Information Sciences 378 (2017) 161–176

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

FIoT: An agent-based framework for self-adaptive and self-organizing applications based on the Internet of Things

Nathalia Moraes do Nascimento*, Carlos José Pereira de Lucena

Software Engineering Lab (LES), Department of Informatics, Pontifical Catholic University of Rio de Janeiro (PUC-Rio), RJ - 22453-900

ARTICLE INFO

Article history:
Received 6 November 2015
Revised 23 September 2016
Accepted 19 October 2016
Available online 20 October 2016

Keywords:
Internet of things (IoT)
Multi-agent system
Machine learning
Self-organizing
Self-adaptive
Quantified things

ABSTRACT

Billions of resources, such as cars, clothes, household appliances and even food are being connected to the Internet forming the Internet of Things (IoT). Subsets of these resources can work together to create new self-regulating IoT applications such as smart health, smart communities and smart homes. However, several challenging issues need to be addressed before this vision of applications based on IoT concepts becomes a reality. Because many IoT applications will be distributed over a large number of interacting devices, centralized control will not be possible and so open problems will need to be solved that relate to building locally operating self-organizing and self-adaptive systems. As an initial step in creating IoT applications with these features, this paper presents a Framework for IoT (FIoT). The approach is based on Multi-Agent Systems (MAS) and Machine Learning Techniques, such as neural networks and evolutionary algorithms. To illustrate the use of FIoT, the paper contains two different IoT applications: (i) Quantified Things and (ii) Smart traffic control. We show how flexible points of our framework are instantiated to generate these IoT application.

1. Introduction

The Internet of Things (IoT) refers to a global infrastructure of physical things interconnected through the Internet [51]. The central notion is that, in the near future, billions of resources, such as cars, household appliances, food and machinery will be connected to the Internet and interact by sharing information about themselves and their environments. IoT will make it possible to develop a large number of applications [1] based on smart homes and cities, e-health, and environmental monitoring to name some examples. Smart traffic management is an example of a smart city application, which aims to provide intelligent transportation through real-time traffic information and path optimization [20]. IoT is a new, exciting and emerging approach that we anticipate will soon be available [20].

According to Atzori et al. [1], most IoT applications are not yet developed because they need to be able to scale to incorporate millions of devices. Centralized solutions do not scale well and systems with a fixed configuration are inadequate as the environment may be continuously evolving [14]. For example, an autonomous application for traffic management depends on the traffic light controllers adapting to traffic situations [50] changing over time. Thus, several challenging issues related to autonomy and local control still need to be addressed before the IoT vision becomes a reality [1,35].

* Corresponding author.
E-mail addresses: nnascimento@inf.puc-rio.br, nathyecomp@gmail.com (N.M. do Nascimento), lucena@inf.puc-rio.br (C.J.P. de Lucena).

http://dx.doi.org/10.1016/j.ins.2016.10.031
0020-0255/

The current focus of IoT research is not applications but operational technology solving problems related to limited Internet traffic capacity, communication protocols, and network architecture [27]. For example, Gubbia et al. [20] discuss the open challenges and future directions in IoT and mention global addressing schemes, cloud storage, and wireless power as the key elements of current IoT research. In their opinion, a self-adaptive system of systems is an example of the key application outcomes expected only in the next decade.

In [14], one of the few papers that discuss the interaction of things using the Internet, the authors address some open issues which are needed to build elements operating autonomously and capable of coping with a changing environment. In an effort to highlight these issues, a new terminology associated with IoT is emerging: Smart Objects or Smart Things (SOs). They represent loosely coupled and decentralized systems of cooperating things. Fortino and Trunfio [14] discuss SOs and define them as autonomous, physical digital objects augmented with sensing/actuating, processing, interpretation, storage, and networking capabilities.

Part of the approach to develop and deploy IoT-based SOs is to define new frameworks/middleware [3,56] to support rapid prototyping of Iot applications. Frameworks are general software systems that consist of abstract and concrete classes, which can be adapted or extended to create more specific applications. According to Ian Sommerville [56], "the sub-system is implemented by adding components to complete parts of the design and instantiating the abstract classes in the framework."

A few framework/middleware approaches have been proposed to support the creation of an SO-based IoT infrastructure [15,8,48]. Fortino et al. [15] analyze these approaches and discuss their limitations. One restriction is that none of these methods appears to have been used in the design of a complex application scenario such as one supporting traffic flow using a large number of SOs. For example, the authors in [16,17] developed a middleware system for SOs and state that support of distributed computing entities is the key and novel feature of their approach. However, to illustrate the use of their architecture, they present a simple case study, which refers to a smart office environment consisting of only two cooperating SOs. The paper does not show complex scenarios, where SOs must cope with a changing environment and where a complex organization is required.

Fortino et al. [15] state that novel software engineering methodologies need to be defined for development of dynamic SO-based IoT systems such as manufacturing control and traffic management [9], where self-organization is a necessity. Thus, we focus on smart systems within the IoT domain, providing a framework that supports systems with more autonomy through self-adaptive and self-organizing properties. Our middleware approach is called the "Framework for Internet of Things" (FIoT) and is based on Multi-Agent Systems (MAS) and adaptive techniques that have been commonly used in robotics to develop autonomous embodied agents [31,13].

The objective of FIoT is the creation of diverse applications, such as controllers for car traffic, machinery, public lighting, smart appliances and smart homes. Hence, the framework allows the creation of autonomous controllers for groups of homogeneous SOs. To illustrate the use of FIoT, we will present examples from two IoT application domains: (i) Quantified Things and (ii) Smart Cities.

Multi-agent Systems (MAS) are used to model real-world and social systems [62] and provide a useful paradigm for managing large and distributed information systems [10]. In addition, an agent can have characteristics, such as autonomy and social ability, which make MAS suitable for structures requiring self-organization.

MAS can be simple, but can become more robust over time if they are developed according to self-organizing principles. However, the models used in Self-Organizing MAS tend to be very complex [10], although the use of learning and evolution strategies in a Self-Organizing MAS can reduce this complexity [62,7,61].

Robotic researchers have studied autonomous self-organizing physical systems and their problems [47,59] with a view to developing methods that allow robots to learn to perform complex tasks automatically. A primary focus of contemporary autonomous robotic research is to develop machine learning methods. Recently, a new machine learning method called Evolutionary Robotics (ER) [31,13,42] has appeared and gained both academic and industrial attention. The primary goal of ER is to develop methods for automatically synthesizing intelligent autonomic robotic systems. ER has the potential to lead to the development of robots that can adapt to unknown environments.

This paper is organized as follows. Section 2 provides a literature survey of related work. Section 3 describes the FIoT model and framework. Section 4 discusses how the proposed framework can be used to create IoT instances by presenting some experiments. The paper ends with concluding remarks and suggestions for future work.

2. Related work

This section provides a review of the literature on frameworks and MAS that combines artificial intelligence and IoT. The section also contains a description of a framework for physical agents that is not focused on IoT, but which provides physical systems with self-organizing and self-adaptive properties.

Fortino et al. [15] describe how middleware can support the development of SO-based IoT systems. Some middleware layers in the Fortino paper are described and compared, based on a set of requirements for smart environments and SOs. Other authors in [15] list four different frameworks, which provide for the efficient development and deployment of SOs namely: ACOSO (Agent-based Cooperating Smart Objects) [16,17], FedNet [24], Ubicomp [19] and Smart Products [34]. These frameworks use different architectural models: ACOSO is agent-oriented and event-driven, FedNet is service-oriented, while Ubicomp and Smart Products are component-based.

Fortino et al. [15] also discuss the limitations of these frameworks or middleware layers in the management of a very large number of cooperating SOs. According to the authors, the scale and dynamic nature of SO-based IoT systems require novel software engineering methodologies. In another publication [18] the same authors state that specific abstractions for system/component evolution are required as progressive change is a typical property of SO-based systems. The authors further argue that agent-oriented methodologies could be used to formalize a development method for SO-based IoT systems as agent-based systems can cope with the main requirements for IoT systems namely: interoperability, abstraction, collective intelligence and experience-based learning.

A framework for IoT systems based on a MAS paradigm is also proposed in [27]. The authors list some requirements for developing IoT applications, which include the domain analysis of the proposed framework. According to the authors, requirements are the acquisition of measurements and related data from devices, processing and translation of the data into useful information, and actuation of devices within the environment. Moreover, the approach shows that agents have characteristics that meet those requirements, such as perception, autonomy and social ability. Although the paper provides motivation for our approach, this publication only offers a brief description of possible framework components. The authors do mention that there is still the need to detail every component and provide each one with intelligent characteristics. Our approach provides intelligent components to develop IoT applications through adaptation and organizational algorithms.

The Framework for Autonomous Robotics Simulation and Analysis (FARSA) [33] developed by the Italian Institute of Cognitive Science and Technologies [45] has properties that make it useful for supporting FIoT. This framework was created to support research in the areas of embodied cognition, adaptive behavior, language, and action. A set of studies of Evolutionary Robotics [31,40,32] was developed using FARSA and related software. Most of these experiments use a group of embedded agents each of which evolves to solve a collective problem.

Other research related to our approach is the Framework for Evolutionary Design (FREVO) [55]. Sobe et al. present FREVO as a multi-agent tool for evolving and evaluating self-organizing simulated systems. The authors state that FREVO allows a framework user to select a target problem evaluation, controller representation and an optimization method. However, FREVO concentrates on evolutionary methods for agent controllers. As a result, this tool can only provide off-line adaptation and evolve simulated environments.

Unfortunately, we are not able to reuse these platforms to control SOs since the platforms are for the simulation of robotic agents and lack some communication structures since they do not support heterogeneous platforms required by current networks, such as desktop, web, mobile and micro-controller boards.

3. FIoT: framework for Internet of Things

In this Section, we survey IoT application requirements using both the current literature and personal experience. We then describe our proposed agent-based model to create IoT systems and 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 a Framework for IoT (FIoT), which is presented in this Section.

During framework development, three stages must be considered: (i) domain analysis, (ii) framework design, and (iii) framework instantiation [30]. Domain analysis surveys the requirements of an application area. In the framework design stage, Unified Modeling Language (UML) diagrams [3] are used to specify FIoT structure, behavior, and architecture. UML use case [60] and UML activity diagrams [12] are also used to describe the main ideas behind FIoT. In addition, an analysis of the kernel ("frozen-spots") and flexible points ("hot-spots") of the FIoT framework is presented. "Frozen-spots" are immutable and must be part of each framework instance. "Hot-spots" represent the flexible points of a system, which are customized to generate a specific application [30].

According to [30], choice of hot spots in a framework design is critical, as bad choices will inevitably lead to unnecessary complexity. The instantiation stage is presented in Section 4, where new application instances are generated through implementation of the FIoT's hot spots.

3.1. Domain analysis

As was emphasized in Sections 1 and 2, we used the material in [14] and [27] as basis for the domain analysis. We also consider the requirements for the development of self-organizing and self-adaptive applications proposed by the authors in [54].

From a software engineering perspective, IoT systems are distributed systems consisting of components (things) that may be static or mobile, collect data about themselves and their environments autonomously and take actions [25]. A smart IoT system can make decisions based on collected data and use dynamic reconfiguration to improve its performance.

All IoT applications share common features, such as connecting to the environment and collecting data; but they also have features that vary according to specific applications. To assist in the development of self-organizing and self-adaptive IoT applications, we developed a list of requirements (R) for domain discovery. The steps shown in Table 1 show the major steps in domain discovery. Each major step labeled R_1 through R_5 may be followed by labeled sub-steps which provide explanatory details.

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

Table 1
Requirement domain discovery.

Requirements	Domain discovery step
R_1	Design-time description (problem domain)
$R_{1.1}$	To analyze environmental conditions that are associated with the problem goal such as temperature or constituent gases
$R_{1.2}$	To define how to collect environmental conditions such as a micro-controller board and sensors
R_2	Decentralization and Inter-operability
R_3	Autonomous things
$R_{3.1}$	Things should be capable of autonomously sensing/monitoring themselves and their environments
$R_{3.2}$	Actuation over the environment
R_4	Self-adapting capability
$R_{4.1}$	Each individual component or the whole system should be capable of identifying any new condition, failure, or problem within the component's environment
$R_{4.2}$	Run-time capability of reasoning and of acting/adapting
R_5	To design software to allow the system
$R_{5.1}$	To recognize things in the environment
$R_{5.2}$	To acquire the data from things that are collecting environmental data
$R_{5.3}$	To interface with device sensors
$R_{5.4}$	To process and translate the data into useful contextual information

3.2. Agent-based model

We developed an agent-based model as a foundation for generating different kinds of applications for IoT. Our approach is completely based on MAS and artificial intelligence paradigms such as neural networks and evolutionary algorithms. Our goal is to provide mechanisms that recognize and manage things in the environment automatically. As depicted in Fig. 1, our model consists of three layers: physical (L1), communication (L2), and application (L3). Each thing in the environment (physical layer) is recognized and controlled by agents in the application layer.

The physical layer consists of simulated or real devices (also named smart things/objects) and environments. In order to model the physical layer, the project designer has to define the features of smart things as well as the features of their surrounding environment. The designer must decide on the environmental conditions that need to be monitored such as temperature, relative humidity or traffic. Once these conditions are chosen the designer can specify performance criteria for the smart things that collect data or make changes to the environment.

The communication layer specifies the communication among agents in the application layer. Each smart thing has one address, so an agent can access this address to obtain and set the necessary information to control the thing. We suggest

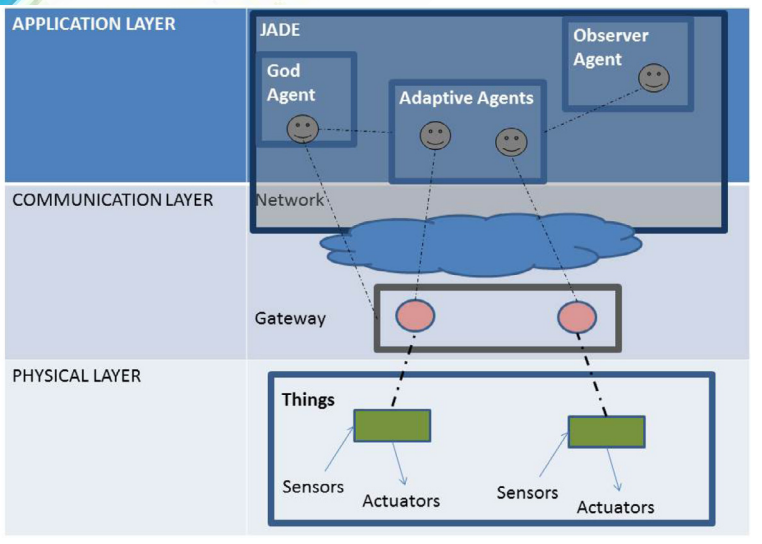


Fig. 1. An agent-based model to generate IoT applications.

the Java Agent Development Framework (JADE) [2] and its variants (JADEX, LEAP) to implement the communication layer among agents and smart things in order to address heterogeneous devices such as PCs, PDAs, resource constrained-devices or Smartphones.

The application layer uses a MAS to provide services, such as collecting, analyzing and transmitting data from several sensors to the Internet and back again. Our intention is to produce MAS-based applications with decentralized, autonomous, self-organizing features. In addition, we provide the ability to create physical agents capable of interacting dynamically with complex environments using approaches from robotics. We recommend developing controllers at the application layer to allow autonomous management of things in the physical layer.

3.3. Central idea for the framework design

A FIoT application requires three types of agents: (i) God Agents [49]; (ii) Adaptive Agents; and (iii) Observer Agents [29]. The primary role of the God Agent is to detect new things that are trying to connect to the system and make that connection. For this connection to occur a thing sends a message to the God Agent's IP address and the God Agent creates an Adaptive Agent to control each connected thing. An Adaptive Agent is an agent embodied in a Smart Thing, according to the description provided in [63]. While a device represents the adaptive agent's body, a JADE software agent contains the adaptive agent's controller. The God Agent sets the controller or the "brain" of an Adaptive Agent based on its type where type is often determined by the number of connected sensors and actuators. Therefore, controller creation is a flexible point in FIoT system implementation.

Neto et al. [39] developed a framework to implement self-adaptive software agents based on the autonomic computing principles proposed by IBM [23]. These adaptive agents have a control loop composed of four activities: collect, analyze, plan and execute, which are briefly described next.

- **Collect:** collect application data;
- **Analyze:** analyze the collected data by trying to detect problems;
- **Plan:** decide on an action in the case of problems; and
- **Execute:** change the application because of executed actions.

We customized the control loop from the IBM proposal [23] 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 could be a finite state machine (FSM) or an artificial neural network (ANN), as shown in Fig. 2.

Our Adaptive Agent must execute three key activities in sequence namely: (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. The 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 based on the controller output. An action (effector activity) can be to interact with other agents, to send messages, or to set actuator devices, thus making changes to the environment.

The Observer Agent examines the environment to determine if the system is meeting its global goals. If the goals are not being met then the Observer Agent provides instructions to the Adaptive Agents to change their behavior. In this way the Adaptive Agents become self-organizing and behavior may emerge that was not defined at design-time.

Some researchers [13,47,59,42,11,43] have investigated the emergence of cooperative or competitive self-organizing MAS. One way to generate a cooperative self-organizing MAS is to perform the adaptive process based on collective evaluations. Self-organizing systems have global goals. Thus, we investigate during the adaptation process if a collection of agents is working together to achieve a global goal. If the system needs to adapt, the adaptation is performed for the whole MAS. If we conduct the adaptive process based on individual evaluations, the agents may compete with each other. This possibility of competition is the main reason for providing an Observer Agent to evaluate the global behavior of the collection of Adaptive Agents and to conduct the adaptation process for the whole system. Therefore, an Observer Agent's main goal is to verify if the Adaptive Agents need to adapt. When the actions of agents are far from what an Observer Agent expects, it executes a supervised or unsupervised learning method, such as back-propagation or a genetic algorithm.

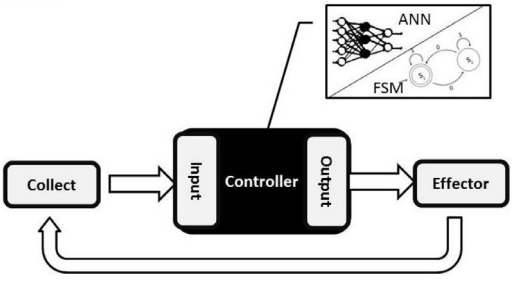


Fig. 2. Control loop provided by the FIoT framework.

Table 2
How the model and FIoT meet the IoT requirements.

Req.	Layer	Description
R_1	L_1	The physical layer design defines the problem domain.
R_2	L_2 and L_3	The application layer uses a Multi-Agent System (MAS) to provide services. The interoperability can be supported by the JADE Framework.
R_3	L_3	Adaptive agents autonomously manage devices, without the need of a human administrator.
$R_{4.1}$	L_3	The Observer Agent can evaluate the whole system, groups of Adaptive Agents or an individual agent.
$R_{4.2}$	L_3	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.
$R_{5.1}$	L_2 and L_3	The God Agent automatically identifies things that are trying to connect to the system.
$R_{5.2}$	L_2 and L_3	Adaptive Agents collect data from the things at the physical layer.
$R_{5.3}$	L_1 and L_3	Adaptive Agents have access to a set of sensors and actuators previously registered. The things at the physical layer provide their type. Then, the Adaptive Agents know how to process the data sent by the things.
$R_{5.4}$	L_3	Adaptive Agents are intelligent agents that make use of a controller to process the data coming from the devices.

The process of adaptation consists of generating new configurations for all the Adaptive Agents' controllers and testing how agents will behave in this environment. The Observer Agent sets the Adaptive Agents' controllers with the configuration that conforms to the Adaptive Agents' desired global action. While the Observer Agent looks for new controller configurations, Adaptive Agents continue normal execution.

The Observer Agent is tightly coupled to the application being developed. The evaluation process has to be implemented according to the expected global solution. For example, if an application for automobile traffic control has the goal of reducing urban traffic congestion, the evaluation may be performed based on the number of vehicles that had finished their routes in a specific period. Another variable activity is the generation of new configurations for controllers, which depend on the applied adaptation technique.

As agents execute specific activities to communicate with smart things at the physical layer, these smart things must execute the following sequential activities:

- Connect to the Internet
- Send a message to the GodAgent, reporting the smart things controller type. 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 for a message from the GodAgent containing the address of the smart thing's Adaptive Agent. This smart thing will then use this address to send and receive the next messages in a cycle. The smart thing will
  - Send a message with data from sensors
  - Wait for a message with data to set the smart thing's actuators

Table 2 summarizes the model and framework description in this Section, and presents how they meet the requirements listed in Section 3.1, based on the layers. FIoT meets the requirements associated with the communication layer (L_2) and application layer (L_3).

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 smart things. FIoT extends JADE, a Java framework to implement MAS through the development of JADE agents, the behavior 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 present the key FIoT classes [56] of the main packages.

The class diagram depicted in Fig. 3 illustrates the FIoT classes associated with the creation of agents and their execution loops. As described before, the FIoT agent classes are the GodAgent, ObserverAgent and AdaptiveAgent classes, which extend the FIoTAgent class. Then, FIoT agents can access and make changes to the list of controllers (ControllerList class). This list stores all controllers already created by the GodAgent for each type of smart thing such as a chair with one temperature sensor, lamp with one presence sensor and one alarm actuator.

All agents execute sequential behaviors, named as ExecutionLoop: GodLoop, AdaptiveLoop and ObserverLoop classes. The sequential behavior is a JADE behavior that supports the composition of activities [2]. Thus, the ExecutionLoop is a sequence of smaller actions. For example, for Adaptive Agents, these execution loops are composed of collect, decide and effect activities.

The class diagram depicted in Fig. 4 illustrates the collection of behaviors already developed. Activities such as evaluation and controller adaptation are examples of hot spots. Thus, new strategies for evaluation and adaptation can be developed to be used by agents. The God Agent's execution loop performs three behaviors: "Detect," "CreateAgent," and "ControllerProvision."

While ObserverAgents access the ControllerList to adapt controller configurations through ChangeControllers behavior, an AdaptiveAgent uses the ControllerList to get its controller, set data input, and obtain the calculated output.

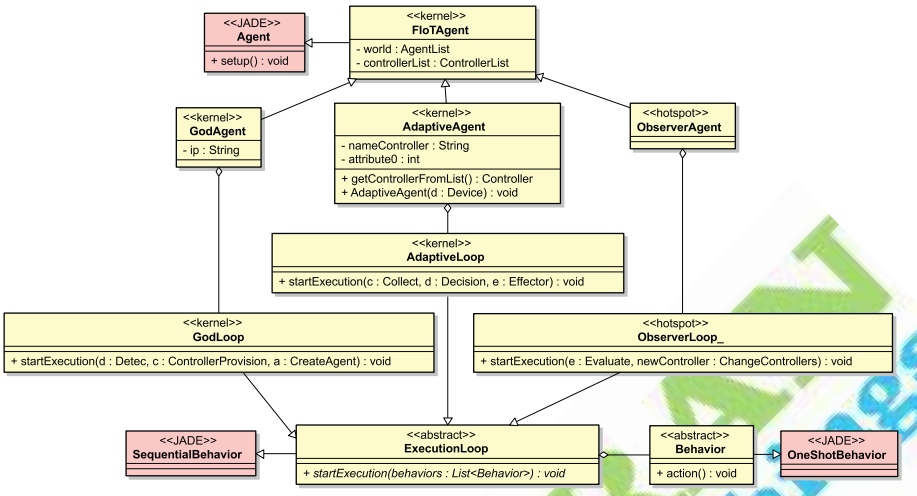


Fig. 3. Class diagram of FIoT - Agents.

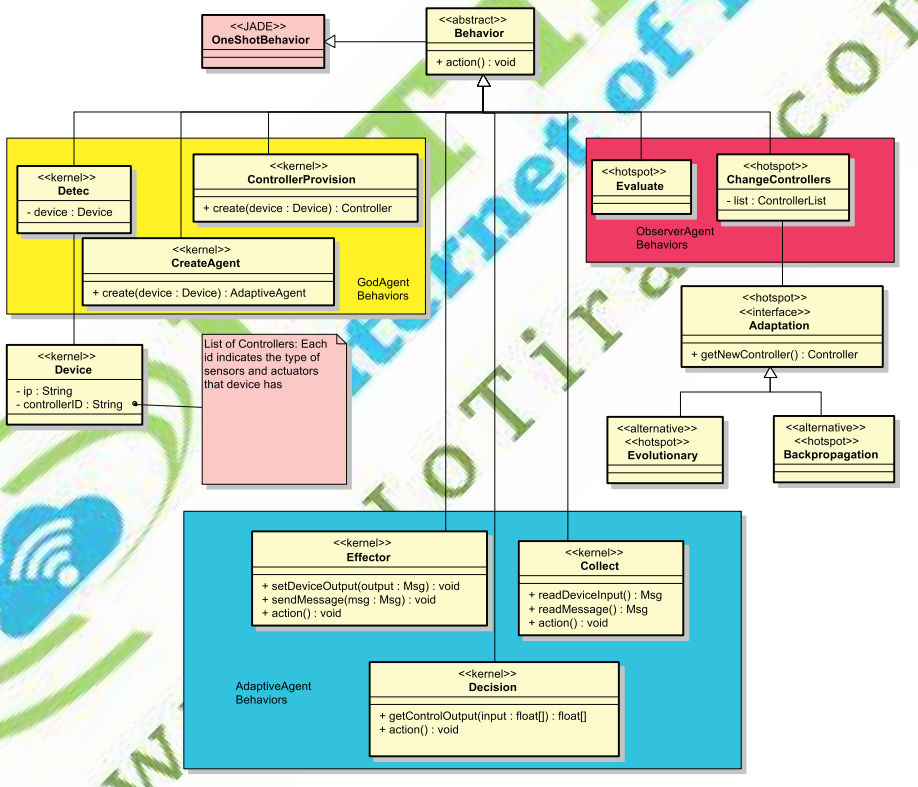


Fig. 4. Class diagram of FIoT - Behaviors.

The class diagram depicted in Fig. 5 illustrates the controller classes. Agents as virtual homogeneous things can use the same controller to make decisions. For example, where similar smart lamps have to be managed, the same ANN controller can be used by Adaptive Agents. The GodAgent stores the smart lamp controller in ControllerList as "lampNeuralNetwork." If there is another group of devices, the GodAgent has to use a different controller.

4. Evaluation: illustrative examples

We evaluate FIoT by implementing its hot spots or flexible points to generate two different applications. As discussed in Section 3, the framework instantiation is the last stage in the development of a framework [30].

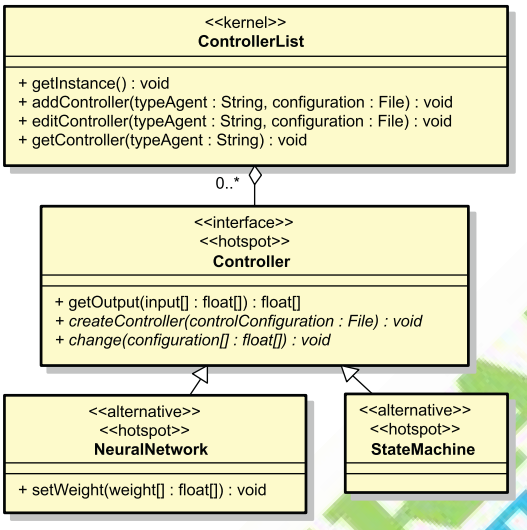


Fig. 5. Class diagram of FIoT - Controllers.

We consider the following IoT applications or instances in the FIoT evaluation process: (i) Quantified Things and (ii) traffic flow in a Smart City. This Section presents a brief description of each example by completing the hot spots and illustrating how the generated applications adhere to the proposed framework.

4.1. FIoT's Instances

The frozen spots are part of the FIoT kernel and 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 decisions 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
- Evaluation by the Observer Agent
- Controller adaptation by the Observer Agent

Thus, to create an FIoT instance, a developer has to implement/choose: (i) a control module such as a neural network or finite state machine; (2) an adaptive technique to train the controller; and (iii) an evaluation process such as a genetic algorithm that performs evaluation via a fitness function. As shown in this Section, we only evaluate applications using a neural network. However, we implemented FIoT to support the use of finite state machines (fsm), since we provided an abstract controller class. For example, a framework user can implement a Mealy machine (a special case of an fsm) and use an evolutionary algorithm to evolve its structure and transition probabilities [55]. Thus, it is possible to generate applications using different configurations. A framework user should select a configuration that works better toward solving a given problem.

4.2. Quantified Things

A new trend in IoT is "Quantified Self" [52], or continuous self-tracking. For example, a person equipped with sensors could allow personal health parameters to be available on the Internet and evaluate his/her health status. Since this health information is available, the community started to ask about the types of inferences that are possible if selected groups of people share their tracked data, thus producing the "Quantified Us" movement [26].

What happens if instead of asking "What can **people** learn when pooling data?" [21], we start to ask, "What can **things** 'learn' when pooling data?" These things could be machines, where one machine could predict a fault based on collective data sharing or things could also be plantations, where the owner of one plantation can predict the crop yield based on the

history of crops from other plantations. Thus, "Quantified Things" is proposed as an extension of the quantify movement and a design for this type of data sharing is presented using FIoT, as depicted in Fig. 6.

Devices scattered across different environments are managed by adaptive agents. In turn, these agents populate a cloud database with sensor data from devices and their inferences. If new devices connect to the system, they can access this database and make predictions based on this historical data.

IoT applications, such as environmental monitoring could incorporate an architecture to capture Big Data [6] from sensors and add value to that data [46]. Quantified Things are an example of an IoT approach that could benefit from a Big Data approach. More detail on Quantified Things, Big Data and MAS are provided in [38].

Quantified Bananas

To illustrate a "Quantified Things" experiment using FIoT, we address the distribution and storage of fresh fruit, where the objective is to minimize the loss of fruit too mature to be consumed. Recently, Ruan and Shi [53] proposed a conceptual framework based on IoT technologies for monitoring in-transit fruit freshness in e-commerce deliveries. They studied the process of fresh fruit transportation and mapped 1024 simulated scenarios to investigate freshness assessment scenarios for different types of fruit by using case and deduction-based learning. However, they do not verify the effectiveness of their proposed framework with actual scenarios.

In our work, we focus on "Quantified Bananas" where data about bananas are shared. The data is used to predict the number of days that pass before the bananas spoil under specific environmental conditions. The data from each banana storehouse are the temperature, humidity, luminosity and concentration of gases such as methane and hydrogen. The physical layer of this scenario showing the sensors is in Fig. 7.

Adaptive agents use a three-layer feed-forward neural network to predict the number of days before bananas spoil. The data from a number of sensors is used as the input to this three-layer network.

The ObserverAgent monitors executions, checking if predictions have been correct or not. To compare results, a user system needs to be informed about how long the monitored bananas lasted. If results are not similar, the ObserverAgent executes the adaptation process to adjust the network parameters. The technique used is supervised learning (back-propagation), since it has historic data to compare results and predict new ones.

Table 3 shows how the "Quantified Banana" application adheres to the proposed framework by extending the FIoT flexible points. The hot spot "making evaluation" is developed for this application as an individual evaluation. The Observer Agent maintains a data set containing input from Adaptive Agents and neural predictions. Based on this historical data, for each adaptive agent execution, the Observer Agent evaluates if an individual result requires a collective adaptation.

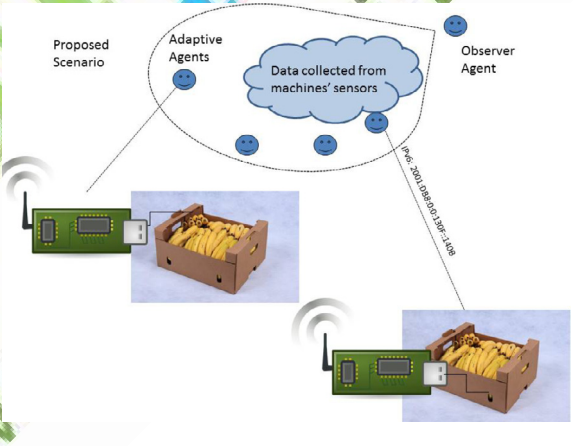


Fig. 6. An instance of FIoT to create "Quantified Things".

Fig. 7. Physical layer for "Quantified Bananas" instance.

Table 3
Case I: Flexible points.

Framework	Application
Controller	Three Layer Neural Network
Making Evaluation	Individual Evaluation: for each agent evaluation, the Observer Agent concludes if all Adaptive Agents need to adapt or not
Controller Adaptation	Supervised Learning (Backpropagation)

4.2.1. Experimental description

We carefully selected the individual bananas and categorized them by appearance. Each experiment was conducted under different conditions. The experiments were created combining four parameters, as shown in Table 4: (i) dark but in a closed or open box; (ii) refrigerated or at room temperature; (iii) stored with rotten fruit or not; and (iv) stored with ripe fruit or not.

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 conducted in a dark place, in the refrigerator, and stored with rotten fruit.

4.2.2. Training results

We verified the training process by comparing the predictions with values for actual fruit shelf life for each experiment. This comparison is shown in Table 5, where the column "Expected Results" shows the "actual" shelf life, the column "Actual Results" shows the predictions provided by the neural network, and the column "Error" the difference between the actual and predicted shelf life, based on normalized values.

As shown in Table 5, differences between expected and actual results are fairly close. The largest errors were present in experiments four and six, corresponding to an error of 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 back-propagation algorithm needs an extensive data set to train a neural network.

4.3. Smart city

The Smart Cities concept is frequently associated with IoT [20,4,36]. A Smart City often has many sensors scattered throughout the city collecting information on activities such as water and energy consumption [44], and vehicle and human traffic flow. For instance, cars, traffic lights and pedestrians could all be connected via the Internet, collecting and sharing data, such as GPS data from cars and smartphones, traffic light intervals, and camera images [58]. Based on this data, traffic lights could operate at different intervals, and GPS consoles and smartphones could offer drivers alternate routes. The reduction of urban traffic congestion is the main goal of this smart approach to traffic management. In order to demonstrate

Table 4
Experimental description.

Experiment	*Dark*	*Fridge*	*Rotten fruit*	*Ripe fruit*
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	

Table 5
Results of backpropagation execution.

Experiment	*Expected results*	*Actual results*	*Error*
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

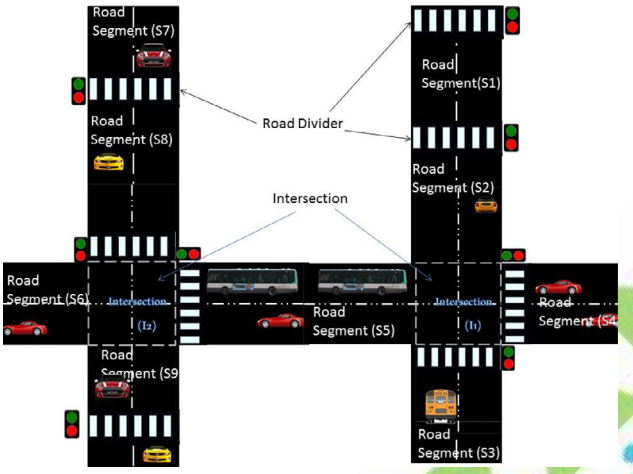


Fig. 8. Traffic elements.

the application of FIoT to smart city services, we apply IoT principles to create a self-managing car traffic control application [58,28], where vehicle traffic is monitored through sensors and the data is used to control the operation of traffic lights.

According to Standford-Clark, an IBM engineer, the problem is not to change the traffic lights, but to anticipate the "interconnection of unintended consequences." Thus, most traffic light sequences are set via longer term algorithms, taking the whole of the road network into account [58]. Unfortunately, determining such sequences is a non-trivial and time-consuming task, as one must account for a wide range of factors such as traffic density, pedestrian flows, and road complexity. In contrast, FIoT creates dynamic controllers for homogeneous things situated in a distributed environment by using a decentralized and adaptive process.

Car traffic application

In this subsection, we describe a simulated car traffic scenario, where Fig. 8 depicts the elements that are part of the application namely: 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 subdivides one segment into two segments. We modeled our scenario as a graph, in which the edges represent segments and nodes represent road dividers and intersections.

4.3.1. Smart road segment

Each road segment in the simulation has a micro-controller that is used to calculate the number of vehicles per time period, interact with the closest segment, and change the segment's traffic light. The GodAgent creates an Adaptive Agent for each road segment in the scenario. Independent of the application, an Adaptive Agent has to execute three tasks: data collection, decision-making and action enforcement. For this experiment, the first task for the micro-controller in each segment consists of receiving data related to its vehicle flow, data from its neighboring segment and its current traffic light color. To make decisions, Adaptive Agents use a "three-layer feedforward" with a feedback loop [22]. Feedback occurs because the output of a segment's traffic light color influences its next network input, as shown in Fig. 9.

By using a recurrent neural network with an input layer of three neurons, we are providing a memory for these agents, where the goal is to remember the duration of a specific color traffic-light. The middle layer of the neural network has

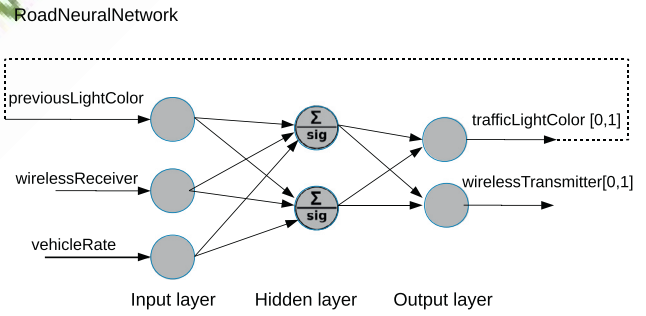


Fig. 9. Adaptive agent's neural controller.

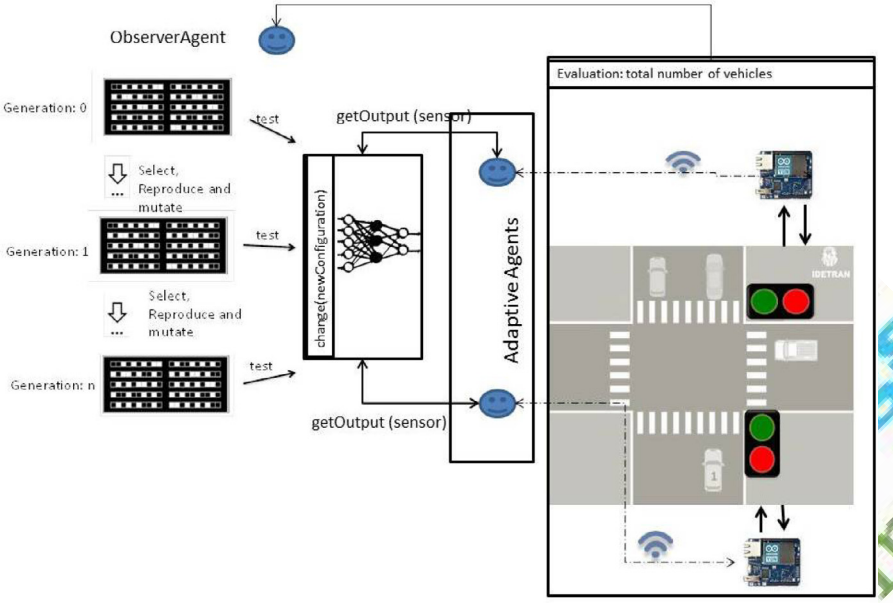


Fig. 10. Performing an adaptive process to adjust the Traffic Neural Controller weights. Figure adapted from [41]. P.7.

Table 6
Case II: Flexible points.

Framework	Application
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

two neurons connecting the input and output layers. These neurons provide an association between sensors and actuators, which represent the system policies that can change based on the neural network configuration.

4.3.2. Observeragent: adaptive process

Evolutionary algorithms are used to support the design of system features automatically. By using a genetic algorithm, we expect that a policy for controlling the traffic lights, with a simple communication system among road segments, will emerge from this experiment. Therefore, no system feature such as the effect of road segment on vehicle rate was specified at design-time. The evaluation and adaptation process performed by the Observer Agents is depicted in Fig. 10.

The weights in the neural network used by the Adaptive Agents vary during the adaptation process, as the ObserverAgent applies a genetic algorithm to find a better solution. The ObserverAgent contains a pool of candidates to represent the network parameters. The ObserverAgent evaluates each candidate based on the number of cars that finished their routes after the simulation ends. Table 6 summarizes how the "Car Traffic Control" application adheres to the proposed framework, while extending the FIoT flexible points.

4.3.3. Experiment

The first simulation scenario is shown in Fig. 11. The urban road network is based on a small section of a real city, Feira de Santana, Bahia, Brazil.

The graph representing the road network consists of 31 nodes and 48 segments. Each segment is one-way and links two nodes. For simulation purposes, we chose 15 nodes as departure points (yellow nodes) and two as destinations (red nodes). Each segment has a traffic light. In the graph, the green and red triangles represent the traffic light colors.

We started with 1000 vehicles in this experiment, where the capacity of each road segment is 75 vehicles. The role of the vehicles is to complete their routes.

4.3.4. Evolutionary algorithm: simulation parameters

Since we are proposing a simple experiment, the process of testing, selecting and reproducing candidates is iterated only 20 times. During the test stage, each team of 48 Adaptive Agents representing the road segments in the scenario is allowed to "live" for 30 cycles by using a candidate, as shown in Fig. 10. As each car departure and target is randomly selected and can affect the test result, more than one test is performed for each candidate.

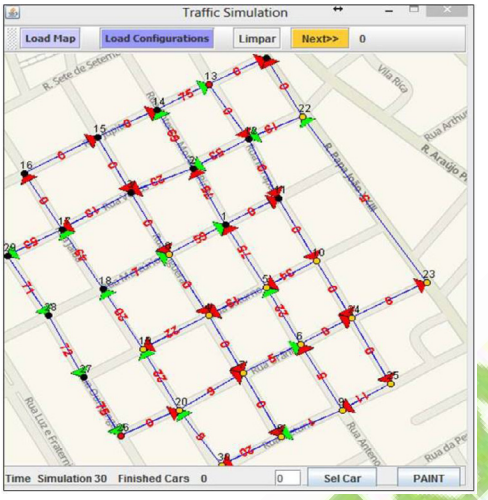


Fig. 11. Simulation urban road network. Adapted from Waze (2014) [37].

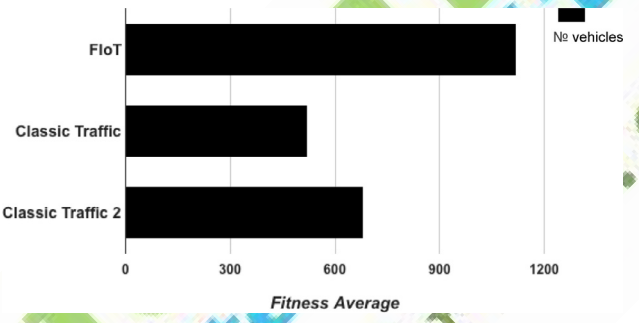


Fig. 12. Comparison of the FIoT approach and conventional systems in the first scenario.

The fitness of each candidate consists of the number of vehicles that finished their route at the end of the simulation. The individuals with the highest fitness are selected to generate the new generation by using crossover and mutation.

4.3.5. Evaluation of the best candidate

After executing the evolutionary algorithm, Adaptive Agents acquire the ability to produce a satisfactory set of traffic light decisions in order to improve urban traffic flow. We selected the best candidate solution from the evolutionary process to provide comparisons between our approach and "conventional" traffic light policies. Conventional policies use fixed-time control, where the sequence of phases (red or green) and their durations [64] are constant. We simulated two fixed-time approaches.

The first conventional approach changes all traffic lights colors in every cycle. The second changes all the traffic-light 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 and each one of the two "conventional" solutions. Fig. 12 presents the number of vehicles that finished their routes.

By using the evolved agents approach, the number of vehicles that finished their routes is higher than using the other conventional approaches.

5. Conclusion

IoT is an emerging approach using information technology, which has the potential for significant impact [57]. IoT applications must easily scale to cope with what are likely to be large numbers of interacting and evolving devices. Based on our experience with the examples presented in this paper, we believe that self-organizing and self-adapting IoT multi-agent applications are an appropriate way to cope with the growth and evolution of IoT systems.

There appear to be few research results in the literature about agent-based architectures for IoT [27,17] and none appears to present the design of a significant case study involving a large number of cooperating smart things. Thus, the results from the literature do not demonstrate scalability, where many things are interacting and coping with a dynamic environment. Further, important features mentioned in this paper regarding self-organization and self-adaptation are also not covered in the published literature. However, we found several experiments in the Robotics literature about complex autonomous

physical systems such as a swarm of robots that may be applicable. We used assumptions made in those studies about self-adaptive and self-organizing properties for the physical agents' domain.

We provided two examples of our proposed IoT agent-based framework: (i) quantified bananas; and (ii) traffic light control. These examples illustrate that our agent-based general software system satisfies its main goals namely:

- 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 reasonable time.
  - In particular, it is possible during the design phase to model a general IoT application.

6. Future work

We believe that as FIoT concepts mature, variants of the basic FIot will be able to support the development of more complex and realistic IoT applications, especially in distributed environments. As future work, we want to investigate the possible generalizations of our proposed framework and assess the framework's limits [5]. As such, we need to evaluate FIoT by considering:

- the different prediction and control application types that can be created using FIoT;
- the different learning algorithms to discover which types of adaptive techniques are applicable;
- the control types that can be used to meet the requirements imposed by the FIoT's controller abstract class and the types of adaptive techniques that are suitable with each proposed control; and
- the types of IoT applications that require adaptation and the types of (self)adaptations that are useful.

As a result of this future research we expect new FIoT requirements and hot spots to appear. For example, an expanded 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.

Acknowledgments

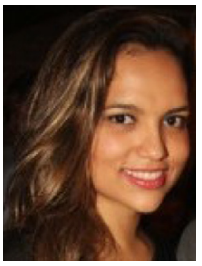
This work has been supported by the Laboratory of Software Engineering (LES) at PUC-Rio. Our thanks to CNPq, CAPES, FAPERJ and PUC-Rio for their support through scholarships and fellowships.

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

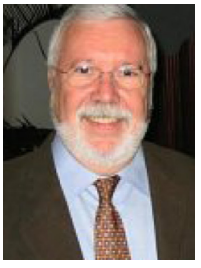
References

[1] L. Atzori, A. Iera, G. Morabito, The internet of things: a survey, Comput. Netw. 54 (15) (2010) 2787–2805.
[2] F. Bellifemine, G. Caire, T. Trucco, G. Rimassa, R. Mungenast, Jade Administrator's Guide, JADE, 2007. Available in http://jade.tilab.com/doc/administratorsguide.pdf.
[3] S. Beydeda, M. Book, V. Gruhn, Model-Driven Software Development, Springer-Verlag Berlin Heidelberg, 2005.
[4] J. Bohli, P. Langendorfer, A.F. Skarmeta, Security and privacy challenge in data aggregation for the iot in smart cities, Internet of Things (2013) 225–244.
[5] J.-P. Briot, N.M. Nascimento, C.J.P. de Lucena, A multi-agent architecture for quantified fruits: design and experience, in: 28th International Conference on Software Engineering & Knowledge Engineering (SEKE'2016), SEKE/Knowledge Systems Institute, PA, USA, 2016, pp. 369–374.
[6] C. Cecchinel, M. Jimenez, S. Mosser, M. Riveill, An architecture to support the collection of big data in the internet of things, in: Services (SERVICES), 2014 IEEE World Congress on, IEEE, 2014, pp. 442–449.
[7] K. Cetnarowicz, K. Kisiel-Dorohinicki, E. Nawarecki, The application of evolution process in multi-agent world to the prediction system, in: Second International Conference on Multiagent Systems, 1996, pp. 26–32.
[8] H. Derhamy, J. Eliasson, J. Delsing, P. Priller, A survey of commercial frameworks for the internet of things, in: 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA), IEEE, 2015, pp. 1–8.
[9] G. Di Marzo Serugendo, M.-P. Gleizes, A. Karageorgos, Self-organization in multi-agent systems, Knowl. Eng. Rev. 20 (02) (2005) 165–189.
[10] G. Di Marzo, A. Karageorgos, O. Rana, F. Zambonelli, Engineering Self-Organising Systems, Springer, Berlin, 2004.
[11] M. Dorigo, V. Trianni, E. Şahin, R. Groß, T.H. Labella, G. Baldassarre, S. Nolfi, J.-L. Deneubourg, F. Mondada, D. Floreano, et al., Evolving self-organizing behaviors for a swarm-bot, Auton Robots 17 (2–3) (2004) 223–245.
[12] M. Dumas, A. ter Hofstede, Uml Activity Diagrams as a Workflow Specification Language, in: UML 2001 The Unified Modeling Language. Modeling Languages, Concepts, and Tools, Springer Berlin Heidelberg, 2001, pp. 76–90.
[13] D. Floreano, C. Mattiussi, Bio-Inspired Artificial Intelligence. Theories, Methods, and Technologies, Cambridge: MIT Press, 2008.
[14] G. Fortino, P. Trunfio, Internet of Things Based on Smart Objects: Technology, Middleware and Applications, Springer, 2014.
[15] G. Fortino, A. Guerrieri, W. Russo, C. Savaglio, Middlewares for Smart Objects and Smart Environments: Overview and Comparison, in: Internet of Things Based on Smart Objects: Technology, Middleware and Applications, Springer, 2014, pp. 1–29.
[16] G. Fortino, A. Guerrieri, W. Russo, Agent-oriented smart objects development, in: IEEE International Conference on Computer Supported Cooperative Work in Design (CSCWD), 2012, pp. 907–912.
[17] G. Fortino, A. Guerrieri, M. Lacopo, M. Lucia, W. Russo, An Agent-based Middleware for Cooperating Smart Objects, in: Highlights on Practical Applications of Agents and Multi-Agent Systems, Springer Berlin Heidelberg, 2013, pp. 387–398.
[18] G. Fortino, A. Guerrieri, W. Russo, C. Savaglio, Towards a development methodology for smart object-oriented iot systems: A metamodel approach, in: Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on, IEEE, 2015, pp. 1297–1302.
[19] C. Goumopoulos, A. Kameas, Smart objects as components of ubicomp applications, Int. J. Multim. Ubiquitous Eng. (2009).
[20] J. Gubbia, R. Buyyab, S. Marusic, M. Palaniswami, Internet of things (iot): a vision, architectural elements, and future directions, Future Gen. Comput. Syst. 29 (2013) 1645–1660.

[21] J. Havens, Hacking Happiness: Why Your Personal Data Counts and How Tracking It Can Change the World, Penguin Publishing Group, 2014.
[22] S. Haykin, Neural Networks: A Comprehensive Foundation, Macmillan, 1994.
[23] P. Horn, Autonomic computing: IBM\'s Perspective on the State of Information Technology, Technical Report, IBM, 2001.
[24] F. Kawsar, T. Nakajima, J. Hyuk Park, S. Yeo, Design and implementation of a framework for building distributed smart object systems, Supercomputing (2010).
[25] M. Kuniavsky, Smart Things: Ubiquitous Computing User Experience Design Book, Morgan Kaufmann, 2010.
[26] V. Lee, Learning Technologies and the Body: Integration and Implementation in Formal and Informal Learning Environments, Routledge Research in Education, Taylor & Francis, 2014.
[27] P. Lopez, G. Prez, Collaborative agents framework for the internet of things, in: Ambient Intelligence and Smart Environments, 2012, pp. 191–199.
[28] D.P. Möller, Introduction to Transportation Analysis, Modeling and Simulation, Springer, 2014.
[29] C. Müller-Schloer, Organic computing: on the feasibility of controlled emergence, in: Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, ACM, 2004, pp. 2–5.
[30] M.E. Markiewicz, C.J.P. de Lucena, Object oriented framework development, Crossroads 7 (4) (2001) 3–9, doi:10.1145/372765.372771.
[31] D. Marocco, S. Nolfi, Emergence of communication in embodied agents evolved for the ability to solve a collective navigation problem, Conn. Sci. (2007).
[32] G. Massera, T. Ferrauto, O. Gigliotta, S. Nolfi, Designing adaptive humanoid robots through the farsa open-source framework, Technical Report, Institute of Cognitive Sciences and Technologies (CNR-ISTC), 2013.
[33] G. Massera, T. Ferrauto, O. Gigliotta, S. Nolfi, farsa: An open software tool for embodied cognitive science, in: Advances in Artificial Life, ECAL, 12, 2013, pp. 538–545.
[34] M. Mhlhuser, Smart products: an introduction, Commun. Comput. Inf. Sci. (2008).
[35] J. Mineraud, O. Mazhelis, X. Su, S. Tarkoma, A gap analysis of internet-of-things platforms, Comput. Commun. (2016).
[36] S. Mitchell, N. Villa, M. Stewart-Weeks, A. Lange, The internet of everything for cities: connecting people, process, data, and things to improve the livability of cities and communities, 2013.
[37] W. MOBILE, Waze. available in: https://www.waze.com/, (2014).
[38] N.M. Nascimento, C. Jos, P. de Lucena, H. Fuks, Modeling quantified things using a multi-agent system, in: 2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT), 1, IEEE, 2015, pp. 26–32.
[39] B. Neto, A. Costa, M. Netto, V. Silva, C. Lucena, Jaaf: a framework to implement self-adaptive agents, in: International Conference on Software Engineering and Knowledge Engineering, 2009, pp. 212–217.
[40] S. Nolfi, D. Parisi, Learning to adapt to changing environments in evolving neural networks, in: Adaptive Behavior, 1997, pp. 75–98.
[41] S. Nolfi, O. Gigliotta, Evorobot*, in: Evolution of Communication and Language in Embodied Agents, Springer, 2010, pp. 297–301.
[42] S. Nolfi, J. Bongard, P. Husbands, D. Floreano, Evolutionary Robotics, Springer International Publishing, Cham, pp. 2035–2068. doi:10.1007/978-3-319-32552-1_76.
[43] L. Panait, S. Luke, Cooperative multi-agent learning: the state of the art, Auton. Agent Multi. Agent Syst. 11 (3) (2005) 387–434, doi:10.1007/s10458-005-2631-2.
[44] J.F.D. Paz, J. Bajo, S. Rodrguez, G. Villarrubia, J.M. Corchado, Intelligent system for lighting control in smart cities, Inf. Sci. (Ny) 372 (2016) 241–255. http://dx.doi.org/10.1016/j.ins.2016.08.045.
[45] G. Pezzulo, G. Baldassarre, A. Cesta, S. Nolfi, Research on cognitive robotics at the institute of cognitive sciences and technologies, national research council of italy, Cogn. Process 12 (4) (2011) 367–374.
[46] I. Portugal, P. Alencar, D. Cowan, A survey on domain-specific languages for machine learning in big data, arXiv preprint:1602.07637 (2016).
[47] M. Quinn, L. Smith, G. Mayley, P. Husbands, P.H. Nds, Evolving controllers for a homogeneous system of physical robots: Structured cooperation with minimal sensors, 2003,
[48] M.A. Razzaque, M. Milojevic-Jevric, A. Palade, S. Clarke, Middleware for internet of things: a survey, IEEE Internet Things J. 3 (1) (2016) 70–95.
[49] A.J. Riel, Object-oriented Design Heuristics, 335, Addison-Wesley Reading, 1996.
[50] F. Rochner, H. Prothmann, J. Branke, C. Müller-Schloer, H. Schmeck, An organic architecture for traffic light controllers., in: GI Jahrestagung (1), 2006, pp. 120–127.
[51] P. Rodrigues, Y.-D. Bromberg, L. Rveillre, D. Ngru, Zigzag: A Middleware for Service Discovery in Future Internet, in: Distributed Applications and Interoperable Systems, Springer Berlin Heidelberg, 2012, pp. 208–221.
[52] D. Rose, Enchanted Objects: Design, Human Desire, and the Internet of Things, Scribner, 2014.
[53] J. Ruan, Y. Shi, Monitoring and assessing fruit freshness in iot-based e-commerce delivery using scenario analysis and interval number approaches, Inf. Sci. (Ny) (2016).
[54] G.D.M. Serugendo, J. Fitzgerald, A. Romanovsky, N. Guelfi, A Generic Framework for the Engineering of Self-adaptive and Self-organising Systems, University of Newcastle upon Tyne, Computing Science, 2007.
[55] A. Sobe, I. Fehrvri, W. Elmenreich, Frevo: a tool for evolving and evaluating self-organizing systems, in: IEEE Self-adaptive and Self-organizing Systems Workshop, 2012, pp. 105–110.
[56] I. Sommerville, Software Engineering, International computer science series, Pearson/Addison-Wesley, 2004.
[57] C. Stamford, 2014 Hype Cycle for Emerging Technologies Maps the Journey to Digital Business, Technical Report, Gartner, 2014. Available in http://www.gartner.com/newsroom/id/2819918.
[58] TheGuardian, Can the internet of things save us from traffic jams?, April, 2015, (Available in http://www.theguardian.com/technology/2015/apr/20/internet-of-things-traffic).
[59] V. Trianni, S. Nolfi, Engineering the evolution of self-organizing behaviors in swarm robotics: a case study, Artif. Life 17 (3) (2011) 183–202.
[60] T. von der Maßen, H. Lichter, Modeling variability by uml use case diagrams, in: Proceedings of the International Workshop on Requirements Engineering for product lines, Citeseer, 2002, pp. 19–25.
[61] H. Wang, X. Wang, X. Hu, X. Zhang, M. Gu, A multi-agent reinforcement learning approach to dynamic service composition, Inf. Sci. (Ny) 363 (2016) 96–119.
[62] G. Weiss, S. Sen, Adaptation and Learning in Multi-Agent Systems,, Springer-Verlag, 1995.
[63] M. Wooldridge, An Introduction to Multiagent Systems, John Wiley & Sons, 2009.



Nathalia Moraes do Nascimento is pursuing a PhD Degree in Computing at Pontifical Catholic University of Rio de Janeiro (PUC-Rio). Researcher at Software Engineer Laboratory (LES PUC-Rio). She received a MSc degree from PUC-Rio in 2015 (August) and a BSc degree in Computer Engineering from the State University of Feira de Santana (UEFS) in 2013.



Carlos J. P. de Lucena received a BSc degree from the Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil, in 1965, an MMath degree in computer science from the University of Waterloo, Canada, in 1969, and a PhD degree in computer science from the University of California at Los Angeles in 1974. He has been a full professor in the Department of Informatics at PUC-Rio since 1982.