading data from the Internet.

The system has some common and alternative features. The type of controllers and the techniques to be used to adapt the controllers are examples of alternative use cases. For example, the use case "Adapt" can extend the use cases "Run Genetic Algorithm" or "Run Backpropagation". If the system executes one of these use cases, the adaptive algorithm may make changes on the controller parameters. If the system developer chooses to work with neural network and genetic algorithm, the use case "Run Genetic Algorithm" will set the weight sequence to be used by the neural network.

The use case "Compute Output" can also make changes on controllers. For all types of controllers, to compute an output is necessary to set the data

input parameter.

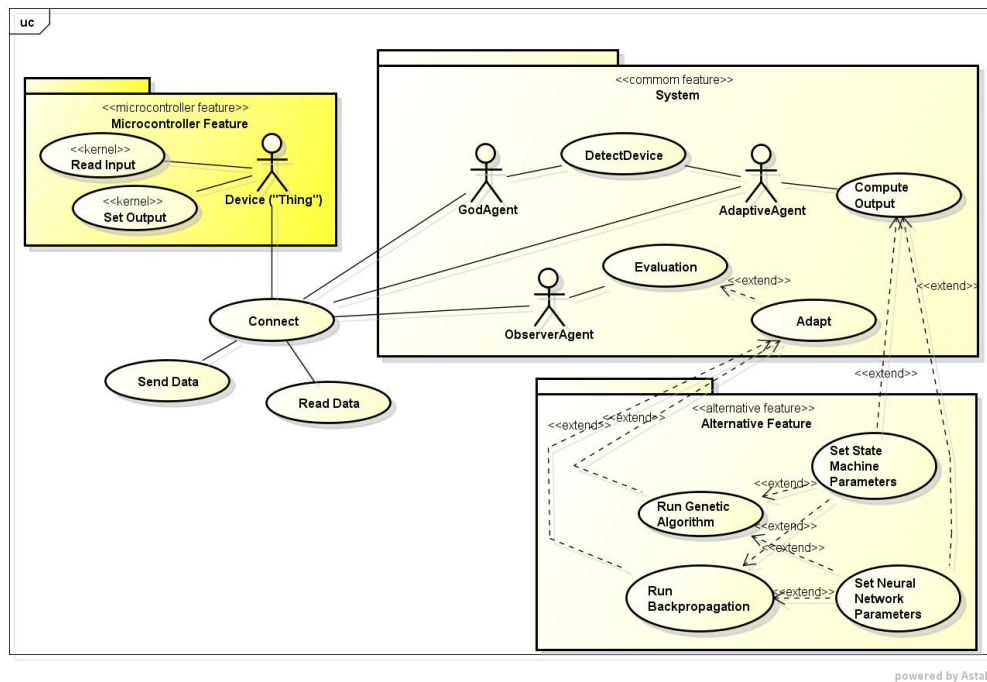


Figure 3.8: Use Case Diagram.

3.4 Details of FIoT

As presented in the section 3.2, our model proposes the use of JADE to support the communication among agents and things. FIoT extends JADE, a Java framework to implement multi-agent systems. Figure 3.9 shows the packages of the FIoT package.

The project consists of the development of JADE agents, the behaviors of agents, the controller to be used by Adaptive Agents, and the adaptive process to be executed by the Observer Agent. In addition, the system gives support to different interface communication message systems, such as sockets and ACL. We will present the main FIoT classes (Sommerville, 2004) of the main packages.

The class diagram depicted in Figure 3.10 illustrates the FIoT classes associated with the creation of agents and their execution loops. As described before, the FIoT agents are represented by the GodAgent, ObserverAgent and AdaptiveAgent classes, which extend the FIoTAgent class. Then, they can access and make changes to the list of controllers (ControllerList class). This list stores all controllers already created by the GodAgent for each thing type (e.g. chair with one temperature sensor, lamp with one presence sensor and one alarm actuator).

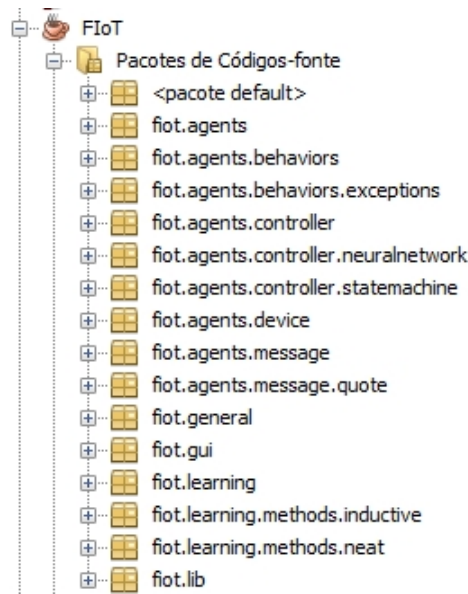


Figure 3.9: Packages of the FIoT project.

All agents execute sequential behaviors, named as ExecutionLoop: God-Loop, AdaptiveLoop and ObserverLoop classes. The sequential behavior is a type of JADE behavior that provides support to the implementation of composed activities (Bellifemine et al., 2007). Thus, the ExecutionLoop is a sequence of smaller actions. For example, for Adaptive Agents, these execution loops are composited of collect, decision and effector activities. We modeled the AdaptiveLoop according to the work presented in (Neto et al., 2009) that provides an adaptive structure based on the IBM controlloop (see Figure 2.12).

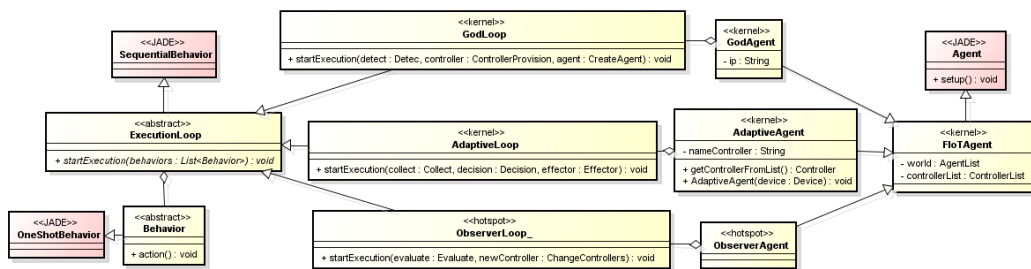


Figure 3.10: Class diagram of FIoT - Agents.

The class diagram depicted in Figure 3.11 illustrates the collection of behaviors already developed. Activities such as making evaluation and adapting are examples of hot spots. Therefore, new strategies for evaluation and adaptation can be developed to be used by agents.

While ObserverAgent access the ControllerList to adapt controllers configuration through ChangeControllers behavior, AdaptiveAgent uses it to get its controller, set data input, and then obtaining the calculated output.

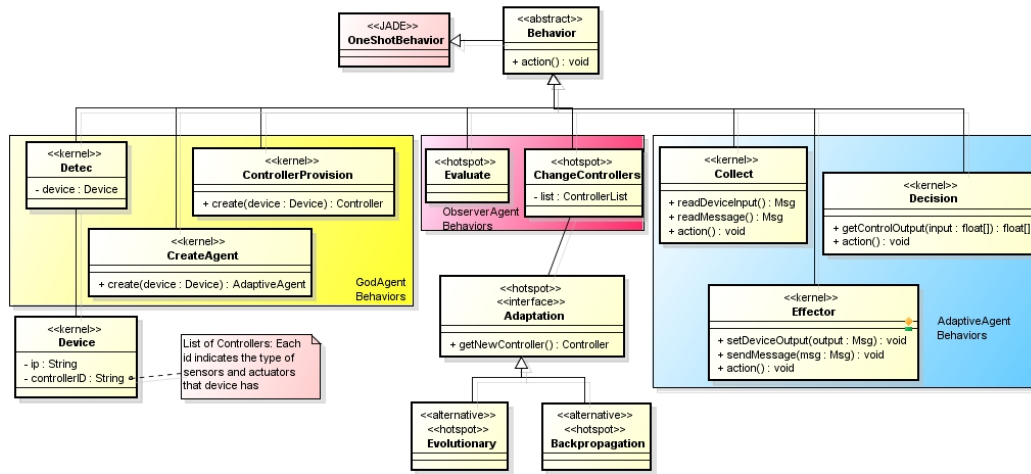


Figure 3.11: Class diagram of FloT - Behaviors.

The class diagram depicted in Figure 3.12 illustrates the controllers classes. Agents whereas virtualize homogeneous devices can use the same controller to make decisions. For example, in a scenario where similar smart lamps have to be managed, the same ANN controller can be used by Adaptive Agents. The GodAgent stores their controller in ControllerList as “lampNeuralNetwork”. If there is another group of devices, the GodAgent has to create a different controller for them.

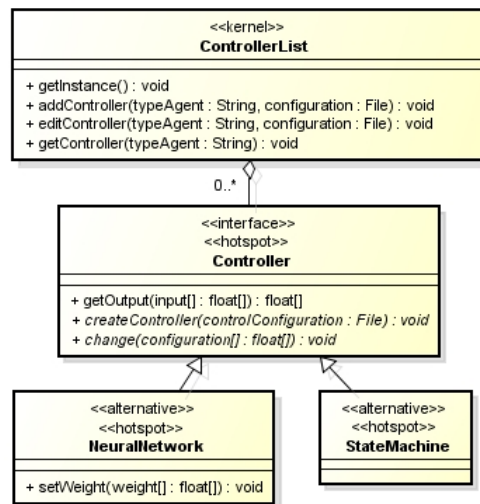


Figure 3.12: Class diagram of FloT - Controllers.

3.5 How to instantiate FloT: Technical Details

A framework is classified according to its extensibility; it can be used as a white box or a black box (Fayad et al., 1999; Markiewicz and Lucena, 2001). In white box frameworks, a instantiation is only possible through the creation of

new classes. Such way, a developer must understand the framework very well in order to produce an instance. A black box framework allows a developer to produce new instances using configuration scripts. This kind of framework creates the classes and source code for the chosen configuration automatically, which facilitates the instantiation process (Markiewicz and Lucena, 2001).

FIoT contains both white and black boxes characteristics. Thus, it is a gray box framework (Markiewicz and Lucena, 2001). In this section, we present how to create a instance via configuration files or creating and using a new class. By providing the possibility to use new classes, we aim at increasing the framework domain coverage. A framework user has the option to implement and use other types of controllers and adaptive techniques.

To create an instance of FIoT, the first step is to specify the type of controller to be used by Adaptive Agents. A developer can use an implemented one or develop another. If he decides to develop a new one, he will create a new Controller type class. This new class must implement the major Controller class methods, as shown bellow:

Listing 3.1: The major methods of the Controller class.

```
public interface Controller {
    ...
    public double[] getOutput(double[] input);
    public void change(double[] configuration);
    public Controller create(File file);
}
```

All controllers have to implement the method `getOutput()` that provides an output after receiving an input. Adaptive Agents use this method on decision activity. The `ObserverAgent` accesses the controller list during the adaption process to change the configuration of the controller used by Adaptive Agents. Listing bellow illustrates part of the codification of `Observer` and Adaptive agents' controlloop behaviors.

Listing 3.2: Parts of the `ObserverLoop` and `AdaptiveLoop` behavior codes.

```
//OBSERVERLOOP
if(geneticAlgorithm){
    ...
    for (int cont = 0; cont < numberOfPopulation; cont++) {
        ...
        Individuo indi = listIndividuos.get(cont);
        double[] genes = indi.getCromossomo().getGenes();
        //Access the Controller List to change the weights of the
```

```

        controller used by Adaptive Agents
    this.listControl.getController(
simulation.getControllerInUse()).change(genes);
    ...
}}
-----
//ADAPTIVELOOP
    ...
    double input[] = msgFromDevice.getContent(); //Receiving
        sensor data from the device
    //getting a controller in the Controller List
    Controller control =
        ControllerList.getInstance().getController(
    this.getDevice().getControllerID());
    double[] output = control.getOutput(input);
    ...

```

Executions differ from each other based on how the current controller implements the common methods of the Controller class. The code below presents the implementation of a three-layer network controller.

Listing 3.3: Implementing a new controller.

```

import fiot.agents.controller.Controller;
public class ThreeLayerNetwork implements Controller{
    public ThreeLayerNetwork(File file){..}
    ...
    @Override
    public double[] getOutput(double[] input){

        for (int i = 0; i < layers.length; i++) {
            if (i > 0) {
                //hidden and output layers
                layers[i].setEntryNumber(
                    layers[i-1].getNumberOfNeurons());
                int numW = layers[i].getNumberOfNeurons()*
                    layers[i-1].getNumberOfNeurons();
                layers[i].setNumberOfWeights(numW);
                layers[i].setWeights(this.weights[i]);
                layers[i].setEntradas(layers[i-1].getOutputs());
                layers[i].getLayerOutput();
            } else {
                //input layer

```

```

        layers[i].setNumberOfNeurons(this.numInputNeurons);
        layers[i].setOutputs(input);
    }
}
return this.layers[this.layers.length-1].getOutputs();
}

@Override
public void change(double[] configuration) {
    //update weights
    weights = new double[layers.length][this.numWeight];
    int numWeight = 0;
    for (int i = 1; i < layers.length; i++) {
        int numW = layers[i].getNumberOfNeurons()*
            layers[i-1].getNumberOfNeurons();
        for(int cont = 0; cont<numW; cont++){
            weights[i][cont] = configuration[numWeight];
            numWeight++;
        }
        layers[i].setWeights(this.weights[i]);
    }
}
}

```

After creating a new controller, the developer must insert it into the ControllerList to be accessed by FIoT agents, as follows:

Listing 3.4: Making use of a new controller.

```

import fiot.agents.controller.ControllerList;
import application.controller.neuralnetwork.ThreeLayerNetwork;

public class FIoTApplication{
    ...
    ControllerList listControl ;
    ThreeLayerNetwork controlLayer = new ThreeLayerNetwork(file);
    listControl.addController(controlLayer.getNameType(),controlLayer);
}

```

Then, a new controller is available to be used by Adaptive Agents. If a developer decides to make use of techniques already available on FIoT, he has to provide a technical description of them. He can make it through a configuration text file, as shown in Figure 3.13. This figure illustrates two examples of controllers' description.

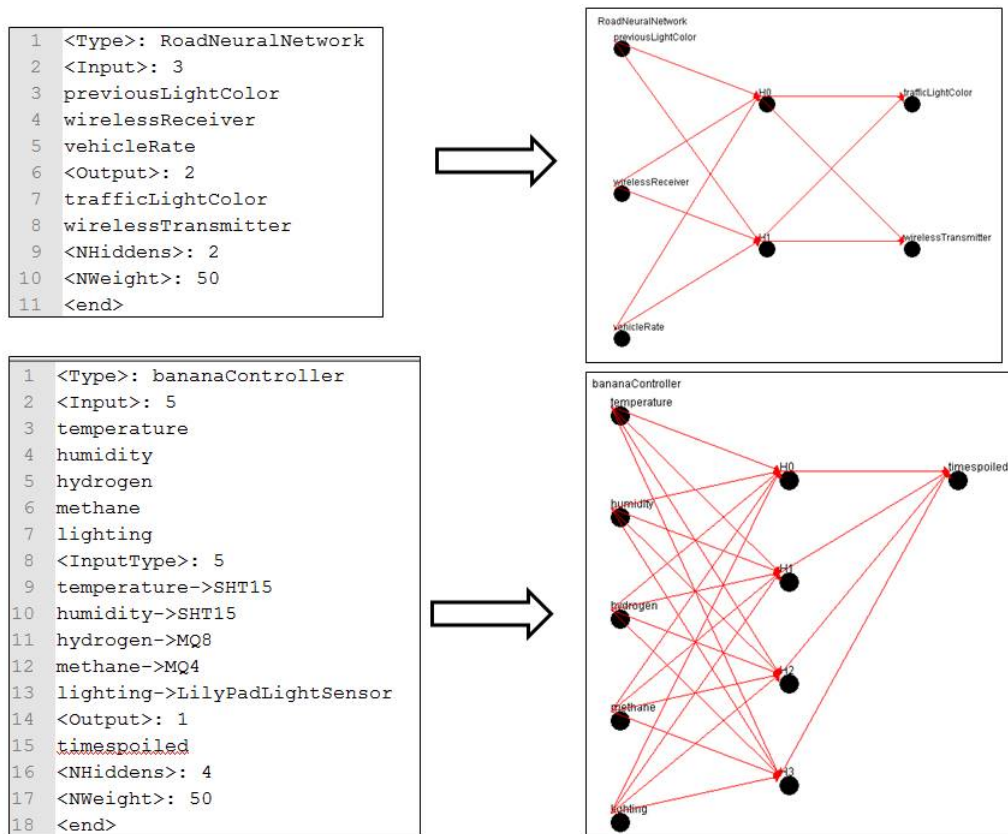


Figure 3.13: Examples of controllers registered by the system developer via text files.

A controller can be a neural network, a finite state machine, or only a simple “if statement” code. Independently, it must have inputs and outputs. A finite state machine, for example, consists of states, input and outputs. It can produce an output sequence according to the input sequence (Lee and Yannakakis, 1996). If the controller is a neural network, the designer will also need to set the number of hidden neurons and connections among neurons. In addition, the description allows the Adaptive Agent to know which type of sensors and actuators the device is using and to use a specific logic to process and translate the collected data.

The ObserverAgent is an extensible feature of the framework. However, FIoT provides a default adaptive class that allows the ObserverAgent to execute a genetic or a backpropagation algorithm. The developer can change the execution of these algorithms via configuration files, as shown in Figure 3.14.

After creating the controller and adaptive method, the developer can configure the application execution. The code below illustrates the configuration parameters to be set before start an IoT application using FIoT. As shown, the ApplicationConfiguration class is one of the FIoT classes (package “general”). First, it is necessary to set the type of controller and adaptive


```

1 <numberOfGeneration>:40
2 <numberOfTests>:2
3 <numOfBestToBeSelected>:20
4 <numOfChildren>:4
5 <numberOfPopulation>:100
6 <elitismSaveFitness>:1
7 <maxNumGenes>: 10
8 <valueMaxOfGene>: 5
9 <rateMutation>:10
10 <fitness>:1

```

Figure 3.14: Configuring the execution of a genetic algorithm.

technique to be used by the application. Then, the developer must provide an IP for the GodAgent before starting the GodAgent and ObserverAgent via the startExecution method.

Listing 3.5: Configuring a FIoT application.

```

import fiot.general.ApplicationConfiguration;
...
public class ApplicationControl {
    ApplicationConfiguration configuration =
        ApplicationConfiguration.getInstance();
    String namecontroller;
    String nameMethod;
public ApplicationControl(){
    this.namecontroller = "Three Layer Network";
    this.configuration.setNameController(namecontroller,
        namecontroller);
}

public void setControllerFile(File file) throws IOException{
    this.configuration.createController(this.namecontroller,
        file);
}

public void setMethodFile(ObserverAgent ob, File file) throws
IOException{
    this.nameMethod = "Back-propagation";
    this.configuration.setLearningMethod(this.nameMethod,
        file);
    this.configuration.setObserverAgent(ob);}

public void startExecution(){
    String ip = "192.168.43.112"; //GoDAgent's IP
    this.configuration.startMainAgents(ip, "SOCKET");
}

```

The importance to define a name for each type of controller is because the device uses this name to communicate to the GodAgent its type. According to the controller's description, the GodAgent automatically build the controller to be used by an Adaptive Agent. The code below is the Arduino code used by an example of device. The first step is to send its identification to the God Agent, as shown:

Listing 3.6: An example of Arduino code: The device informs to the GodAgent that it requires a "bananaNeuralNetwork" controller type.

```
String remote_server = "192.168.43.112"; // GodAgent's IP address
String remote_port = "9876";

// Initialize client with IP address and port number
WifiClient client(remote_server, remote_port, PROTO_UDP);
if (client.connect()) {
    // Send message over UDP socket to peer device
    client.println("bananaNeuralNetwork"); //its controller type
    Serial.println("receiving message from God: ");
    waitMessageGod();
}

void waitMessageGod(){
    String msgFromGod="";
    ...
    //GodAgent provides the port number of a new AdaptiveAgent.
    //Then, the device connects to this AdaptiveAgent
    WifiClient client(remote_server, msgFromGod, PROTO_UDP);
    Wireless.begin(&wireless_prof);
}
```

4

Evaluation: Illustrative Examples

We evaluate FIoT by implementing the flexible points of it to generate different applications. As discussed in the chapter 3, the framework instantiation is the last stage to be considered during the development of a framework (Markiewicz and Lucena, 2001).

We consider the following IoT instances in FIoT evaluation process: (i) Quantified Things and (ii) Smart City. For each one, we developed an illustrative example, and this chapter presents a brief description, the experimental setup, and the evaluation result of them. In addition, we present the initial design of a future instance, a Quantified Us experiment.

Section 4.5 presents how the generated applications adhere to the proposed framework, filling the main variable parts.

4.1

FIoT's Instances

The frozen spots are part of FIoT kernel. Then each of the proposed applications will have the following modules in common:

- Detection of devices by the GodAgent;
- The assignment of a controller to a particular Adaptive Agent by the GodAgent;
- Creation of Agents;
- Data Collection execution by Adaptive Agents;
- Making decision by Adaptive Agents;
- Execution of effective activity by Adaptive Agents;
- The communication structure among agents and devices.

Some features are variable and may be selected/developed according to the application type, as follows:

- Controller creation;
- Making evaluation by the ObserverAgent;

- Controller adaptation by the Observer Agent.

Therefore, to create a FIoT instance, a developer has to implement/-choose: (i) a control module (e.g. neural network, finite state machine); (2) an adaptive technique to train the controller; and (iii) an evaluation process (e.g. genetic algorithm performs evaluation via fitness function). As shown in this chapter, we only evaluate applications using a neural network. However, we implemented FIoT to give support for the use of finite state machines (fsm), since we provided an abstract controller class. A framework user can implement a Mealy machine (a special case of a fsm), for example, and use an evolutionary algorithm to evolve its structure and transition probabilities (Pintér-Bartha et al., 2012; Sobe et al., 2012).

To support the construction of new applications via FIoT, Figure 4.1 depicts different configurations that can be selected before creating an instance.


Instance	Evaluation	Adaptation	Controller Creation
	Prediction Evaluation OR Performance criterion	Genetic Algorithm (Evolution) OR Back-propagation (Supervised Learning) OR No adaptation	Neural Network OR State machine

Figure 4.1: Different configurations to create FIoT’s instances.

As shown above, it is possible to generate applications using different configurations. A framework user needs to select a configuration that works better for solving a given problem. It is also possible to create an application without adaptation. For example, the user framework can start the system making use of a previously trained neural network, or only making use of an “if statement” code to represent the controller. In such case, the Observer Agent will not perform any activity.

4.2 Quantified Things

The “Quantified” movement is emerging via the Internet of Things (IoT). The most popular concept is the “Quantified Self” (QS) (Swan, 2012), based on which individuals use sensors and monitoring devices to quantify their health and behavior (Swan, 2013). Some researchers recently started to refer to “Quantified Us” (Lupton, 2014) and “Quantified Community” (Barrett et al.,

2013) as a way of articulating how the small data produced by self-trackers could be usefully incorporated into large data sets to provide new insights about collective features.

IoT aims at empowering not only anyone but also anything to connect to the Internet and to collect data about itself and the environment in which it is situated. As such, the question arises as to what happens if we use the “Quantified Self” and “Quantified Community” concepts to quantify a thing and groups of things. Instead of asking, “What can **people** learn when pooling data among themselves?” (Havens, 2014), we start to ask, “What can **things** ‘learn’ when pooling data among themselves?”. Thus, we propose a new branch of the “Quantified” movement, named “Quantified Things” (QT). In section 4.2.1, we will further analyze works in the “Quantified” area.

The things that can be quantified range from fruits to factory machinery. The former represents the “Quantified Fruit” experiments: storing fruit, collecting data about fruit and environmental conditions, sharing it on the Internet, and providing an advanced informative perspective about fruit shelf life. The latter represents the “Quantified Factory Machinery” experiments and part of the Industry 4.0 (Fantana et al., 2013), when one machine predicts a fault based on the history of other faults. Another possible scenario is to quantify a bean plantation, when the owner of such plantation predicts the crop yield based on the history of other crops.

By using our approach, we aim at providing the creation of many applications, such as “Quantified Fruit,” “Quantified Factory Machines,” “Quantified Bean Plantation,” among others. To illustrate the use of our proposed model, we derived an example from one of the “Quantified Things” applications: “Quantified Fruit.” Using the “Quantified Fruit” concept, some fruit storage use sensors to monitor environmental conditions, such as temperature, relative humidity, lighting and some gases that may affect fruit ripening. In turn, they populate a cloud database with collected data. By using these data, the system predicts how many days it takes for a fruit to spoil under specific environmental conditions.

The remainder of this section provides a brief review of the “Quantified” movement, performs a survey of specific “Quantified Things” requirements, and describes how the FIoT’s model can be adapted to create “Quantified Things” applications. Then, we discuss how the proposed model can be used to design and to implement “Quantified Things” instances. In addition, it presents an experimental setup.

4.2.1

A Brief Review of the “Quantified” Movement

Several projects were implemented to build knowledge bases for different kinds of problems (Boe and Salunkhe, 1967) (Mpelkas and Kenyon, 1972) (Yoshida et al., 1989) (UCI, 2014). These involved applying factor analysis to experimental data in order to define problem hypotheses and extract rules for each hypothesis. For example, we know of some approaches proposed to understand the fruit ripening process (Boe and Salunkhe, 1967) (Mpelkas and Kenyon, 1972). They were designed to investigate the factors (e.g. conditions of temperature, relative humidity, gases and lighting) affecting ripening and spoilage of fresh fruits and vegetables.

One of the key issues is how to build a knowledge base. Since design and population of a knowledge-based expert system usually rely upon a series of expensive and long-duration experiments, very laborious and time-consuming testing is required.

The “Quantified” movement appears as an alternative to the creation of these datasets, since it seeks to automate the data collection and interpretation process. As pointed out in (Barrett et al., 2013), the objects have been autonomously collecting data about themselves and their environments, and voluntarily sharing this information over the Internet.

A lot of systems have been developed in the “Quantified” context. The Quantified Self website (Quantified Self Guide to self-tracking tools, 2014) (Labs, 2014) lists over 500 self-tracking tools. However, few works (such as those performed in (Li et al., 2010) and in (Rivera-Pelayo et al., 2012)) present a conceptual model to assist the creation of new “Quantified” approaches. Currently, there is no unifying framework that clusters and connects these many emergent tools with the goals and benefits of their use.

Not all of existing “Quantified” works involve a reflexive monitoring subject. We know of conceptual works for “Quantified Cars”(Swan, 2015) and “Quantified Travelers” (Jariyasunant et al., 2011) and an application for “Quantified Babies” (Gaunt et al., 2014). For example, the “Quantified Babies” app lets parents measure and track their babies. However, they extend only the Quantified Self concept. They do not use the idea of knowledge sharing among things and a collective-level analysis yet.

4.2.2

Modeling “Quantified Things”

As “Quantified Things” applications are kind of IoT applications, a “Quantified Things” model has to meet the requirements imposed in the

section 3.1. However, some additional features are particular for these kind of applications, as follows:

- R6. To create a dataset to maintain the collected and processed data;
- R7. To process a dataset and use a pattern recognition mechanism;
- R8. To provide a graphic interface to allow a human administrator to monitor and analyze data.

The Adaptive Agents have to execute the mandatory activities of collection, decision, and action. The effector task of the Adaptive Agents in “Quantified Things” applications consists of making predictions based on collected data. They use an artificial neural network (ANN) to perform predictions, as depicted in Figure 4.2. It could also use another machine learning approach (Ugulino et al., 2012).

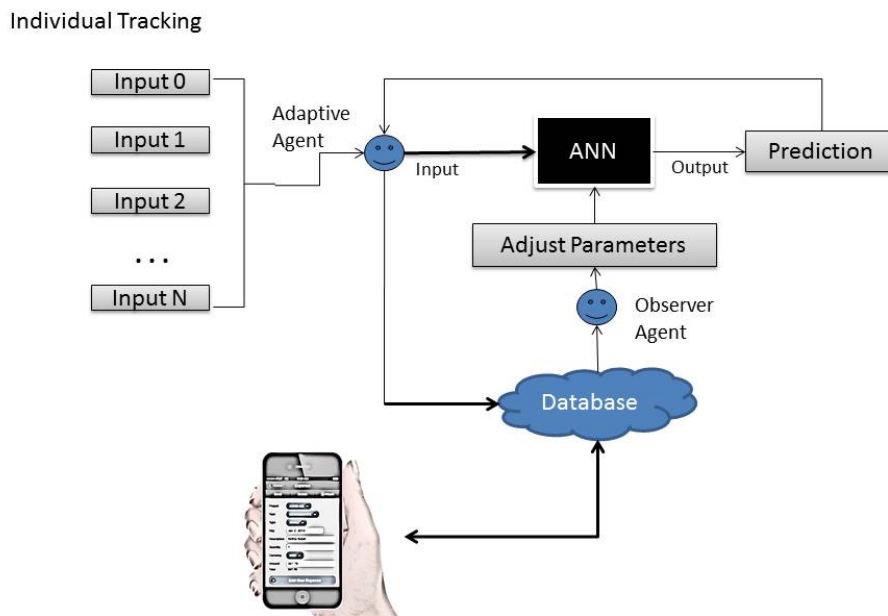


Figure 4.2: The model for individual tracking in QT systems.

By using a graphical interface, a human administrator can view the collected data from an Adaptive Agent and the performed prediction. Also he/she can indicate how much correct the performed prediction was.

The Adaptive Agents share their experiences through a cloud database, which represents the knowledge base of a particular type of “Quantified Things” experiments. In parallel, the Observer Agent observes the data provided by Adaptive Agents to verify whether predictions are satisfactory. If

they are not acceptable, the Observer Agent executes the adaptation process to adjust the parameters of the neural network used by Adaptive Agents. The adaptive technique used by the Observer Agent is a supervised learning algorithm, such as backpropagation (see Section 2.4), since we have historical data to train the network.

Table 4.1 summarizes how FIoT can be instantiated to meet the novel requirements listed above.

Table 4.1: Specific requirements for “Quantified Things” applications.

Req.	Layer	Description
R6	L3	A cloud database is maintained by the Agents. They insert into the database: the collected data from devices, predictions made by neural networks, and observed/actual results.
R7	L3	The application provides predictions about a problem domain using a neural network and a supervised learning method.
R8	L3	Since the agents maintain a cloud database, all collected and processed data can be presented to a human administrator through a graphic interface.

4.2.3

“Quantified Fruit”

To illustrate the use of our proposed model, we derived an example from “Quantified Things” applications: “Quantified Fruit.” We developed an illustrative example using bananas named “Quantified Bananas.” In this section, we provide the environment set up details as well as the experimental results to evaluate the efficacy of the proposed model.

Why “Quantified Fruit”?

The percentage of fresh fruit losses after fruit picking is extremely high. In the case of tropical fruit, such as bananas, more than a half of production is lost during transportation and distribution. The main problem is that the people allocated to distribute and sell fruit usually are not concerned about the existing techniques to preserve the fruit after it has been picked. In addition, tons of fruits are thrown away by consumers each year, as a result of not being consumed in time (Johnson et al., 2008).

There have been a number of investigations into what are the satisfactory conditions to prolong fruit shelf life (Boe and Salunkhe, 1967; Mpelkas and Kenyon, 1972; Johnson et al., 2008). Therefore, we can infer that fruit quality changes when products are stored under different environmental conditions (e.g. refrigerated, ambient temperatures), as shown in Figure 4.3. However, to combine these factors to develop recommendations for the best storage

conditions for each product is far from trivial. Consumers and retailers find it difficult to follow these recommendations. For example, to preserve onions at home, consumers are advised to keep them in a cool, dark and dry place. Bananas must be kept cool, but cannot be refrigerated. Some researchers affirm that bananas are best stored at room temperature (Johnson et al., 2008).



Figure 4.3: Bananas from the same bunch after being stored in different conditions.

We have prototyped a tool named “Quantified Fruit.” That is capable of monitoring fruit storage and making inferences about it. This tool indicates how many days it takes for fruit to spoil under specific environmental conditions. Depending on the chosen storage, fridge or room temperature, this tool will indicate a different number of days before spoilage. Consumers, distributors and retailers make use of this information to compare results and select the best place for storing a specific fruit.

Modeling “Quantified Fruit”

The “Quantified Fruit” system is depicted in Figure 4.4. The collection of “Quantified Fruit” devices that are composed of Arduinos (section 2.1) and sensors, represents the physical layer of our model. The sensors that are depicted in Figure 4.5 are used by the Arduino to collect environmental conditions, such as temperature, relative humidity, lighting and gases that may affect fruit ripening (e.g. hydrogen, methane).

According to the proposed model to develop “Quantified Things” systems, each device is managed by an Adaptive Agent. It uses a three-layered feed-forward neural network (Haykin, 1994) to make predictions (i.e. number of days to spoil). Then, predictions are also exhibited at a graphical interface. Thus, a human administrator (e.g. consumer) can monitor and evaluate the environment where the fruit is stored.

As shown in Figure 4.4, we set a parameter in the neural network indicating the type of fruit (e.g. “0” to indicate banana, “1” to indicate apple). So, the system can provide a prediction according to the selected fruit type. Besides informing the type of fruit, a user is also responsible for reporting

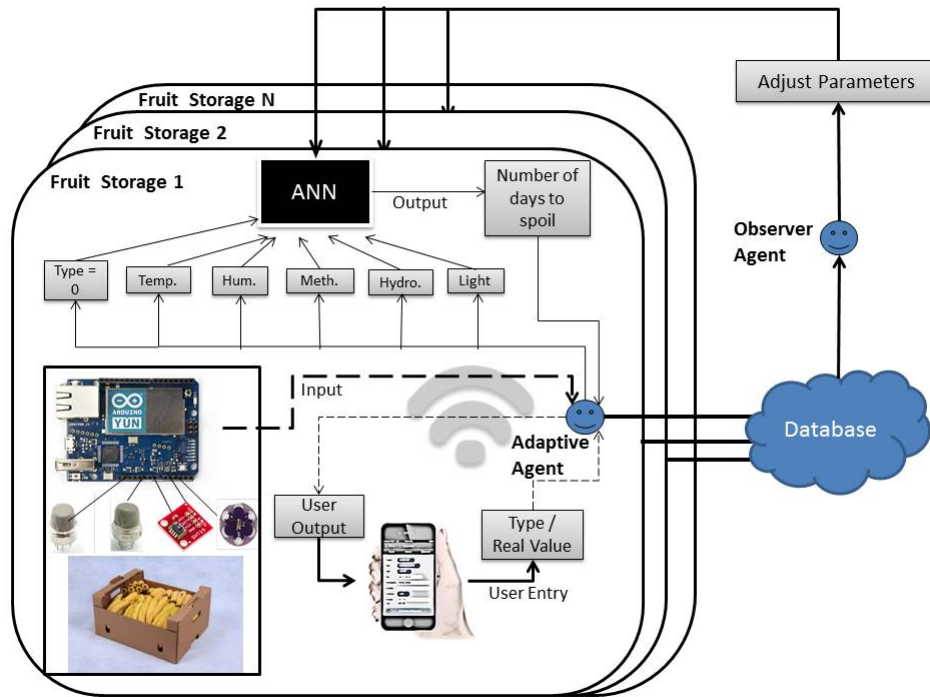


Figure 4.4: An instance of FIoT to create "Quantified Things" Instances.

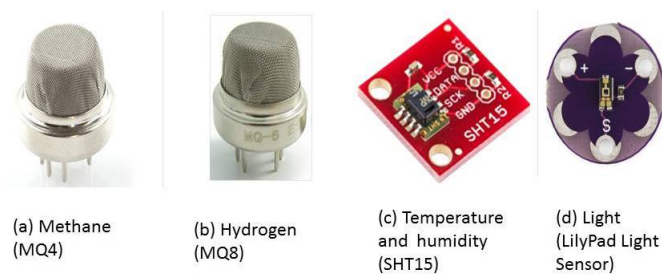


Figure 4.5: Set of sensors.

the actual amount of time the fruit took to spoil. In future releases, we may improve the system to automatically provide this information, since we may use the collected value of methane gas to assess whether the fruit is spoiled.

4.2.4 Quantified Bananas

Ten banana storage experiments were conducted to create an initial knowledge base for our application and to investigate how the system behaves.

Experimental Description

We carefully selected the individual bananas for haven a similar look. Each experiment was executed in a different condition. The experiments were created combining four parameters, as shown in Table 4.2: (i) dark (i.e. in a closed or open box); (ii) fridge (i.e. in the fridge or at room temperature); (iii)

rotten fruit (i.e. put together with rotten fruit or not); and ripe fruit (i.e. put together with ripe fruit or not).

Table 4.2: Experimental Description

<i>Experiment</i>	<i>Dark</i>	<i>Fridge</i>	<i>Rotten Fruit</i>	<i>Ripe Fruit</i>
1				
2				X
3			X	
4			X	X
5	X			
6	X			X
7	X		X	X
8	X		X	
9	X	X	X	
10		X	X	

For example, in the first experiment, we placed a banana in an open box (not dark), at room temperature, and by itself. The ninth experiment was executed in a dark place, in the fridge, and together a rotten fruit. Figure 4.6 illustrates conditions of experiments one, three, ten and five, respectively.

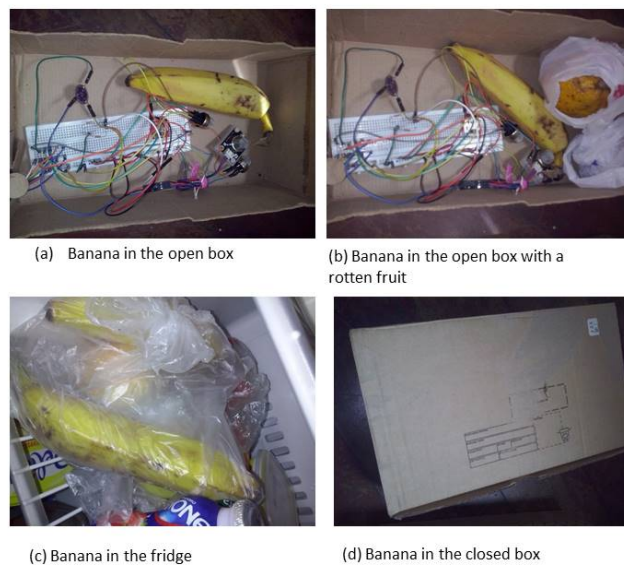


Figure 4.6: Example of scenarios.

Initial Database For Training Set

We used data collected by Adaptive Agents in these experiments to elaborate an initial database, as presented on the Table 4.3.

Table 4.3: Initial Database for neural network training.

<i>Exp.</i>	<i>Temp.(C)</i>	<i>RH.</i>	<i>Hyd.(V.)</i>	<i>Met.(V.)</i>	<i>Lum.(V.)</i>	<i>LifeSpam(Days)</i>
1	27.62	70.22	2	184.0	15.0	14
2	27.59	70.39	9	231.0	10.0	12
3	28.02	72.53	8	275.0	10.0	5
4	27.92	72.67	19.0	271.0	5.0	5
5	27.81	72.75	3.0	258.0	3.0	10
6	27.74	71.35	15.0	262.0	3.0	9
7	28.26	73.23	18.0	309.0	3.0	3
8	28.69	71.75	7.0	316.0	4.0	3
9	27.77	59.89	5.0	411.0	2.0	4
10	25.86	70.08	5.0	331.0	40.0	6

At the beginning of each experiment, the Adaptive Agents collected temperature (abbreviated Temp.), which is registered in centigrades (C), relative humidity (RH), hydrogen gas (Hyd.), methane gas (Met.), and luminosity (Lum.). The values of gas sensors were recorded according to the sensor output value (V.). At the end of each experiment, we made note of “actual” fruit shelf life (this information is not precise since naked-eye observations determined it).

Initial Training

The Observer Agent used these data to train the neural network and to provide first predictions. We established a stopping criterion to be used in backpropagation training based on minimum error (this error is calculated according to the standard deviations). Thus, the algorithm execution is repeated until the performance of the network is satisfactory.

Training Results

We verified the training process getting the predictions and comparing them with values registered as actual fruit shelf life for each experiment. This comparison is shown on Table 4.4, where the column “Expected Results” shows the “actual” shelf life, the column “Real Results” shows the predictions provided by the neural network, and the column “Error” the difference between these values, based on the normalized values.

Table 4.4: Results of backpropagation execution

<i>Experiment</i>	<i>Expected Results</i>	<i>Actual Results</i>	<i>Error</i>
1	1.0	0.999	≈ 0
2	0.857	0.866	-0.0095
3	0.35	0.340	0.01
4	0.357	0.382	-0.025
5	0.714	0.719	-0.005
6	0.642	0.614	0.028
7	0.214	0.207	0.006
8	0.214	0.225	-0.010
9	0.285	0.285	≈ 0
10	0.428	0.428	≈ 0

As showed on Table 4.4, differences between expected and actual results are not so far off. The largest errors were presented in experiments four and six, corresponding to approximately one day. Both tests were executed at room temperature and with ripe fruit inside the box. A possible solution to reduce this error is to provide new experiments with similar settings, since the backpropagation algorithm needs an extensive dataset to train neural networks.

Initial Evaluation

After training the neural network training, we executed two new experiments to evaluate the configuration chosen by the Observer Agent to set the neural network and verify how the system behaves in new situations. . The former was performed outside the fridge and in an open box. The system predicted thirteen days, and we observed that the banana spoiled in approximately twelve days. However, the previous experiments did not cover this new experiment's conditions: the banana was put in the fridge in a closed box, without other fruit. Although, the system suggested five days, we observed that the banana spoiled in approximately fifteen days.

First Adaptation

Given that the difference between results was so large, the Observer Agent had to execute backpropagation again to improve the neural network. This novel execution takes two new experiments into account, as showed on Table 4.5.

Table 4.5: New data to also be used in backpropagation.

<i>Exp.</i>	<i>Temp. (C)</i>	<i>RH.</i>	<i>Hyd. (V.)</i>	<i>Met. (V.)</i>	<i>Lum. (V.)</i>	<i>Expec.</i>	<i>Real</i>
11	28.21	70.24	3	183.0	16.0	12	13
12	27.77	59.89	5	283.0	30.0	15	5

Discussion

In order to analyze our device, our first experiments used bananas. Since we provided it with the capacity of self-adaptation, we expect that the device system will adapt itself to support new types of fruit. Then, the idea is to use other types of fruit during the training experiments or at run-time (by users) to check whether the system automatically improves itself.

We need to execute many more experiments using bananas and other fruits to optimize the “Quantified Fruit” device and to improve predictions. On the top of offering predictions about shelf life, this system can be adapted to provide other kinds of predictions, such as the percentage of fruit production that could be lost under specific transportation conditions. Besides making predictions, this device could also be extended to make suggestions and act on its own. For example, the “Quantified Fruit” device could make suggestions for temperature changes in real time. If we added a cooler to the device, we understand that it could perform temperature changes by itself.

We expect this device will lead to improvements both in transportation methods by distributors, consumers and retailers, as well as their storage patterns and practices (in the fridge, at ambient temperature, loose, in original packaging, etc.).

As the framework allows the system designer to use different types of controllers and algorithms, he/she could also use another structure to provide temporal and more precise predictions. Actually, an Adaptive Agent makes a prediction only based on the data provided during the training step. It has not been providing a continuous prediction, which takes the daily environment changes into account. As a solution, a framework user can add a new input parameter in this neural network to indicate the time of collection.

We conducted an initial evaluation of our scenario and of the capacity in generalization of our neural network based prediction system. In the future, we may conduct a more detailed evaluation (e.g., extended training sets, cross validation (Setiono, 2001)).

4.3 Smart City

Smart Cities is currently the hottest trend associated with the Internet of Things (Bohli et al., 2013; Mitchell et al., 2013; Gubbia et al., 2013). In order to have a Smart City it is fundamental to have many sensors scattered throughout the whole city collecting information, such as water and energy consumption, traffic and garbage monitoring. Based on that information, Smart Cities should have smart traffic light, smart lighting, smart waste management, and smart environment monitoring (Mitchell et al., 2013). Therefore, the infrastructure of a Smart City consists of self-managed entities, not relying only on a human centralized control (Mitchell et al., 2013).

In order to support smart services, IoT principles are applied to create self-managing car traffic control applications (Möller, 2014; TheGuardian, 2015), aiming at rebuilding the actual structure of traffic lights. For instance, cars, traffic lights and pedestrians will all be connected via the Internet, collecting and sharing data, such as GPS data from cars, traffic lights intervals, and camera images (TheGuardian, 2015). Based on this data, traffic lights will turn green or red, GPS consoles will offer drivers different routes, etc. Figure 4.7 illustrates a futuristic vision of cars and traffic lights interaction (Möller, 2014).



Figure 4.7: Introducing the IoT paradigm for transportation system analysis (Möller, 2014). Pg. 104.

The reduction of urban traffic congestion continues to be the main goal of this new smart approach of car traffic management. For example, (Carlino et al., 2012) propose that optimized traffic policies should be determined by the use of autonomous cars. However, given that their research focus on a intersection control mechanism, they only analyze how different policies affect a small portion of the road network.

According to Standford-Clark, an IBM engineer, the problem is not to change the traffic lights, but the “interconnection of unintended consequences.” Thus, most traffic lights sequences are set via longer term algorithms, taking the whole of the road network into account (TheGuardian, 2015). Unfortunately, determining such sequences is a non-trivial and time consuming task,

as one must take into account a wide range of factors like, traffic density, pedestrian flows, and road complexity.

FIoT makes it possible to create dynamic controllers for homogeneous things situated in a distributed environment by using a self-developing decentralized and adaptive process. Figure 4.8 shows the model used by FIoT for generating control applications for distributed environments.

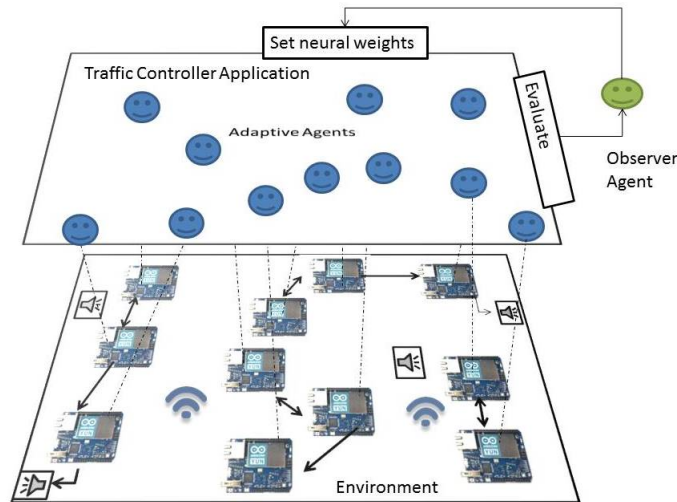


Figure 4.8: A FIoT instance for creating a smart controller for homogeneous things.

The model comprises two layers: the environment, in which the homogeneous things are situated, and the controller application layer. The devices in the environment layer indirectly interact with other devices within a range. Our aim is to provide self-organizing systems consisting of several independent components communicating and taking decisions on the basis of local level data. Each device is controlled by an Adaptive Agent that collects data and establishes how its device interacts with the environment. If the things in the environment are GPS equipped, the interaction may occur on the application level via Adaptive Agents. The Observer Agent evaluates the overall system performance. For example, maximizing the traffic throughput, is the global goal in a car traffic experiment.

4.3.1 Car Traffic Application

In this subsection, we describe a simulated car traffic scenario that stands as our application physical layer. Figure 4.9 depicts the elements that are presented in our scenario: vehicles, traffic lights, road segments, dividers and intersections. All roads are one-way; a segment is a portion of a road; intersections connect two or more segments; and a road divider divides a

segment into two segments. We modeled our scenario as a graph, as shown in Figure 4.10. There edges represent segments and nodes represent road dividers and intersections.

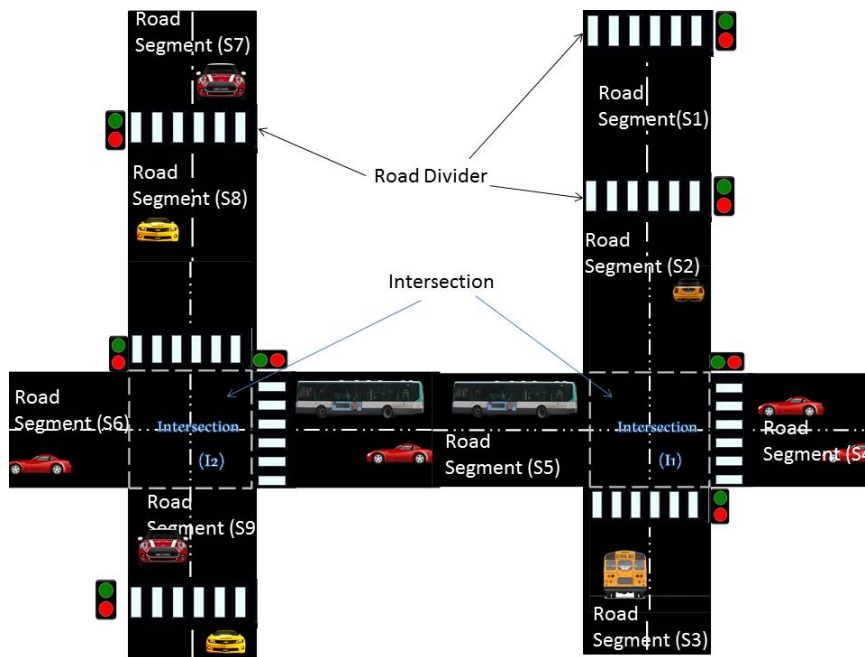


Figure 4.9: Traffic elements.

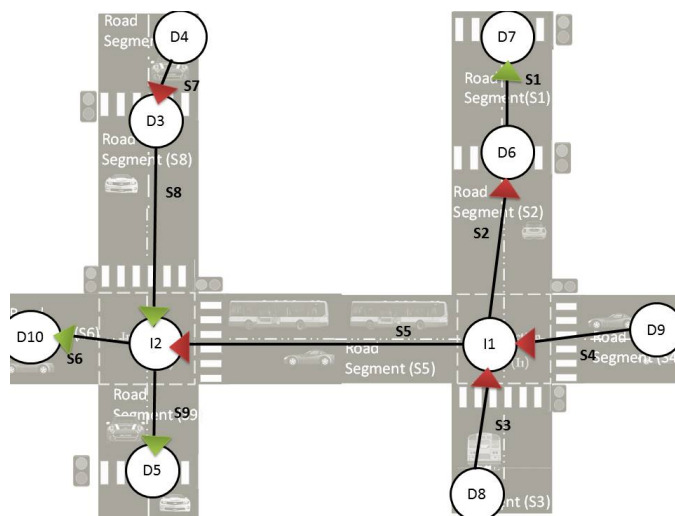


Figure 4.10: Modeling a scenario as a graph.

Each segment has a traffic light and supports a maximum number of vehicles. A road intersection element, which connects two or more segments, does not let more than one segment to set its traffic light to green (in order to avoid vehicle collisions).

At the beginning of the simulation, each vehicle randomly chooses a departure and a target node. According to these selections, the system calculates an optimum and fixed route for each car based only on the number

of nodes. After that a vehicle completes its journey, it must start a new one, as the system calculates a new route. The simulation ends after the execution of a specific number of cycles. Each cycle lasts for the same fixed duration. The elements in the simulation do the following for each cycle:

1. All vehicles try to change from the current road segment to the following one. A vehicle will stay in the same segment if the following segment reached the maximum supported number of cars, or if the traffic light color of its current segment is red;
2. Vehicles that already have concluded their journeys are reintroduced into the map at the start of a new journey;
3. Each road segment decides whether to change the current color of its traffic light or not. If it is in an intersection and the decision is to set the light color to green, it must send a requirement value to the respective road intersection element.
4. Each road intersection receives the requirement values from its segments. Then, it lets the segment with the highest requirement value to set its light color to green.

As described above, only road segments are able to make decisions. The others, vehicles and intersections, have to execute previously established tasks.

Smart Road Segment

Each road segment has a simulated microcontroller board associated with it that has an apparatus for calculating the rate of vehicles, interacting with the closest segment, and sending a request value to its respective node element in order to set its own traffic light color to green.

Thus, the GodAgent creates an Adaptive Agent for each road segment in the scenario. Independently of the application, an Adaptive Agent always has to execute three tasks: data collection, decision making and action enforcement. For this experiment, the first task consists of receiving data collected by the respective road segment's microcontroller. It provides data related to vehicle flow, information from its neighbor segment and its current traffic light color.

To make decisions, Adaptive Agents use a "three-layer feedforward" with a feedback loop (see Section 2.4). Feedback occurs because the output of its traffic light color influences its next network input, as shown in Figure 4.11. By using a recurrent network, we aim at providing a kind of memory for these agents. Thus, our goal is to enable them to consider the duration span of

a traffic light in a specific color. Therefore, the input layer consists of three neurons.

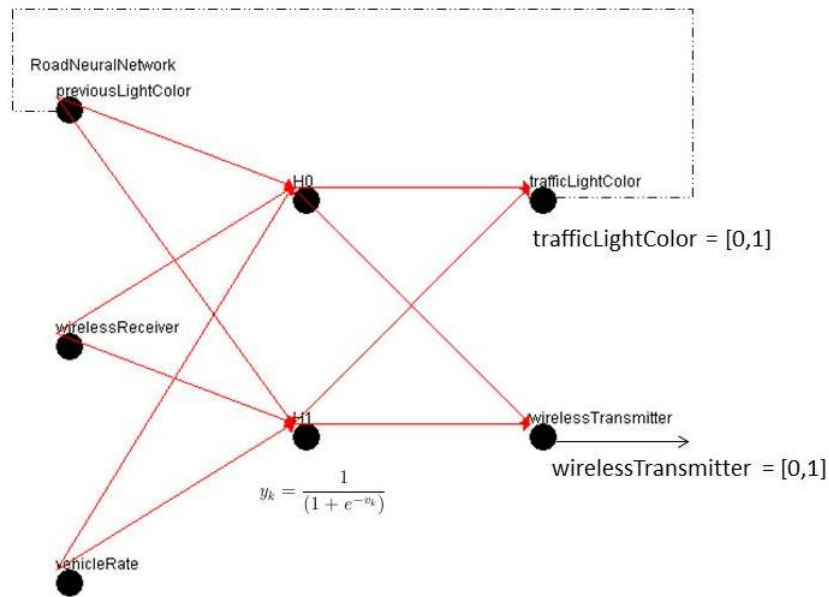


Figure 4.11: Adaptive Agent's Neural Controller.

The effector task of an Adaptive Agent consists in sending the network outputs to the microcontroller board. The outputs are a value associated with the traffic light color and a value to be send to its neighbor. Although the light color is only red or green, the color output can be a value in the closed interval $[0,1]$. If the color output value is less than 0.6, the color light will be red. Else, the segment has to send this value to the respective node as a request.

The middle layer of the neural network has two neurons to connect the input and output layers. These neurons provide associations among sensors and actuators (see Section 2.4). These associations represent the system policies changing according to the encoded neural network configuration.

At the beginning of the simulation, the messages exchanged between neighbors have no meaning. Moreover, a road segment agent is dumb and does not know how to use the collected data to make a decision on the setting of the traffic light color. Therefore, we configured the Observer Agent to perform an adaptive process. It will adjust the parameters of the neural network in order to find a solution for relating inputs (message received, vehicleRate, and currentColor) and outputs (message to send and traffic light color).

ObserverAgent: Adaptive process

Evolutionary algorithms have been applied to provide the design of system features automatically (see Section 2.5.1). By using a genetic algorithm,

we expect that a light policy, sporting a simple communication system among road segments, will emerge from this experiment. Therefore, no system feature was specified at design-time (e.g. a communication system, the effect of vehicle rate on road segment decision).

The evaluation and adaptation process performed by the Observer Agents is depicted in Figure 4.12.

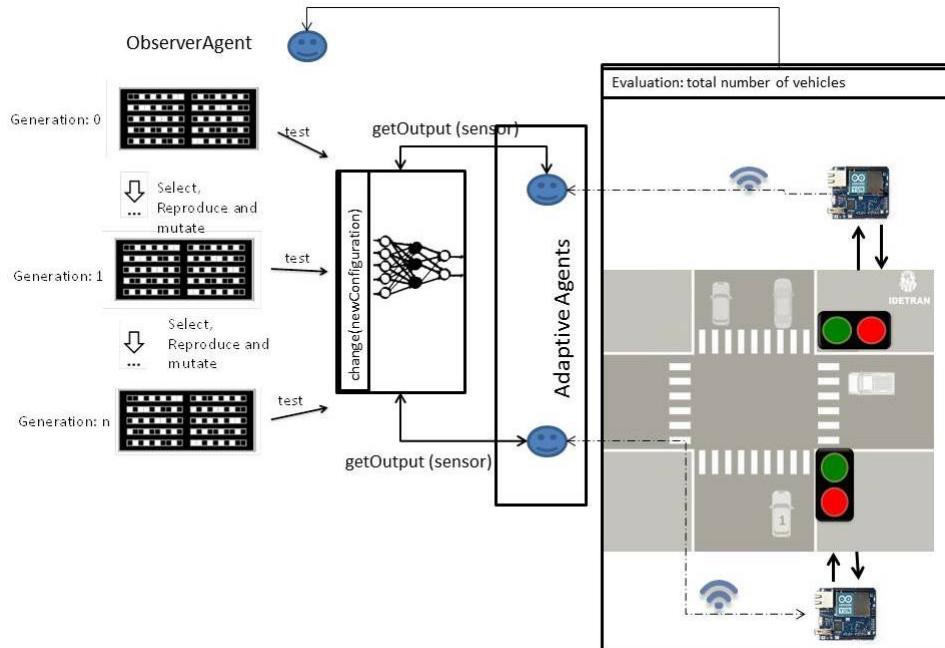


Figure 4.12: Performing an adaptive process to adjust the Traffic Neural Controller weights. Figure adapted from (Nolfi and Gigliotta, 2010). P.7.

The weights of the neural network used by the Adaptive Agents vary during the adaptive process. The ObserverAgent applies a genetic algorithm to find a better solution. It contains a pool of candidates to represent the network parameters. The ObserverAgent evaluates each one of them according to the number of cars that concluded their routes after the end of the simulation.

4.3.2 First Experiment

The first simulation scenario is depicted in Figure 4.13. We have created the urban road network scenario based on a small section of a real city, Feira de Santana, Bahia, Brazil. This chart is composed of 31 nodes and 48 segments. Each segment links two nodes having only one-way direction. For simulation purpose, we established 15 nodes as departures (yellow points) and two as targets (red points). Each segment has a traffic light. In the graph, the green and red triangles represent the traffic light colors.

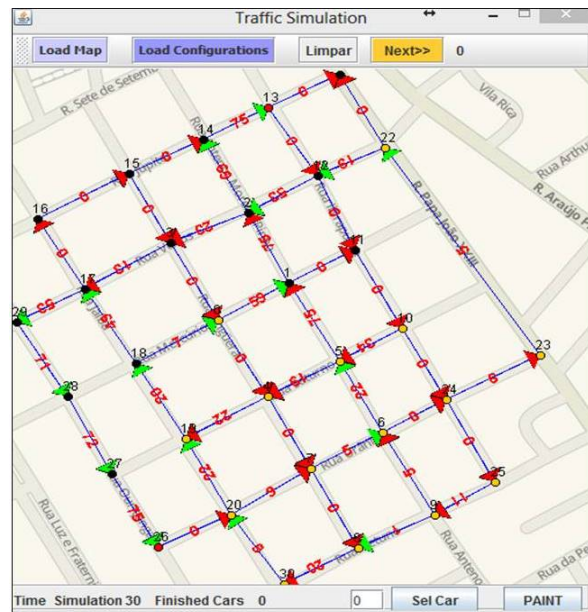


Figure 4.13: Simulation Urban Road Network. Adapted from Waze (2014) (MOBILE, 2014).

We started with 1000 vehicles for this experiment. The capacity of each road segment in this experiment is 75 vehicles. As we described before, the only role of the vehicles is to try to end their routes.

Evolutionary Algorithm: Simulation Parameters

Given that we are proposing a simple experiment, the evolutionary process lasts only 20 generations (i.e. the process of testing, selecting and reproducing candidates is iterated 20 times). During the test stage, each team of 48 Adaptive Agents (i.e. the number of road segments in the scenario) is allowed to “live” for 30 cycles by using a candidate, as shown in Figure 4.12. As each car departure and target are randomly selected and can affect the test result, more than one test is performed for each candidate.

The fitness of each candidate consists of the number of vehicles that concluded their route after the simulation ended.

The individuals with the highest fitness are selected to generate the new generation by using crossover and mutation (see Section 2.3). Figure 4.14 illustrates the evolutionary configuration file that was used to set the algorithm in our experiment.

Simulation Results

After executing the evolutionary algorithm, Adaptive Agents evolve the ability to find a satisfactory logical of traffic light decision in order to improve urban traffic flow.

```

1 <numberOfGeneration>:20
2 <numberOfTests>:3
3 <numOfBestToBeSelected>:20
4 <numOfChildren>:4
5 <numberOfPopulation>:100
6 <elitismSaveFitness>:1
7 <maxNumGenes>: 10
8 <valueMaxOfGene>: 5
9 <rateMutation>:10
10 <fitness>:1

```

Figure 4.14: The configuration file for the evolutionary algorithm.

We aim at evaluating the best solution during the training period. Figure 4.15 illustrates the best individual of each generation (i.e. the candidate with the highest fitness value). Normally, the individual with the highest fitness presents performance better than the others. Accordingly, we selected two individuals in the graph to investigate their solutions (i.e., how the Adaptive Agents act by using their neural configurations): (i) the best of the second generation (point A), and (ii) the best of all generations (point B).

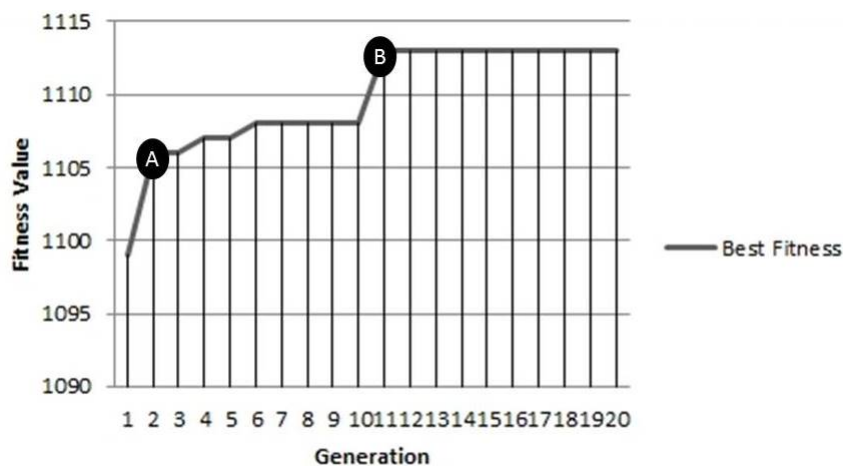


Figure 4.15: Simulation Results - Best Fitness.

The best candidate of the second generation (point A) provided the following solution: road segment always try to set its traffic light to green. Thus, if the road segment is not situated in a intersection, its traffic light remains green during all the simulation. The segments comprising an intersection make decisions based only on the vehicles rate and the current traffic light color. We conclude that they do not take messages into consideration for making decisions.

By configuring the neural network with the best candidate representation (point B), each road segment displays the following solution:

- if it is not in a intersection, set its traffic light color to green.

- if its current traffic light color is green, it increases the light color output (so the road intersection will prioritize this segment)
- if its traffic light color is red and the vehicle rate is close to its maximum, send message “1.0” to its neighbor
- if it receives the message “1.0”, and the vehicle rate is not close to its maximum, then it decreases the light color output (losing priority in the road intersection’s decision)

Road Segments in a intersection seem to use a simple communication system to ‘negotiate’ the light color scheduling. First they “analyze” the current color of their traffic light. The priority is given for the agent that already has the traffic light set to green, except when the traffic light of the other agent is red and its vehicle rate is considerably higher.

The candidates performed many other solutions along the evolutionary process. An example of strategy is to block agents that are situated in a full road segment (i.e. a segment showing the maximum number of cars). If the vehicle rate in a segment reached its maximum and its actual traffic light color is red, the traffic light color remains red. Some of these behaviors are lost during evolution, since only those that showed a higher number of cars concluding their routes remained during evolution.

Evaluation of the Best Candidate

We selected the best candidate from the evolutionary process to provide comparisons between our approach and “conventional” traffic light policies. We have been considering as conventional the normal way to control traffic lights, the so called fixed-time control. This type of control fixes the sequence of phases (red or green) and their durations (Prothmann et al., 2011). We simulated two fixed-time approaches. The former changes all traffic lights colors in every cycle. The latter changes all the traffic lights colors at the intersections every two cycles, and sets the others green for 5 cycles and then red for only one cycle.

We executed the simulation three times, using the best solution presented above and each one of the two “conventional” solutions. Figure 4.16 presents the number of vehicles that concluded their routes.

To estimate how generic is our approach, we will now apply this best solution to a new scenario.

The new simulation scenario is depicted in Figure 4.17. We selected the Copacabana neighborhood, one of the most congested areas of the city of Rio de Janeiro, that has a large number of scattered traffic lights. The simulated

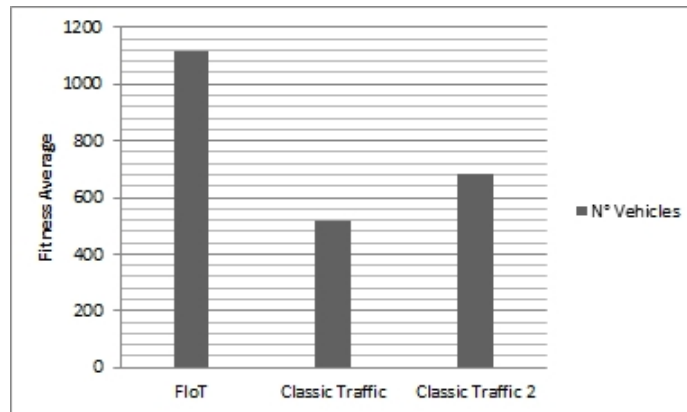


Figure 4.16: Comparison of the FIoT approach and conventional systems in the first scenario.

scenario takes the actual directions of the roads into account, but the amount of traffic lights does not correspond to reality.



Figure 4.17: The second scenario - Copacabana - RJ -BR. The car 9's route. Adapted from Waze (2015) (MOBILE, 2014).

We initialized 1000 vehicles for this experiment and increased the capacity of road segments to a 100. As the scenario got larger, we increased the simulation time to 40 cycles (40 minutes).

As shown in Figure 4.17, we illustrated the smallest route calculated for car 9, situated in road 2 (as pointed in the graph) aiming at reaching road 30 (the red arrow). Using our approach, this car spent 17 cycles to reach its target. By testing the first fixed-time control (classic 1 approach), the same car spent 34 cycles to conclude its route. Figure 4.18 illustrates a comparison between our solution and the fixed-time approaches in this new scenario.

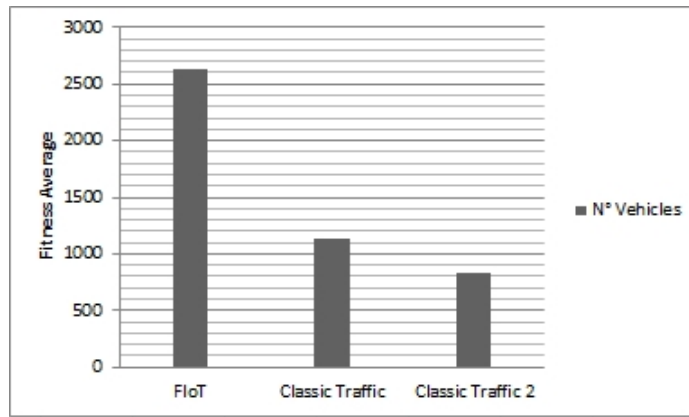


Figure 4.18: Comparison of the FIoT approach and conventional systems in the second scenario.

Design-Time Adaptation

IoT-connected traffic elements have other goals besides cutting traffic delays. One of these goals is to boost public transport. So, we assume that traffic engineers have decided to prioritize bus flow. Thus, the overall objective of this new experiment is to maximize the number of buses that conclude their route in a specific period of time.

The road segments are able to classify vehicles such as cars and buses (a new sensor to provide the bus flow rate). Thus, we create a new controller to be used by Adaptive Agents, as shown in Figure 4.19.

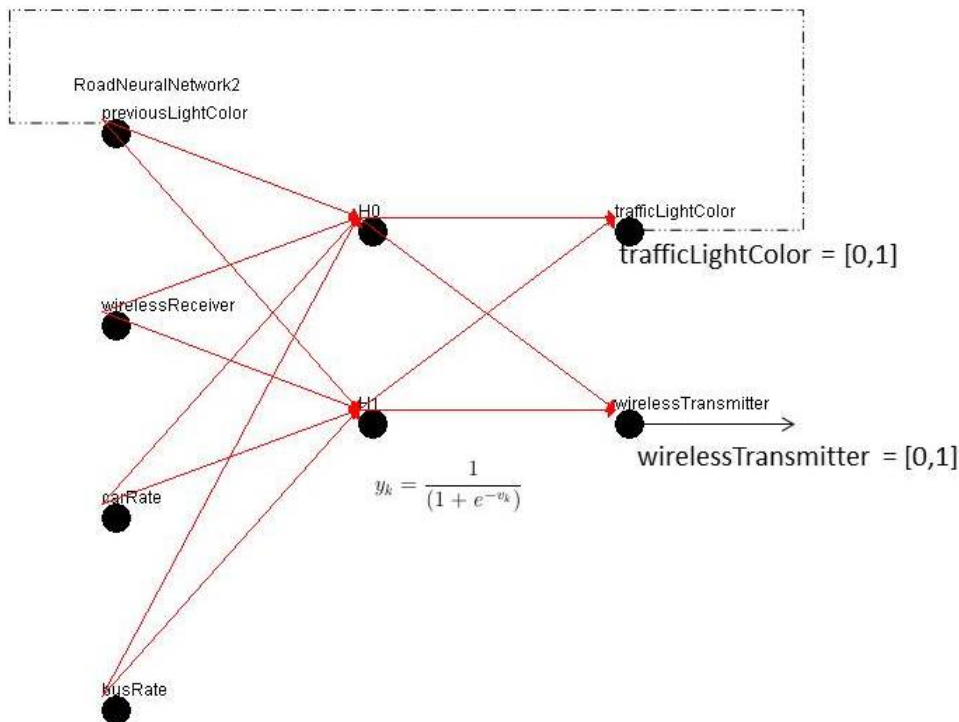


Figure 4.19: Agent's Neural Controller.

The Observer Agent has to execute the evolutionary process again. To emphasize solutions found during the evolutionary process that gave priority to the bus flow, the Observer Agent calculates the fitness of each team of Adaptive Agents according to the equation 4-1. It consists of the sum of the number of cars and (100X) the number of buses that concluded their route after the end of the simulation.

$$fitness_i = numCars + (100 * numBuses) \quad (4-1)$$

From the 1000 vehicles initialized in previous experiments, we now initialized 800 cars and 200 buses. After running the evolutionary process, the Observer Agent found a new solution to solve this new problem.

Now, in addition to the classical approaches presented above, we used a new approach to compare our solution. Some researcher proposals (Evans and Skiles, 1970; Al-Sahili and Taylor, 1996; Jacobson and Sheffi, 1981) aim at improving public transit through bus preemption of traffic lights. This approach is the same that is commonly used by ambulances. Normally, an ambulance's driver has a device to set traffic lights to green. To simulate it in our scenarios, we enable all road segments to set their traffic light colors to green when detecting one or more buses.

Figures 4.20 and 4.21 present the results of the simulation in both scenarios, comparing the number of vehicles (cars and buses) that had concluded their routes resulting from the “conventional” solutions and the evolved agent approach.

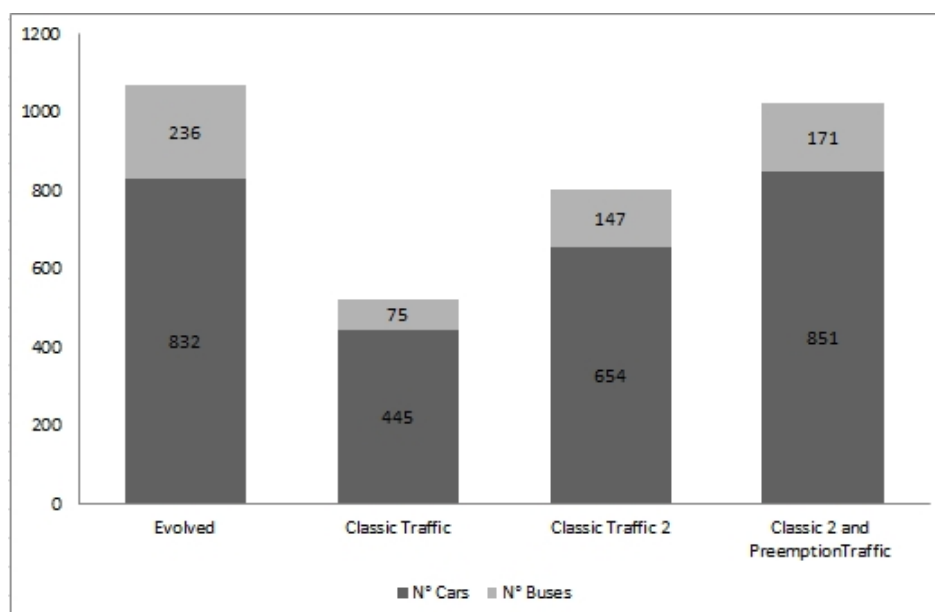


Figure 4.20: Comparison of the evolved agents approach and conventional systems in the first scenario.

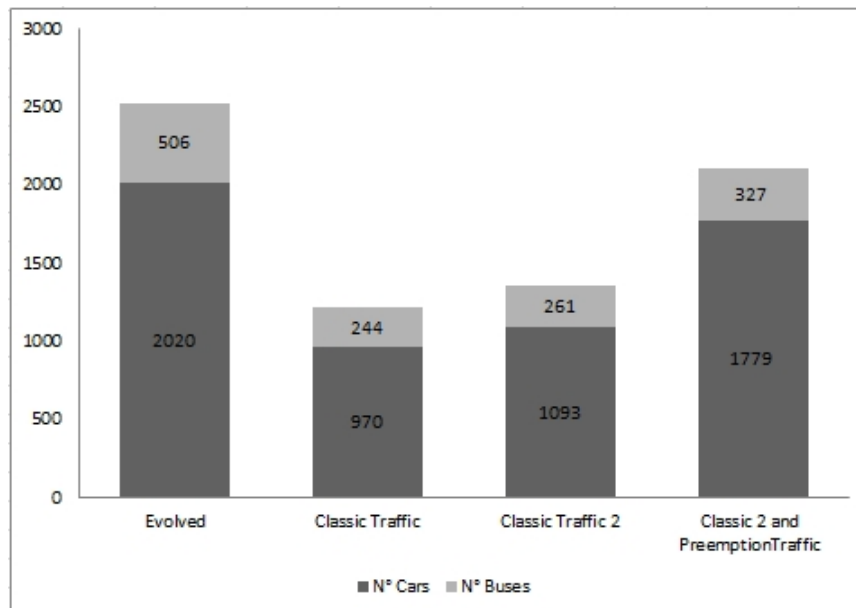


Figure 4.21: Comparison of the evolved agents approach and conventional systems in the second scenario.

By using the evolved agents approach, the number of buses that concluded their routes is higher than using other approaches.

Discussion

Although we considered only a few factors to make decisions about setting the traffic light colors in the experimental examples that we presented here: the vehicle flow, the previous traffic light color, and a neighborhood interaction mechanism, even for a traffic expert, to set a satisfactory strategy based only on these factors would be a complicated task. If we had taken all of the other conditions into account, such as the number of pedestrians and ambulances, messages from cars and road problems, the problem would be impossible to be solved by a human.

Our experiment showed that FIoT allows the creation of a controller to determine traffic light policies automatically. We described the results of this experiments in which a simple communication system arises among a collection of initially non-communicating things, evolved in order to decrease congestion.

From the results, we also showed that road segments that evolved for maximizing traffic flow in a specific scenario can be ported as it is to a new environment and still obtain satisfactory results. In addition, we presented comparisons between our approach and “conventional” traffic control systems in two scenarios. In both scenarios, the number of vehicles that had concluded their routes based on the FIoT approach was twice as high.

4.4

A Future Instance: Quantified US

In this section, we show another FIoT's instance to be prototyped in the future. We briefly describe it and show in section 4.5 how it can be generated by filling the variable points of the framework.

Firstly, we design this application as an individual self-tracking tool. Then, we show that by changing the variable points of FIoT, we can extend this application to model a Quantified Us instance.

The next subsection describes the device that we intend to use in such instances.

4.4.1

Physical Layer: Modeling the device

The popularization of microelectronics has boosted the creation of many specialized devices, such as Fitbit pedometers, Apple Watch (Apple, 2014), myZeo sleep, Oura ring, and Nike + and Jawbone UP fitness trackers (Swan, 2013; Labs, 2014). Most devices use air quality sensor, heart rate sensor, biomedical electrodes, blood pressure measure, GPS and accelerometer technologies (Apple, 2014; Labs, 2014).

As a result, a lot of applications that collect, report, and respond to information from the user's own body have already been developed. People measure and understand more about their sleep quality, health, exercise performance, and many all-day activities. Consequently, people have been shifting from passive to active participants in diagnosis and preventive care (Hirsch, 2015).

However, none of the known gadgets take flatulences and other gases daily emitted by humans into account. Most people pass gas 13 to 21 times a day (Diabetes et al., 2015), and these emissions tell a lot about a person's health (Pimentel et al., 2012).

Moreover, a person emits various gases from different parts of the body. These gases may be related to different diagnostics (Mathew et al., 2015; Pimentel et al., 2012; Tangerman, 2009; Franciosa, 1977; Hirakoba et al., 1992). We list a few in Figure 4.22.

Passing gas through the mouth is called belching, burping, or eructation. Passing gas through the anus is called flatulence (Diabetes et al., 2015) that usually contains nitrogen, hydrogen, carbon dioxide, oxygen, and methane (Tangerman, 2009; Suarez et al., 1998).

As a future work, this device will consist of an Arduino couple with gas sensors for the detection of hydrogen, methane and carbon dioxide.

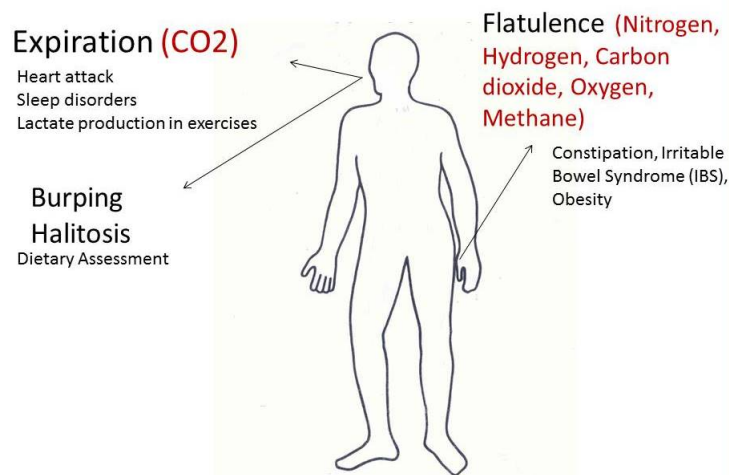


Figure 4.22: Relation between gases emitted and diagnoses.

4.4.2

Quantified Self Approach

Our aim is to propose the design of a simple self-tracking application to allow a user to collect and measure her gases. Based on the collected data, we believe that a person will be able to analyze the following characteristics:

- The amount of gas may be related to the certain types of food. “Foods that produce gas in one person may not cause gas in another” (Diabetes et al., 2015). By using this information, a person will investigate which food types cause such problem. Accordingly, she will take the appropriate decisions to reduce the amount of gas, such as to avoid specific aliments, especially milk products given that she has lactose intolerance (Diabetes et al., 2015).
- The amount of methane and hydrogen in her farts may be related to some diseases: According to the authors in (Pimentel et al., 2012), “the prevalence of methane over hydrogen in human farts may correlate with obesity, constipation and irritable bowel syndrome.” If her flatulence has these characteristics, she should look for a health care provider.

Based on the research presented in (Pimentel et al., 2012), this application will limit itself to collecting the amount of gases and, if it is the case, suggesting a diagnosis of irritable bowel syndrome.

Given that each person has its own numbers and does not share them with the others, the system tracks only one individual, requiring only one Adaptive Agent. This agent collects data and provides an advice based on a simple “if-statement” control, as depicted in Figure 4.23.

Given the simplicity of the application, the Observer Agent does nothing.

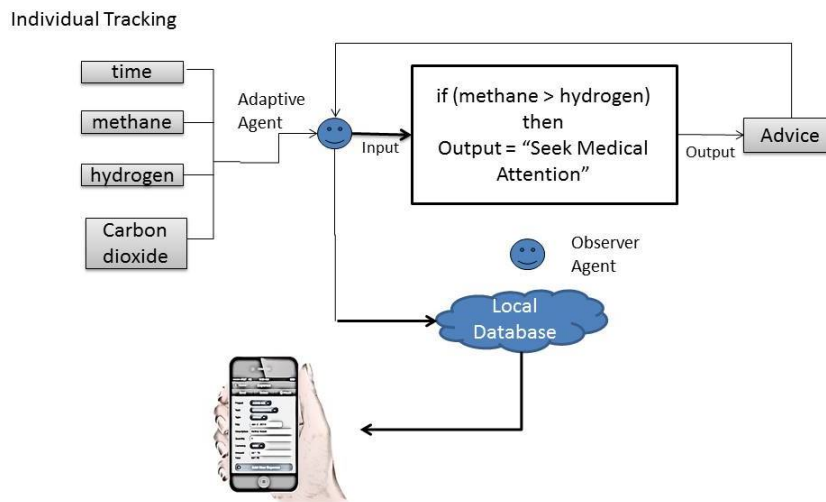


Figure 4.23: Future Instance: QS model.

4.4.3 Quantified Us Approach

According to the aforementioned references the relation between the gases that we emit and diseases should be further investigated. Thus, we do not have enough data to program an all encompassing "if-statement" control for covering more diseases.

The idea behind the Quantified Us Approach is that this application can be improved by providing it with knowledge sharing and collective-level classification features. Figure 4.24 illustrates the model that we are suggesting for this "Quantified Us" experiment (similar to the "Quantified Things" model).

In this case, the user will directly set two neural inputs: the type of problem to be diagnosed (e.g., lactose intolerance) and the information about the last meal (e.g., milk, bean, soft drink). As an output, the system will show the user's chance of having the selected problem type.

4.5 How the generated applications adhere to FIoT

This section presents how the applications listed above adhere to the proposed framework by filling the main variable parts: controller creation by the Framework user, leaving to the Observer Agent the system evaluation and the controller adaptation.

Table 4.6 shows how "Quantified Things" applications adhere to the Framework, extending FIoT's flexible points. A dataset provided by a group of agents fills the "making evaluation" hotspot developed for this applications. The Observer Agent evaluates a dataset containing input from Adaptive

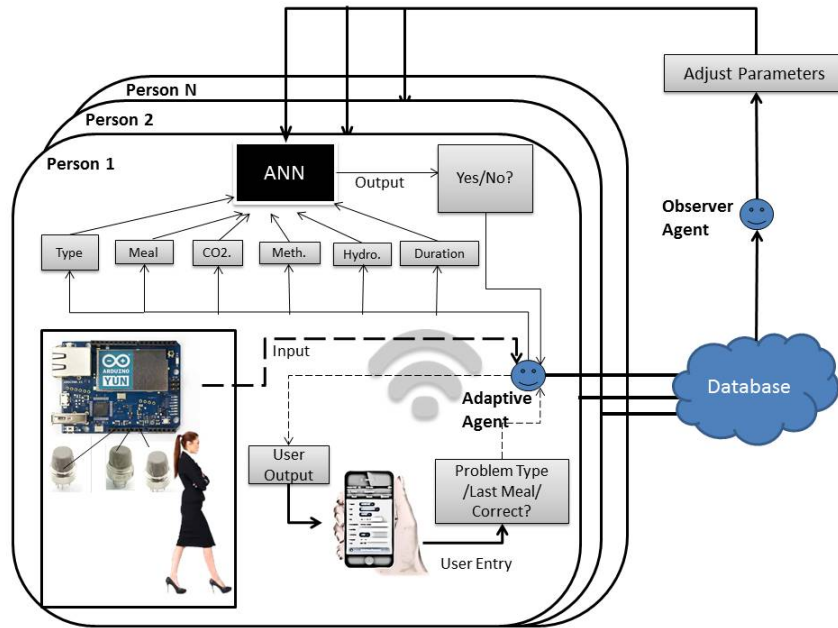


Figure 4.24: Future Instance: QU model.

Agents and neural predictions. Based on this historical data, the Observer Agent calculates the error standard deviation of a set of data. Then, it evaluates whether it is necessary to generate a collective adaptation.

Table 4.6: Instance I: Flexible Points

<i>Framework</i>	<i>Application</i>
Controller	Three Layer Neural Network
Making Evaluation	Evaluation of a Group of Adaptive Agents: for a evaluated group, the Observer Agent concludes if all Adaptive Agents need to adapt or not
Controller Adaptation	Supervised Learning (Backpropagation)

The Table 4.7 shows how the “Car Traffic Control” application adhere to the proposed framework, extending the FIoT flexible points.

Figure 4.25 illustrates a comparison between these approaches. It also depicts a briefly technical description of how the applications’ controllers implemented the main variable methods. As we explained in section 3.5, all controllers have to implement the methods `getOutput()` and `change()`. The first method provides an output after receiving an input. Adaptive Agents use this method on decision activity. The second method is accessed by the

Table 4.7: Instance II: Flexible Points

<i>Framework</i>	<i>Application</i>
Controller	Three Layer Neural Network
Making Evaluation	Collective Fitness Evaluation: Test a pool of candidates to represent the network parameters. For each candidate, it evaluates the collection of Adaptive Agents, comparing fitness among candidates
Controller Adaptation	Evolutionary Algorithm: Generate a pool of candidates to represent the network parameters

ObserverAgent during the adaptation process to change the configuration of the controller used by Adaptive Agents. For example, if a framework user chooses to create an application by using neural network and genetic algorithm, while Adaptive Agents will use the neural network controller to calculate an output, the Observer Agent will execute the genetic algorithm to adapt the neural network by changing its weights.

PUC-Rio - Certificação Digital Nº 1322090/CA





Instance	Evaluation	Adaptation (controllerChange(): newConfiguration)		Controller Creation (getOutput (input): output)	
	Predicted value	Back-propagation (Supervised Learning)		Three-Layer Feedforward Neural Network	
	 Observed value (number of days to spoil)	(oldConfiguration) Neural weights	(New) New neural weights	(Input) Gas, temperature, humidity sensors	(Output) Predicted value
	Fitness Function (number of cars)	Genetic Algorithm (Evolution)		Three-Layer Neural Network with feedback	
		(oldConfiguration) Neural weights	(New) New neural weights	(Input) Car rate, wireless receiver, Light color output	(Output) Light color and Wireless transmitter 

Figure 4.25: How the generated applications fill the main variable parts of FIoT.

We also provide a picture to illustrate how we imagine that the Quantified Self application and its extension can be instantiated by using FIoT.

Figure 4.26 depicts how these future applications will be generated by filling the variable points of the framework.

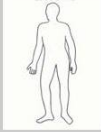

Instance	Evaluation	Adaptation (controllerChange(): newConfiguration)		Controller Creation (getOutput (input): output)	
	\emptyset	\emptyset		"if-else statement"	
		\emptyset	\emptyset	(Input) Methane,hydrogen, CO2, time	(Output) "Message"
	Classification Error	Back-propagation (Supervised Learning)		Multi-Layer Feedforward Neural Network	
		(oldConfiguration) Neural weights	(New) New neural weights	(Input) Type, duration, meal, methane, hydrogen, CO2	(Output) Yes/No

Figure 4.26: A comparison between Quantified Self and Quantified Us instances' design.

4.5.1 Discussion

If we decided to create a traffic application by using a neural network and back-propagation algorithm, for example, we would have to provide an input-output mapping dataset based on several classification of traffic regime policies. Quantified Things experiments require continuous adaptation at run-time. Therefore, genetic algorithm could not be a good solution, since this algorithm execution is slow and is normally applied to offline adaptations. Thus, it is important to investigate and select techniques that can be designed to fit a particular application.

In this chapter, we evaluated our proposed approach in two types of applications: (i) Quantified Bananas, using a predictive system, and (ii) Car Traffic Application, using a distributed control. For both applications, we tested various scenarios and showed how our approach of self-adaptation can provide pertinent solutions.

In addition, we presented a future application and described how it can be instantiated by using FIoT.

5

Conclusion

The area of Agent Oriented Software Engineering (AOSE) has so far addressed only small scale and even toy applications. We showed in this dissertation that a self-organizing and adaptive multi-agent software framework can be designed and implemented to derive now-a-days complex applications while doing so in an effective way. Software frameworks are domain oriented and we have chosen the Internet of Things (IoT) as such domain.

IoT applications are increasingly complex and require scalability beyond billions of devices and the ability to cope with environments that are in continuous transition. IoT is an emergent technology which has the potential for significant impact (Stamford, 2014). Self-organizing and self-adapting IoT multi-agent applications are an important evidence to show that agent oriented IoT assistants are an original contribution to both AOSE and IoT. The breath and relevance of MAS associated with Software Engineering to solve real world problems is the evidence we used to support our above claims.

We know of few research results in the literature about agent-based architectures for Internet of Things (Fortino et al., 2013; Lopez and Pérez, 2012). None of them presents the design of a complex case study (i.e. using a vast number of cooperative things). Thus these works do not show efficiency in wide scenarios, where things must cope with a changing environment and where a sophisticated organization system is required. Therefore, important features mentioned on our work regarding self-organization and self-adaptation are not covered by the related literature. On the other hand, we found several effective experiments in the Robotic Agents literature about complex autonomous physical systems (as swarm of robots). We used in our approach assumptions made by those studies regarding self-adaptive and self-organizing properties for physical agents domain.

We provided two instances of our proposed agent-based framework for the IoT domain: (i) quantified bananas; and (ii) traffic light control. Through those instances we have showed that our agent-based general software system satisfies its main goals:

- Autonomous things;

- Things that are able to cooperate and execute complex behavior without the need for centralized control to manage their interaction.
- Things that are able to have behavior assigned at design-time and/or at run-time.
- Feasible modelling characteristics;
 - It is possible to use our framework model to deal with complex problems in considerable time.
 - In particular, it is possible during the design phase, that one does not need to be concerned with the application domain.

5.1

Evaluation and Future Works

We implemented a framework named “FIoT” to facilitate the development process of IoT applications. We showed that FIoT has a generality potential, since we derived two completely different IoT applications from it. The former allows a human administrator to monitor and quantify things and the latter provides a dynamic controller to manage things in a distributed system.

We also showed that these applications are able of self-adapting and improving their functionalities at design-time, but they cannot adapt at runtime yet. Nevertheless, during this dissertation we presented the fundamentals to improve our framework to meet such requirement. We aim at introducing new controller types, such as state machines and temporal neural networks, as well as creating new applications to illustrate the use of them and show their benefits.

We believe that as FIoT matures, it will be able to support the development of more complex and realistic IoT applications, especially in actual distributed environments. As future work, we want to investigate the generalization capacity of our proposed framework and its application limits. As such, we need to evaluate FIoT by taking the following criteria into account:

- To investigate different applications types from prediction and control to discover which IoT application types can be created by using FIoT;
- To investigate different techniques from back-propagation and genetic algorithm to discover which types of adaptive techniques can be used;
- To evaluate which types of control can be used to meet the requirements imposed by the FIoT’s controller abstract class. In addition, we have to investigate the adaptive techniques that are suitable with each proposed control.

- To investigate which IoT applications require adaptation and the types of (self)adaptations that are useful for them.

As a result, new FIoT requirements and hot spots can appear. For example, a further application may require the management of heterogeneous environments and devices. Thus, to enable the production of new instances, we will probably need to increase the FIoT domain coverage and create new hot spots.

The centralized architecture is the major problem on God Agent specifications. There is only one God for each application. It may be a problem for applications that requires multiples devices connecting simultaneously. In future works, we can investigate existing discovery services architectures to provide FIoT applications with self-discovery protocols and more scalability.

6

Bibliography

AL-SAHILI, K.; TAYLOR, W. Evaluation of bus priority signal strategies in ann arbor, michigan. **Transportation Research Record: Journal of the Transportation Research Board**, Transportation Research Board of the National Academies, n. 1554, p. 74–79, 1996.

APPLE. **Watch**. December 2014. <https://www.apple.com/br/watch/>.

ARDUINO. **Arduino**. December 2014. <http://www.arduino.cc/>.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. **Computer networks**, Elsevier, v. 54, n. 15, p. 2787–2805, 2010.

BABAOGLU, O.; SHROBE, e. H. **First IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2007)**. Boston, MA, USA, 9-11 July 2007.

BABAOGLU, O.; SHROBE, e. H. **Ninth IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2015)**. Cambridge, MA, USA, 21-25 September 2015.

BANDYOPADHYAY, D.; SEN, J. Internet of things: Applications and challenges in technology and standardization. **Wireless Personal Communications**, Springer, v. 58, n. 1, p. 49–69, 2011.

BARRETT, M. A. et al. Big data and disease prevention: From quantified self to quantified communities. **Big data**, Mary Ann Liebert, Inc. 140 Huguenot Street, 3rd Floor New Rochelle, NY 10801 USA, v. 1, n. 3, p. 168–175, 2013.

BELEW, R. K.; MCINERNEY, J.; SCHRAUDOLPH, N. N. Evolving networks: Using the genetic algorithm with connectionist learning. In: CITESEER. In. [S.l.], 1990.

BELLIFEMINE, F. et al. **Jade Administrator's Guide**. jade.tilab.com/doc/administratorsguide.pdf, 2007.

BELLIFEMINE, F. et al. **Jade Programmer's Guide**. jade.tilab.com/doc/programmersguide.pdf, April 2010.

BEYDEDA, S.; BOOK, M.; GRUHN, V. **Model-Driven Software Development**. [S.l.]: Springer-Verlag Berlin Heidelberg, 2005.

BOE, A.; SALUNKHE, D. Ripening tomatoes: ethylene, oxygen, and light treatments. **Economic Botany**, Springer, v. 21, n. 4, p. 312–319, 1967.

BOHLI, J.; LANGENDORFER, P.; SKARMETA, A. F. Security and privacy challenge in data aggregation for the iot in smart cities. **Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems**, River Publishers, p. 225–244, 2013.

BRIOT, J.-P. et al. Experience and prospects for various control strategies for self-replicating multi-agent systems. In: ACM. **Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems**. [S.l.], 2006. p. 37–43.

BROOKS, R. A. Intelligence without reason. **The artificial life route to artificial intelligence: Building embodied, situated agents**, Lawrence Erlbaum Associates Hillsdale, New Jersey, p. 25–81, 1995.

CARLINO, D. et al. Approximately orchestrated routing and transportation analyzer: Large-scale traffic simulation for autonomous vehicles. In: IEEE. **Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on**. [S.l.], 2012. p. 334–339.

CETNAROWICZ, K.; KISIEL-DOROHINICKI, K.; NAWARECKI, E. The application of evolution process in multi-agent world to the prediction system. In: **Second International Conference on Multiagent Systems**. [S.l.: s.n.], 1996. p. 26–32.

CHEN, B.; CHENG, H. H.; PALEN, J. Mobile-c: a mobile agent platform for mobile c/c++ agents. **Software: Practice and Experience**, Wiley Online Library, v. 36, n. 15, p. 1711–1733, 2006.

DIABETES, N. I. of; DIGESTIVE; NIH, K. D. **Medline-Plus: Trusted Health Information for You**. August 2015. <https://www.nlm.nih.gov/medlineplus/gas.html>.

DORIGO, M. et al. Evolving self-organizing behaviors for a swarm-bot. **Autonomous Robots**, Kluwer Academic Publishers, v. 17, n. 2-3, p. 223–245, 2004. ISSN ISSN:0929-5593.

DUMAS, M.; HOFSTEDE, A. ter. Uml activity diagrams as a workflow specification language. In: **UML 2001 — The Unified Modeling Language. Modeling**

Languages, Concepts, and Tools. [S.l.]: Springer Berlin Heidelberg, 2001. p. 76–90.

ERIKSSON, M. An approach to software product line use case modeling. Datavetenskap, 2006.

EVANS, H.; SKILES, G. Improving public transit through bus preemption of traffic signals. **Traffic Quarterly**, v. 24, n. 4, 1970.

FANTANA, N. et al. IoT applications—value creation for industry. **Internet of Things: Covering Technologies for Smart Environments and Integrated Ecosystems**, p. 153–206, 2013.

FAYAD, M. E.; SCHMIDT, D. C.; JOHNSON, R. E. Building application frameworks: object-oriented foundations of framework design. John Wiley & Sons, Inc., 1999.

FIPA. **The Foundation for Intelligent Physical Agents.** 08 2015. [Http://www.fipa.org/](http://www.fipa.org/).

FLOREANO, D.; MATTIUSSI, C. **Bio-Inspired Artificial Intelligence. Theories, Methods, and Technologies.** [S.l.]: Cambridge: MIT Press, 2008.

FLOREANO, D. et al. Evolutionary conditions for the emergence of communication in robots. **Current biology**, Elsevier, v. 17, n. 6, p. 514–519, 2007.

FLOREANO, D.; URZELAI, J. Evolutionary robots with on-line self-organization and behavioral fitness. **Neural Networks**, Elsevier, v. 13, n. 4, p. 431–443, 2000.

FORTINO, G. et al. An agent-based middleware for cooperating smart objects. In: **Highlights on Practical Applications of Agents and Multi-Agent Systems.** [S.l.]: Springer Berlin Heidelberg, 2013. p. 387–398.

FORTINO, G.; GUERRIERI, A.; RUSSO, W. Agent-oriented smart objects development. In: **IEEE International Conference on Computer Supported Cooperative Work in Design (CSCWD).** [S.l.: s.n.], 2012.

FORTINO, G. et al. Middlewares for smart objects and smart environments: Overview and comparison. In: **Internet of Things Based on Smart Objects: Technology, Middleware and Applications.** [S.l.]: Springer, 2014. p. 1–29.

FORTINO, G.; TRUNFIO, P. **Internet of Things Based on Smart Objects: Technology, Middleware and Applications.** [S.l.]: Springer, 2014.

FRANCIOSA, J. Evaluation of the CO₂ rebreathing cardiac output method in seriously ill patients. **Circulation**, Am Heart Assoc, v. 55, n. 3, p. 449–455, 1977.

GAUNT, K.; NACSA, J.; PENZ, M. Baby lucent: pitfalls of applying quantified self to baby products. In: ACM. **CHI'14 Extended Abstracts on Human Factors in Computing Systems**. [S.l.], 2014. p. 263–268.

GOUMOPOULOS, C.; KAMEAS, A. Smart objects as components of ubicomp applications. **International Journal of Multimedia and Ubiquitous Engineering**, 2009.

GUBBIA, J. et al. Internet of things (iot): A vision, architectural elements, and future directions. **Future Generation Computer Systems**, v. 29, p. 1645–1660, 2013.

HAVENS, J. **Hacking Happiness: Why Your Personal Data Counts and How Tracking it Can Change the World**. Penguin Publishing Group, 2014. ISBN 9781101621950. Disponível em: <<https://books.google.com.br/books?id=rRQLZTnkUpYC>>.

HAYKIN, S. **Neural Networks: A Comprehensive Foundation**. Macmillan, 1994. ISBN 9780023527616. Disponível em: <<http://books.google.com.br/books?id=PSAPAQAAMAAJ>>.

HIRAKOBA, K. et al. Effect of endurance training on excessive co2 expiration due to lactate production in exercise. **European journal of applied physiology and occupational physiology**, Springer, v. 64, n. 1, p. 73–77, 1992.

HIRSCH, L. **Wearable Tech, CES2015, and the Quantified Self of Healthcare**. January 2015. [Http://www.healthcaresuccess.com/blog/physician-marketing/wearable-tech-ces2015-quantified-self.html](http://www.healthcaresuccess.com/blog/physician-marketing/wearable-tech-ces2015-quantified-self.html).

HORN, P. Autonomic computing: Ibm\'s perspective on the state of information technology. IBM, 2001.

HP. **Adaptive Enterprise: Infrastructure and management solutions for the adaptive enterprise**. [S.l.], 2003.

HUDSON, J.; DENZINGER, J. Risk management for self-adapting self-organizing emergent multi-agent systems performing dynamic task fulfillment. **Autonomous Agents and Multi-Agent Systems**, Springer, p. 1–50, 2014.

JACOB, B. et al. **A practical guide to the IBM autonomic computing toolkit**. [S.l.]: IBM, International Technical Support Organization, 2004.

JACOBSON, J.; SHEFFI, Y. Analytical model of traffic delays under bus signal preemption: theory and application. **Transportation Research Part B: Methodological**, Elsevier, v. 15, n. 2, p. 127–138, 1981.

JAEGER, T.; METZGER, A. **Open-source-Software**. [S.I.]: Beck, 2002.

JARIYASUNANT, J. et al. The quantified traveler: Using personal travel data to promote sustainable transport behavior. **University of California Transportation Center**, 2011.

JOHNSON, D.; HIPPS, N.; HAILS, S. **Helping Consumers Reduce Fruit and Vegetable Waste: Final Report**. [S.I.]: WRAP, 2008.

KAWSAR, F. et al. Design and implementation of a framework for building distributed smart object systems. **Supercomputing**, 2010.

KEPHART, J. O. Research challenges of autonomic computing. In: IEEE. **Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on**. [S.I.], 2005. p. 15–22.

KEPHART, J. O.; CHESS, D. M. The vision of autonomic computing. **Computer**, IEEE, v. 36, n. 1, p. 41–50, 2003.

KUNIAVSKY, M. **Smart Things: Ubiquitous Computing User Experience Design Book**. [S.I.]: Morgan Kaufmann, 2010.

KUURKOVA, V. Kolmogorov's theorem and multilayer neural networks. **Neural networks**, Elsevier, v. 5, n. 3, p. 501–506, 1992.

LABS, Q. S. **Quantified Self Guide to self-tracking tools**. 2014. [Http://quantifiedself.com/guide/](http://quantifiedself.com/guide/).

LEE, D.; YANNAKAKIS, M. Principles and methods of testing finite state machines—a survey. **Proceedings of the IEEE**, IEEE, v. 84, n. 8, p. 1090–1123, 1996.

LI, I.; DEY, A.; FORLIZZI, J. A stage-based model of personal informatics systems. In: ACM. **Proceedings of the SIGCHI Conference on Human Factors in Computing Systems**. [S.I.], 2010. p. 557–566.

LOPEZ, P.; PÉREZ, G. Collaborative agents framework for the internet of things. In: **Ambient Intelligence and Smart Environments**. [S.I.: s.n.], 2012. p. 191–199.

LUCENA, C. **Software engineering for multi-agent systems II: research issues and practical applications**. [S.I.]: Springer Science & Business Media, 2004.

LUPTON, D. Self-tracking modes: Reflexive self-monitoring and data practices. **Available at SSRN 2483549**, 2014.

LYYTINEN, K.; YOO, Y. Ubiquitous computing. **Communications of the ACM**, v. 45, n. 12, p. 63–96, 2002.

MARKIEWICZ, M. E.; LUCENA, C. J. P. de. Object oriented framework development. **Crossroads**, ACM, New York, NY, USA, v. 7, n. 4, p. 3–9, jul. 2001. ISSN 1528-4972. Disponível em: <<http://doi.acm.org/10.1145/372765.372771>>.

MAROCCO, D.; NOLFI, S. Emergence of communication in embodied agents evolved for the ability to solve a collective navigation problem. **Connection Science**, 2007.

MARZO, G. D. et al. **Engineering Self-Organising Systems**. Berlin: Springer, 2004.

MASSEN, T. von der; LICHTER, H. Modeling variability by uml use case diagrams. In: CITESEER. **Proceedings of the International Workshop on Requirements Engineering for product lines**. [S.l.], 2002. p. 19–25.

MASSERA, G. et al. **Designing adaptive humanoid robots through the FARSA open-source framework**. [S.l.], 2013.

MASSERA, G. et al. Farsa: An open software tool for embodied cognitive science. In: **Advances in Artificial Life, ECAL**. [S.l.: s.n.], 2013. v. 12, p. 538–545.

MATHEW, T. L. et al. Technologies for clinical diagnosis using expired human breath analysis. **Diagnostics**, Multidisciplinary Digital Publishing Institute, v. 5, n. 1, p. 27–60, 2015.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, Springer, v. 5, n. 4, p. 115–133, 1943.

MICROSOFT. **Microsoft Dynamic Systems Initiative Overview**. [S.l.], 2004.

MILLER, G. F.; TODD, P. M.; HEGDE, S. U. Designing neural networks using genetic algorithms. In: MORGAN KAUFMANN PUBLISHERS INC. **Proceedings of the third international conference on Genetic algorithms**. [S.l.], 1989. p. 379–384.

MITCHELL, S. et al. **The internet of everything for cities: Connecting people, process, data, and things to improve the ‘Livability’ of cities and communities**. 2013.

MOBILE, W. Waze. disponível em: <https://www.waze.com/pt-br>. **Acesso em**, v. 2, 2014.

MÖLLER, D. P. **Introduction to Transportation Analysis, Modeling and Simulation**. [S.l.]: Springer, 2014.

MPELKAS, C.; KENYON, E. **THE EFFECT OF LIGHT QUALITY ON THE RIPENING OF DETACHED TOMATO FRUIT**. [S.l.], 1972.

MÜLLER-SCHLOER, C. Organic computing: on the feasibility of controlled emergence. In: ACM. **Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis**. [S.l.], 2004. p. 2–5.

MüHLHäUSER, M. Smart products: An introduction. **Communications in Computer and Information Science**, 2008.

NELSON, A.; BARLOW, G.; DOITSIDIS, L. Fitness functions in evolutionary robotics: A survey and analysis. **Robotics and Autonomous Systems**, 2007.

NETO, B. et al. JAAF: A framework to implement self-adaptive agents. In: **International Conference on Software Engineering and Knowledge Engineering**. [S.l.: s.n.], 2009.

NOLFI, S. **Laboratory of Autonomous Robotics and Artificial Life**. <http://lral.istc.cnr.it/>, March 1995.

NOLFI, S.; FLOREANO, D. **Co-evolving predator and prey robots: Do ‘arms races’ arise in artificial evolution?** 1998.

NOLFI, S.; FLOREANO, D. **Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines**. Cambridge, MA, USA: MIT Press, 2000. ISBN 0262140705.

NOLFI, S.; GIGLIOTTA, O. Evorobot*. In: **Evolution of communication and language in embodied agents**. [S.l.]: Springer, 2010. p. 297–301.

NOLFI, S.; PARISI, D. Learning to adapt to changing environments in evolving neural networks. In: **Adaptive Behavior**. [S.l.: s.n.], 1997. p. 75–98.

PANAIT, L.; LUKE, S. Cooperative multi-agent learning: The state of the art. **Autonomous Agents and Multi-Agent Systems**, Kluwer Academic Publishers, Hingham, MA, USA, v. 11, n. 3, p. 387–434, nov. 2005. ISSN 1387-2532. Disponível em: <<http://dx.doi.org/10.1007/s10458-005-2631-2>>.

PARISI, D.; NOLFI, S. **Laboratory of Autonomous Robotics and Artificial Life**. 1994. Website.

PARK, J. et al. **Information Technology Convergence: Security, Robotics, Automations and Communication**. Springer, 2013. (Lecture Notes in Electrical Engineering). ISBN 9789400769960. Disponível em: <<https://books.google.com.br/books?id=6sbEBAAAQBAJ>>.

PĚCHOUČEK, M.; MAŘÍK, V. Industrial deployment of multi-agent technologies: review and selected case studies. **Autonomous Agents and Multi-Agent Systems**, Springer, v. 17, n. 3, p. 397–431, 2008.

PEZZULO, G. et al. Research on cognitive robotics at the institute of cognitive sciences and technologies, national research council of italy. **Cognitive processing**, Springer-Verlag, v. 12, n. 4, p. 367–374, 2011.

PFEIFER, R.; BONGARD, J. **How the body shapes the way we think: a new view of intelligence**. [S.l.]: MIT press, 2006.

PFISTER, C. **Getting Started with the Internet of Things: Connecting Sensors and Microcontrollers to the Cloud**. 1st. ed. [S.l.]: O'Reilly Media, Inc., 2011. ISBN 1449393578, 9781449393571.

PIMENTEL, M. et al. Methanogens in human health and disease. **The American Journal of Gastroenterology Supplements**, Nature Publishing Group, v. 1, n. 1, p. 28–33, 2012.

PINTÉR-BARTHA, A.; SOBE, A.; ELMENREICH, W. Towards the light—comparing evolved neural network controllers and finite state machine controllers. In: IEEE. **Intelligent Solutions in Embedded Systems (WISES), 2012 Proceedings of the Tenth Workshop on**. [S.l.], 2012. p. 83–87.

POLANI, D. An informational perspective on how the embodiment can relieve cognitive burden. In: IEEE. **Artificial Life (ALIFE), 2011 IEEE Symposium on**. [S.l.], 2011. p. 78–85.

POSLAD, S. Specifying protocols for multi-agent systems interaction. **ACM Transactions on Autonomous and Adaptive Systems (TAAS)**, ACM, v. 2, n. 4, p. 15, 2007.

PROTHMANN, H. et al. **Organic traffic control**. [S.l.]: Springer, 2011.

QUINN, M. et al. **Evolving Controllers For A Homogeneous System Of Physical Robots: Structured Cooperation With Minimal Sensors**. 2003.

RIEL, A. J. **Object-oriented design heuristics**. [S.l.]: Addison-Wesley Reading, 1996.

RIVERA-PELAYO, V. et al. A framework for applying quantified self approaches to support reflective learning. **Mobile Learning**, 2012.

ROCHNER, F. et al. An organic architecture for traffic light controllers. In: **GI Jahrestagung (1)**. [S.l.: s.n.], 2006. p. 120–127.

RODRIGUES, P. et al. Zigzag: A middleware for service discovery in future internet. In: **Distributed Applications and Interoperable Systems**. [S.l.]: Springer Berlin Heidelberg, 2012. p. 208–221.

RUSSELL, S.; NORVIG, P. Artificial intelligence: a modern approach. 1995.

SERUGENDO, G. D. M. et al. **A generic framework for the engineering of self-adaptive and self-organising systems**. [S.l.]: University of Newcastle upon Tyne, Computing Science, 2007.

SERUGENDO, G. D. M.; GLEIZES, M.-P.; KARAGEORGOS, A. Self-organization in multi-agent systems. **The Knowledge Engineering Review**, Cambridge Univ Press, v. 20, n. 02, p. 165–189, 2005.

SETIONO, R. Feedforward neural network construction using cross validation. **Neural Computation**, MIT Press, v. 13, n. 12, p. 2865–2877, 2001.

SOBE, A.; FEHÉRVÁRI, I.; ELMENREICH, W. Frevo: A tool for evolving and evaluating self-organizing systems. In: **IEEE Self-adaptive and Self-organizing Systems Workshop**. [S.l.: s.n.], 2012.

SOMMERVILLE, I. **Software Engineering**. Pearson/Addison-Wesley, 2004. (International computer science series). ISBN 9780321210265. Disponível em: <<http://books.google.com.br/books?id=fIJQAAAAMAAJ>>.

STAMFORD, C. **2014 Hype Cycle for Emerging Technologies Maps the Journey to Digital Business**. <http://www.gartner.com/newsroom/id/2819918>, August 2014.

STEELS, L. **ECAGENTS: Embodied and Communicating Agents**. [S.l.], 2004.

SUAREZ, F.; SPRINGFIELD, J.; LEVITT, M. Identification of gases responsible for the odour of human flatus and evaluation of a device purported to reduce this odour. **Gut**, BMJ Publishing Group Ltd and British Society of Gastroenterology, v. 43, n. 1, p. 100–104, 1998.

SWAN, M. Sensor mania! the internet of things, wearable computing, objective metrics, and the quantified self 2.0. **Journal of Sensor and Actuator Networks**, v. 1, n. 3, p. 217–253, 2012.

SWAN, M. The quantified self: fundamental disruption in big data science and biological discovery. **Big Data**, Mary Ann Liebert, Inc. 140 Huguenot Street, 3rd Floor New Rochelle, NY 10801 USA, v. 1, n. 2, p. 85–99, 2013.

SWAN, M. Connected car: Quantified self becomes quantified car. **Journal of Sensor and Actuator Networks**, Multidisciplinary Digital Publishing Institute, v. 4, n. 1, p. 2–29, 2015.

TANGERMAN, A. Measurement and biological significance of the volatile sulfur compounds hydrogen sulfide, methanethiol and dimethyl sulfide in various biological matrices. **Journal of Chromatography B**, Elsevier, v. 877, n. 28, p. 3366–3377, 2009.

TELECOM. **JAVA Agent DEvelopment Framework**. 08 2015. [Http://jade.tilab.com/](http://jade.tilab.com/).

THEGUARDIAN. **Can the internet of things save us from traffic jams?** April 2015. [Http://www.theguardian.com/technology/2015/apr/20/internet-of-things-traffic](http://www.theguardian.com/technology/2015/apr/20/internet-of-things-traffic).

THRUN, S. et al. Stanley: The robot that won the darpa grand challenge. In: **The 2005 DARPA Grand Challenge**. [S.l.]: Springer, 2007. p. 1–43.

TRIANNI, V.; NOLFI, S. Engineering the evolution of self-organizing behaviors in swarm robotics: A case study. **Artificial Life**, MIT Press, v. 17, n. 3, p. 183–202, 2011.

UCI. **Machine Learning Repository**. Dezembro 2014. [Https://archive.ics.uci.edu/ml/datasets.html](https://archive.ics.uci.edu/ml/datasets.html).

UGULINO, W. et al. Wearable computing: accelerometers' data classification of body postures and movements. In: **Advances in Artificial Intelligence-SBIA 2012**. [S.l.]: Springer, 2012. p. 52–61.

VAIDYA, J.; RANINGA, P.; BHALANI, J. Revolution technique for internet of things “6lowpan”.

VALADARES, C.; NETTO, M.; LUCENA, C. A normative and self-organizing piloting model for virtual network management. In: **Workshop-Escola de Sistemas de Agentes, seus Ambientes e aplicações, IEEE**. [S.l.: s.n.], 2013. p. 41–46.

VELLOSO, E.; RAPOSO, A.; FUKS, H. Web of things: The collaborative interaction designer point of view. In: **1st Workshop of the Brazilian Institute for Web Science Research**. [S.l.: s.n.], 2010.

VIDE, M. D.; NOLFI, S. Emergence of communication in teams of embodied and situated agents. In: WORLD SCIENTIFIC. **The Evolution of Language: Proceedings of the 6th International Conference (EVLANG6), Rome, Italy, 12-15 April 2006**. [S.I.], 2006. p. 198.

VIMALA, T.; RAJARAM, U. Self powered energy aware internet of things. **Journal of Computer Science**, Science Publications, v. 10, n. 9, p. 1819, 2014.

VISION, C. **Astah* community-Free UML Modeling Tool**. 2011.

WEISS, G.; SEN, S. **Adaptation and Learning in Multi-Agent Systems**. [S.I.]: Springer-Verlag, 1995.

WHITE, J. E. Telescript technology: The foundation for the electronic marketplace. **General Magic white paper**, v. 282, 1994.

WOOLDRIDGE, M. **An introduction to multiagent systems**. [S.I.]: John Wiley & Sons, 2009.

WOOLDRIDGE, M.; JENNINGS, N. R. Intelligent agents: Theory and practice. **The knowledge engineering review**, Cambridge Univ Press, v. 10, n. 02, p. 115–152, 1995.

YAO, X. Evolving artificial neural networks. **Proceedings of the IEEE**, IEEE, v. 87, n. 9, p. 1423–1447, 1999.

YOSHIDA, K. et al. Statistical method application to knowledge base building for reactor accident diagnostic system. **Journal of Nuclear Science and Technology**, Taylor & Francis, v. 26, n. 11, p. 1002–1012, 1989.

ZAHEDI, K.; AY, N. Quantifying morphological computation. **Entropy**, Multidisciplinary Digital Publishing Institute, v. 15, n. 5, p. 1887–1915, 2013.