iMobile: A Framework to Implement Software Agents for the iOS Platform

Pedro Augusto da Silva e Souza Miranda, Andrew Diniz da Costa, Carlos José Pereira de Lucena
Laboratório de Engenharia de Software – LES
Pontifícia Universidade Católica do Rio de Janeiro
Rio de Janeiro, Brazil
email: pedro.augusto@les.inf.puc-rio.br, email: andrew@les.inf.puc-rio.br, email: lucena@inf.puc-rio.br

***Abstract*—Appropriate implementation of software agents for the iOS platform able to represent their main features, such as distribution, autonomy and pro-activity, is still an open issue. Therefore, this paper proposes the iMobile Framework that allows the creation of software agents for applications (apps) made for iPhone and iPad devices. A developed scenario demonstrates the applicability of the proposed framework, where an app allows registration of issues identified in some company. The main goal of the software agents is to speed up the process of contacting people, as quickly as possible, to solve critical issues.**

Keywords-mobile computing; software agent; framework; iOS platform.

I. INTRODUCTION

Currently, the development of complex apps for mobile platforms (e.g., iOS and Android), which have autonomous, pro-active and distributed entities, is common. Taking this scenario into consideration, the use of software agents [2][3][4][16][39][40] is expected.

In the literature, there are several proposals [1][5][6][15][17][18][29][30] that help the development of software agents. However, few solutions are directed to mobile environments, such as agents being executed in smartphones and tablets. Even with the exponential evolution of hardware used for manufacturing, mobile device memory, processing and disk drive space are still concerns. In most cases, an important reason is that a mobile operating system (OS) has very restricted policies regarding what an application may or may not do with its memory, processor and disk drive. This situation is different from desktop applications, where services may run in the background until the user stops the application, or when the OS is shut down.

One of the best-known mobile-oriented frameworks is JADE-Leap [5]. It is an extension of the Java Agent Development Framework (JADE) framework [6], but with a limited feature set. JADE already offers support to the implementation of autonomous and pro-active agents. Such limitation of features by JADE-Leap was defined to avoid performance issues when ran on tablets or smartphones. Furthermore, JADE-Leap was developed in Java [9] and it may not be used on the iOS platform.

In 2014, Apple presented a new programming language for the iOS platform, called Swift. It came to replace the Objective-C, the language used for many years to develop the iOS apps. Thereafter, in 2015, a library called GameplayKit [35] was presented to offer the collection of foundational tools and technologies for building games. One of the facilities provided was to simulate characters of a game to react based on high-level goals and to react to their surroundings. GameplayKit used the agent term to represent these characters, however, it did not reproduce all characteristics that represent software agents [2][3][4], such as sociability.

According to [31], Swift is a programming language that quickly became one of the fastest growing languages in history. Swift has been an open source since 2015. It was widely adopted by developers and now is one of the top 10 most-searched languages on the internet [32].

Considering the growth of this language and the complexity of developing iOS apps, approaches that help to improve software agents for the iOS platform are expected. Then, based on this context, this paper presents the iMobile, a practical development framework. It allows the creation of software agents for the iOS platform using native resources, such as the Bonjour API [20], which helps the discovery of devices and services published on networks. Consequently, the agents' development for the iOS environment will be easier and faster with iMobile.

This paper is organized as follows: Section II describes the related works that helped to identify which information could and should be represented in an agent framework for the iOS platform. In Section III, the proposed iMobile framework is explained in detail. In Section IV, a use case scenario that extends the iMobile framework and uses agents to speed up the process of contacting people, as quickly as possible, in order to solve critical issues in a company, is presented. Finally, in Section V, the conclusions and future works are presented.

II. RELATED WORKS

In order to propose a framework that could help to create software agents to the iOS platform, several frameworks offered in the literature, that allow the agents' development, were studied. The main works that greatly influenced our proposal are presented below.

As mentioned in Section I, JADE-Leap [5] is an extension of the JADE Framework [6]. It allows the use of software agents in a variety of mobile platforms, such as Android and MIDP (Mobile Information Device Profile) [12]. In the JADE framework, agents inhabit containers and every JADE agent application must have a main container, in which all secondary agents on the system are registered. Containers are Java processes that provide the JADE run-time and all the necessary services for hosting and executing agents [13].

From the JADE framework, you may create software agents that are able to trade messages with each other, change behaviors and migrate to another container. JADE agents are

 ISBN: 978-1-61208-590-6

Foundation for Intelligent Physical Agents (FIPA) compliant [14], which means that they obey certain structures when they are trading messages. These messages, called Agent Communication Language (ACL), meet FIPA's ACL standards. The ACL message standard considers a protocol that allows message exchanges between agents, even if they are in different systems. Each ACL message has, at least, one receiver, one language and one ontology, which are the message context and the message content itself. More details about that standard can be seen at [28].

JADE LEAP extends all these JADE features for a mobile environment using the same concept of containers. In other words, JADE LEAP, just as JADE, forces the user to have a fixed IP server in order to make its agents communicate with each other. Then, for years, JADE and JADE LEAP were useful and mature agents' frameworks, besides being excellent standards to begin understanding how to work with software agents. However, they were created from the Java language and cannot be used for the iOS platform.

Another known solution analyzed was the Jadex framework [15]. The JADE framework was developed from the Java language and it implements the Belief, Desire and Intent (BDI) concept [16] for the software agents. BDI stands for:(i) Belief, (ii) Desire and (iii) Intention. Belief represents the group of information that an agent has concerning its environment and itself. Desire represents the agent's wishes and directly influences its actions to achieve its goals. Intention represents the selected desire to be achieved by the agent. Therefore, the intention is the momentary state of the agent until the plan has been executed and its results analyzed.

It was very important to learn about JADE, Jadex and implementing examples of both frameworks (for instance, examples with until 30 agents simultaneously executing) to better understand how a multi-agent software works and how agents communicate between themselves.

BDI4JADE [18] is another framework that, as the name implies, extends the JADE framework. BDI4JADE allows JADE developers to design multi-agents systems with agents that implement BDI architecture. The main motivation for the BDI4JADE creation is because the JADE framework is not Domain Specific Language (DSL), in other words, it is not dependent like JADEX and JACK [17] on another agent framework. The work in [18] informs that BDI4JADE, even though based on general purpose programming languages, limits the integration with up-to-date, available technologies and the use of advanced features of the underlying programming language. Therefore, BDI4JADE agents are called BDI agents. These agents have a reasoning cycle. The algorithm used for the implementation of BDI4JADE reasoning cycle is presented by the work in [18]. The reasoning cycle contains six main procedures, that involve revising agent's beliefs and desire management and generation, removing agent's undesired desires and agents' intentions generation (selected desires within all desires and the unselected ones are still desires, but not intentions) and updating intentions status (working, finished, fail, etc.). For more details of the framework, please see [16] and [18].

All the proposals mentioned are known frameworks in the community and they allow the development of software agents. However, none of these approaches allow the creation of agents for the iOS platform. Moreover, considering such a platform, to have a framework that could use native resources would bring a set of advantages that have not been properly exploited, such as better performance, easier understanding to develop from the language used, in order to create new iOS apps, and the reuse of known libraries that help to represent agents' concepts (e.g., communication among agents, distribution, etc.).

In Table I, the main features of the iMobile Framework, in comparison with other known software agent frameworks, are presented. It is important to notice that iMobile was created to offer support to the iOS platform and, in addition, the framework followed recommendations offered by FIPA, such as providing services of yellow pages to meet agents. It is also relevant to mention that since the iMobile framework uses Bonjour Networking services to solve network communication between agents, there is no need for a main container like the JADE framework. Agent services are published in the same DNS-Zone [42]. For local network communication, a DNS-Zone is not necessary.

TABLE I. COMPARISON OF IMOBILE WITH OTHER AGENT FRAMEWORKS

Features	Agent Frameworks		
	iMobile	***JADE/JADE Leap***	***JADEX***
Offering multiple agent's behaviors	Yes	Yes	Yes
Offering distributed agents	Yes	Yes	Yes
Allowing the communication between one or more agents	Yes	Yes	Yes
Representing the BDI concept	No	No	Yes
Offering iOS support	Yes	No	No
Offering services of yellow pages	Yes	Yes	Yes

III. IMOBILE FRAMEWORK

This section presents the iMobile Framework that allows the development of software agents for the iOS platform. The iMobile agents are able to manage the gathering and to exchange and display information from a given iOS app. Furthermore, the framework helps to discover agents that are on the same network to identify rendered services and to exchange messages.

 ISBN: 978-1-61208-590-6

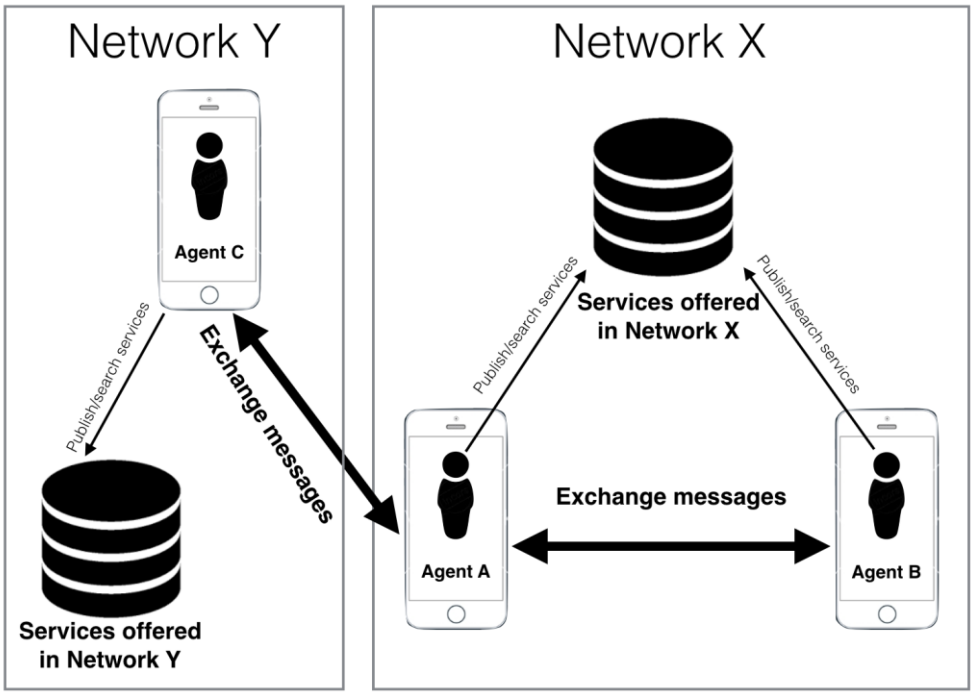

Figure 1. Use case scenario to apply the iMobile framework.

Figure 1 shows an example that illustrates the main idea behind how iMobile may be used. In this image, there are three iPhones connected to some network (Network X or Y). Each iPhone has an iMobile agent that may exchange messages with agents on the same or a different network. For each network, there is a database with all services offered for agents connected on the same network. Then, to discover such available services, it is not necessary to directly exchange messages with other agents. To discover and publish services and exchange messages among agents, iMobile uses an API [21] offered by Apple and which is explained in detail on the next subsection. Therefore, the overall communication is performed and it is not necessary to create containers and platforms as it is to other frameworks, such as JADE, JADE-Leap and JADEX.

In order to present how the iMobile framework was created, an overview of the architecture is demonstrated in subsection A. Then, in subsection B, the hot-spots and frozen-spots [36] are explained in detail. In subsection C, a guiding step to instantiate the framework is described.

A. Architecture Overview

Figure 2 presents the framework's class diagram. It is possible to realize that a software agent (*Agent* class) may execute a set of behaviors (*Behavior* class). Two different types of behaviors are offered by the framework: Cyclic Behavior (*CyclicBehavior* class) and One-Shot Behavior (*OneShotBehavior* class). The first one is a type of action that executes in loop. On the other hand, the second executes an action only once.

At any time, an agent may send or receive some message (*AgentMessage* class) from some agent. *AgentMessage* implements the *NSCoding* protocol [24], which is already offered by Apple, to allow the serialization of messages to be sent to other agents.

iMobile allows that agents, on the same network, may exchange messages among themselves faster, aiming to offer a polite communication solution. The technology that helps this improved communication is the Bonjour API. It is a solution created by Apple for the iOS platform and which is a zero-configuration network solution [21], that has a very important role on the design of this framework. Zero-configuration means that you may publish services on a network without the need of any network troubleshooting.

In order to use the Bonjour API, iMobile offers the *BonjourServer* class, which is used by the agent to look for services offered by agents (from the *NSNetServiceBrowser* class) to publish its services (from the *NSNetService* class) and to send and receive messages to/from other agents (from the *NSNetService* class). *BonjourServer* applies the Façade design pattern [37][40] to offer all the necessary methods to agents to perform these actions. In addition, iMobile offers a protection against object substitution attacks, because *AgentMessage* class implement the *NSSecureCoding* protocol [42]. If developers desire to provide some additional treatment of the data (e.g., performing cryptography), they may use some native library that is available at [43].

 ISBN: 978-1-61208-590-6

B. Hot-spots and Frozen-spots

Below, the hot-spots defined by iMobile and registered by using the stereotype <<hot-spot>>, in Figure 2, are explained in detail.

Representing a software agent (*Agent* Class): There are two ways of developing an agent: (i) creating an instance of the *Agent* class, and (ii) defining a new class that extends the *Agent* class to represent the desired agent. Each agent has a name, may offer a set of services and may execute a set of behaviors.

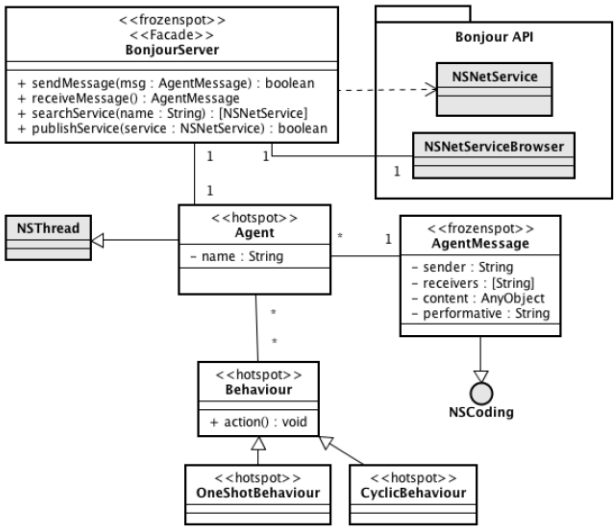

Figure 2. Class diagram of the iMobile framework.

Behaviors executed by agents (*Behavior* Class): Such class represents the behavior of an agent. Behaviors are actions that agents may execute on an application. This class has a method called *action* that, when executed, initiates the agent's behavior. Their subclasses should have it implemented.

Behavior executed once (*OneShotBehavior* Class): It represents a generic behavior offered by the framework that is executed only once. When its execution is finished, such behavior is removed from the agent's behaviors list.

Behavior executed in cycle (*CyclicBehavior* Class): Another generic behavior offered by the framework. This behavior is executed in a loop with a predetermined timer to wait between the loops.

Next, iMobile's frozen-spots are explained below.

Manager to use the Bonjour API (*BonjourServer* Class): This class was created to access a set of features offered by Bonjour API: (i) publication of services provided by agents, (ii) search of services offered by other agents, (iii) message sending and receiving to/from agents. *BonjourServer* uses BSD Sockets [23] to create a listening socket for the agent, which allows the agent to receive messages from other agents and filter out unwanted messages.

Messages that are sent and received by agents (*AgentMessage* Class): Such class represents the communication protocol between agents. Most properties of the *AgentMessage* class are basic agent information as name, current service, service type, sender, receiver, data and content. The content property accepts any type of object to be sent to another agent. These objects, that are stored in luggage, should be instances of classes that implement the *NSCoding* protocol in order to send it through the network.

C. Using iMobile

These are the main steps that should be performed to use the iMobile framework:

1. Defining which agents will be used in the solution to be developed. Agents may be either created from instantiated objects using the *Agent* class, or from other classes that extend such class;
2. Creating behaviors that will be executed by the agent. For instance, such behaviors may request services and send messages to other agents;
3. Defining which services each agent will be able to offer;
4. Defining which type of service the agent will search. It is necessary for the creation of each agent to inform these service types.

IV. USE SCENARIO: ISSUE TRACK

Daily, it is common for companies to raise issues and some of them have high priority. When such issues are quickly solved, a lot of damage may be avoided, such as losing money. According to [33][34], one of the top reasons for closing companies is the bad management of how issues are handled.

Taking this context into consideration, we implemented an iOS app with software agents that help speed up the process of contacting people, as quickly as possible, to solve critical issues. First, we explain the main solution idea. Secondly, we present in detail how the proposed app extended the iMobile framework.

A. Main Idea

The implemented multi-agent app allows users to register issues that occur within a company. When a registered issue is critical and the deadline is short (e.g., maximum of two more days), a software agent verifies if another user connected on the same network may solve it. Considering that all employees use the app (iPhone was the device used) and each one informs his/her data (e.g., name and services that perform), a software agent is created for them.

 ISBN: 978-1-61208-590-6

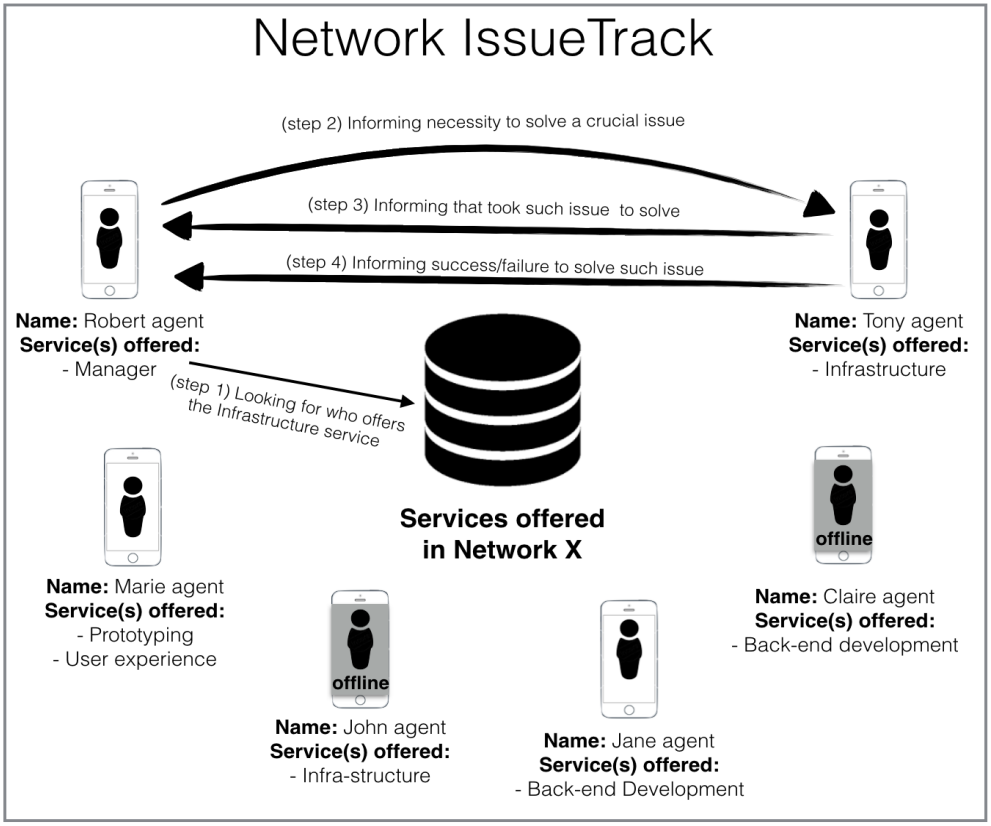

Figure 3. Issue track scenario.

The afore mentioned scenario is illustrated in Figure 3. Notice that six agents are registered. At a given time, two agents are not connected to the company network. Probably, the reason is that these registered employees are not in the company.

As the iMobile framework was extended to create this app, Bonjour API was inherited. Therefore, it is possible to verify which services are registered and which agents offer them. For instance, it considers that a Robert agent (see Figure 3) needs someone to solve a critical issue related to infrastructure with deadline for tomorrow. Hence, it is important to locate a person that may solve it as soon as possible.

In order to identify an infrastructure employee, the Robert agent uses Bonjour to look for such employee (step 1 illustrated in Figure 3). Thereafter, Robert sends a message to the Tony agent, which offers infrastructure services and is connected to the same network (step 2). When Tony receives that message, a notification is created on his smartphone for the employee (user app) confirm that he/she will take it. Upon performing such confirmation, Tony sends a message to Robert (step 3). When Robert receives this message, a notification is presented to Robert's user. Subsequently, when the issue is solved, Tony informs Robert about its resolution (step 4).

When no agent is found (from the step 1), an email is sent to those that could solve the registered issue (considering that they are not in the company). In addition, to allow the messages receiving from agents, at any time, when a person arrives at the company, he/she receives a notification requesting them to open the app. Then, even if the app is not in use, at a given time, but has been opened in the background, the agent will be able to receive messages and perform some executions.

B. *Extending the iMobile Framework*

In Figure 4, a class diagram, with the main classes made to extend the iMobile framework and to create the issue track app, is presented. The yellow classes are the new developed entities, while the others are offered from iMobile. Below, these new classes are explained in detail.

App User (*User* class): When an employee of the company uses the app for the first time, a set of data should be informed to him/her: name, date of birth, email and cellphone. Part of these data is taken to create a software agent that will represent the employee.

Agent app (*IssueAgent* class): This class represents the responsible agent for executing the following activities: (i) publishing which services are offered in its creation, based on the data informed by app user, (ii) requesting the resolution of critical issues to other agents, and (iii) taking or not taking it to solve these issues. Each iPhone has one *IssueAgent* created to represent the app user.

 ISBN: 978-1-61208-590-6

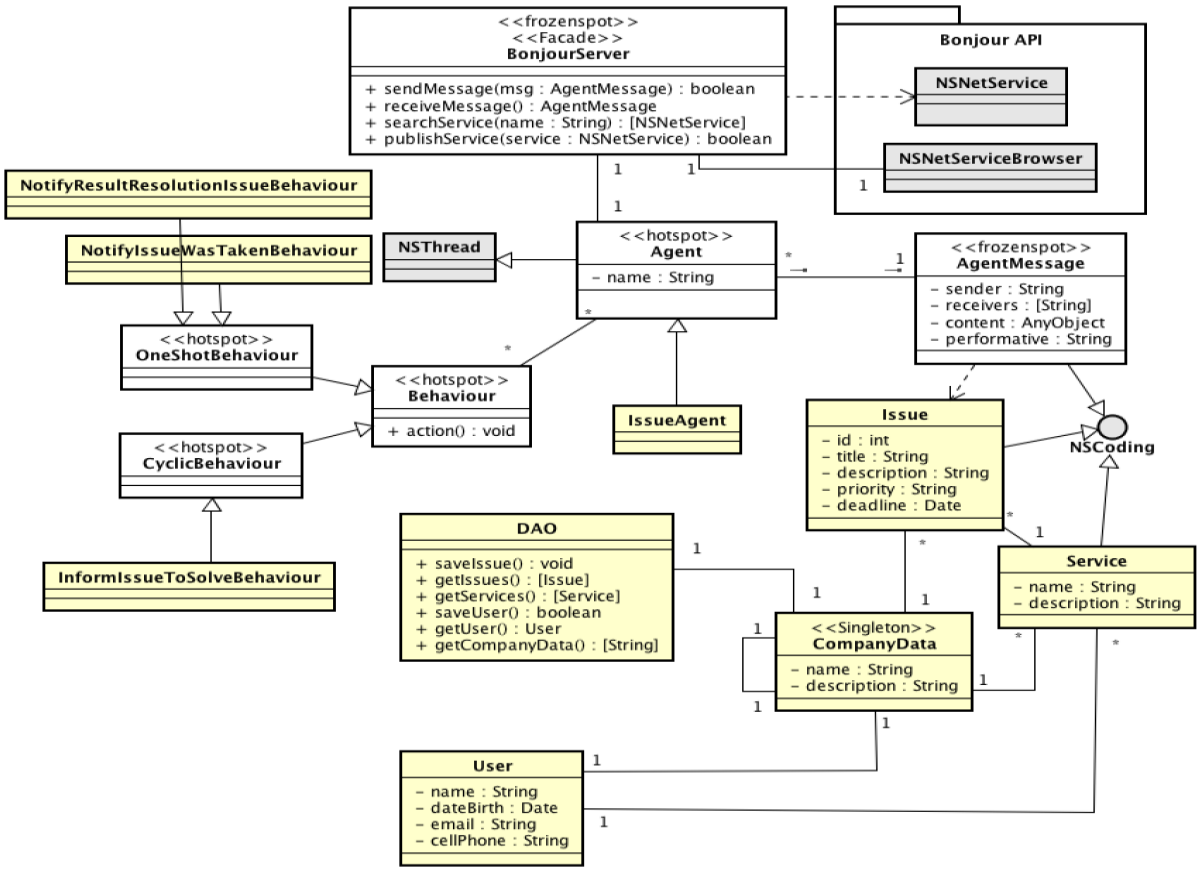

Figure 4. Class diagram of the Issue track scenario.

Registered issue (*Issue* class): It represents a registered issue that needs to be solved. An issue has the following data: an identification represented by an integer number, title, its description, priority (critical, high, medium, low), deadline that such issue needs to be solved and the service type related to the problem.

Service offered in the company (*Service* class): This class represents all the service types performed in the considered company. Some examples of services are the following: infrastructure, back-end development, front-end development, prototyping and user experience. Then, each instance of *Service* has a name and a description giving an overview of its goal.

Management to persist data (*DAO* class): This class is responsible for taking and saving data in a database used by the app. In Figure 4, the main methods offered by class are presented.

Company information (*Company* class): It applies the Singleton pattern [37] and takes into consideration all of the company's data. Thus, it is possible to access the company name and description, besides its issues, app user and registered services.

Behavior which informs that an issue needs to be solved (*InformIssueToSolveBehavior* class): This class represents the behavior executed by an *IssueAgent,* that requests the treatment of some issue by other agents. In order to decide if such a request will be sent, the agent verifies the priority of the registered issue and its deadline. The criteria adopted by the agent to send a request is when an issue has: (i) critical priority, or (ii) high priority with deadline of two more days. Hence, according to the company, critical scenarios were taken into consideration to include an additional way to contact employees that may solve these issues.

Behavior to inform that an agent took an issue to be solved (*NotifyIssueWasTakenBehavior* class): This behavior is executed by agents that received a request to solve a given issue. Such agent informs the requester if it could or could not take that issue. The reasons why an agent may or may not accept to solve a problem are presented below.

Reasons to accept an issue:

- Employee is free to take an issue;
- Employee is working to solve an issue with medium or low priority.

Reasons to not accept an issue:

- Employee is solving an issue with critical or high priority and a short deadline (maximum of two more days);
- Employee is not connected to the company network.

When no agent accepts the requested issue, an email is sent to all employees that offer the necessary service to solve it.

Behavior that informs the result of an issue resolution (*NotifyResultResolutionIssueBehavior* class): Behavior executed by an agent that accepted to solve a given issue. From this behavior, the agent notifies the result of its resolution to the requester: solved or not with additional information reported by the employee.

V. CONCLUSION

This paper presents the iMobile framework created from the Swift language for the iOS platform. With iMobile, developers will be able to speed up the creation of software agents for the iOS. This framework used the Bonjour API in order to help identifying which agents are or are not on the same network and to allow the communication between them. Bonjour is a solution that does not require the creation of containers to enable agents to exchange messages, as do JADE and JADEX. Before proposing iMobile, known frameworks, that help in the development of software agents, were studied to identify how a mobile framework for the iOS platform could be created and offered.

In order to demonstrate the use of the proposed framework, we have used it to help solving issues that occur in companies. Agents are responsible for identifying these issues' priorities and deadlines, in order to contact employees that are in the company, and to quickly solve them. This scenario illustrates the use of iMobile agents exchanging messages from Bonjour and executing a set of defined behaviors.

Nowadays, we have two master students of Informatics at PUC-Rio who used the proposed framework. Moreover, a set of ten people, with previous experience developing multi-agent systems, watched a presentation of the iMobile related to its idea, architecture and examples of case studies. From this, we have decided to organize an interview with such people to receive feedbacks of the framework. One of them was that the framework was easy to be extended to create iOS apps. In addition, they gave a set of information that allowed us to achieve the version presented in this paper, such as (i) offering better names for some created classes and methods, and (ii) continuing to offer a short number of classes to create a multi-agent system to the iOS platform.

Currently, we are in the process of analyzing how we may include other important concepts related to the multi-agent paradigm in the framework, such as: offering ways to develop self-adaptive agents, including the concept of Belief-Desire-Intention (BDI) in the framework, bringing more cognition to the agents and providing ways to test the created agents. These three ideas are known research lines investigated in other works. However, considering the mobile scenario, several issues are open and deserve more attention.

REFERENCES

[1] G. Butler, Object-Oriented Frameworks, Available at: http://users.encs.concordia.ca/~gregb/home/PDF/ecoop-tutorial.pdf, [retrieved: June, 2017].

[2] N. R. Jennings and M. Wooldridge, "A Methodology for Agent-Oriented Analysis and Design", In Proc. Of the third annual conference on Autonomous Agents (AAMAS 1999), Seattle, WA, USA, pp. 69-76, 1999.

[3] M. Wooldridge and N. R. Jennings and D. Kinny "The Gaia Methodology for Agent-Oriented Analysis and Design", Autonomous Agents and Multi-Agent Systems, vol. 3, issue 3, Sep. 2000, pp. 285-312, doi: 10.1023/A:1010071910869.

[4] G. Boella, L. Sauro, and L. van der Torre. "Power and Dependence Relations In Groups of Agents", In Proceedings of the conference on intelligent agent technology, (IAT 2004), Beijing, China, China, Sep. 2004, doi: 10.1109/IAT.2004.1342951.

[5] JADE Leap. Available at: http://jade.tilab.com//, [retrieved: June, 2017].

[6] JADE Framework. Available at: http://jade.tilab.com//, [retrieved: June, 2017].

[7] Android. Available at: www.android.com/, [retrieved: June, 2017].

[8] IOS, Available at: https://www.apple.com/ios/, [retrieved: June, 2017].

[9] Java. Available at: http://www.java.com/pt_BR/, [retrieved: June, 2017].

[10] Mobile Application Lifecycle. Available at: https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/TheAppLifeCycle/TheAppLifeCycle.html, [retrieved: June, 2017].

[11] I. Podnar, M. Hauswirth, and M. Jazayeri. Mobile Push: "Delivering content to mobile users", In Proceedings of International Conference on Distributed Computing Systems Workshops, (ICDCS 2002), IEEE, Nov. 2002, pp. 563-568, doi: 10.1109/ICDCSW.2002.1030826.

[12] G. Eric. What is Java 2 Micro Edition, Available at: http://www.developer.com/ws/j2me/article.php/1378921/What-is-Java-2-Micro-Edition.htm, [retrieved: July, 2017].

[13] Y. Weihong and Y. Chen, "The Development of Jade Agent for the Android Mobile Phones", Proceedings of the 2012 International Conference on Information Technology and Software Engineering, (ICITSE 2012), Springer Press, Nov. 2012, pp. 215-222, doi: 10.1007/978-3-642-34531-9_23.

[14] FIPA. Available at: www.fipa.org/, [retrieved: June, 2017].

[15] JADEX. Available at: https://www.activecomponents.org/, [retrieved: June, 2017].

[16] M. Wooldridge and N. R. Jennings, "Intelligent agents: theory and practice", The Knowledge Engineering Review (KER 1995), Cambridge Press, July 2009, pp. 115–152, doi: 10.1017/S0269888900008122.

 ISBN: 978-1-61208-590-6

[17] JACK. Available at: http://aosgrp.com/products/jack/, [retrieved: June, 2017].

[18] I. Nunes, C. J. P. Lucena, and M. Luck, “BDI4JADE: a BDI layer on top of JADE, Monografias em Ciência da Computação”, PUC-Rio, No. 15/10, Nov. 2010.

[19] R. P. Bonasso, R. J. Firby, E. Gat and et. al, “Experiences with an architecture for intelligent reactive agents”, Journal of Experimental & Theoretical Artificial Intelligence (TAI 1997), Taylor&Francis, Nov. 2010, vol. 9, issue 2-3, pp. 237-256, doi: 10.1080/095281397147103.

[20] Bonjour API. Available at: https://www.apple.com/support/bonjour/, [retrieved: June, 2017].

[21] D. H. Steinberg and S. Cheshire. Zero Configuration Networking: The Definitive Guide, O’Reilly Media, p. 53, 2005.

[22] NSThread. Available at: https://developer.apple.com/reference/foundation/thread, [retrieved: June, 2017].

[23] GCD. Available at: https://developer.apple.com/reference/dispatch, [retrieved: June, 2017].

[23] J. Frost. BSD Sockets: A Quick And Dirty Primer, 1991.

[25] NSCoding. Available at: https://developer.apple.com/reference/foundation/nscoding, [retrieved: June, 2017].

[26] NSObject. Available at: https://developer.apple.com/reference/objectivec/nsobject, [retrieved: June, 2017].

[27] UIApplication. Available at: https://developer.apple.com/reference/uikit/uiapplication, [retrieved: June, 2017].

[28] UIDevice. Available at: https://developer.apple.com/reference/uikit/uidevice, [retrieved: June, 2017].

[29] ACL Message. Available at: http://www.fipa.org/specs/fipa00061/, [retrieved: June, 2017].

[30] A. S. Tanenbaum and V. S. Maarten, Distributed systems: principles and paradigms. New Jersey: Pearson Education. Inc, 2007. p.669

[31] B. A. De Maria, V. T. Silva, C. J. P. Lucena, and R. Choren, "VisualAgent: A Software Development Environment for Multi-Agent Systems", In Proc. of the 19th Brazilian Symposium on Software Engineering (SBES 2005), Tool Track, Uberlândia, MG, Brazil, 2005.

[32] Swift Language. Svailable at: https://swift.org, [retrieved: June, 2017].

[33] PYPL PopularitY of Programming Language. Available at: https://pypl.github.io/PYPL.html, [retrieved: June, 2017].

[34] Top Reasons Businesses Close Down. Available at: http://smallbusiness.chron.com/top-reasons-businesses-close-down-20466.html, [retrieved: June, 2017].

[35] Common Reasons for Closing a Company. Available at: http://www.closeaeuropeancompany.com/common-reasons-for-closing-a-company.html, [retrieved: June, 2017].

[36] GameplayKit. Available at: https://developer.apple.com/library/content/documentation/General/Conceptual/GameplayKit_Guide/, [retrieved: June, 2017].

[37] M. E. Fayad, D. C. Schmidt and R. E. Johnson, Building Application Frameworks: Object-Oriented Foundations of Framework Design (Hardcover),Wiley publisher, first edition ISBN-10: 0471248754, 1999.

[38] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., Design Patterns: Elements of Reusable Object-Oriented Software, 1994.

[39] M. Ivanovic and Z. Budimac, “Software Agents: state-of-the-art and possible applications”, In Proceedigns of the 13th International Conference on Computer Systems and Technologies, (CompSysTech 2012), ACM Digital Library, June 2012, pp. 11-22, doi: 10.1145/2383276.2383279.

[40] M. Żytniewski and A. Sołtysik, A., Sołtysik-Piorunkiewicz and B. Kopka, “Modelling of Software Agents in Knowledge-Based Organisations. Analysis of Proposed Research Tools”, Springer, Sep. 2015, pp. 91-108, 2015.

[41] J. T. C. Tan and T. Inamura, “Extending chatterbot system into multimodal interaction framework with embodied contextual understanding”, In International Conference on Human-Robot Interaction (HRI) (ACM/IEEE 2012), IEEE Press, Mar. 2012, pp. 251-252, doi: 10.1145/2157689.2157780.

[42] DNS Zone, Available at: http://www.dns-sd.org/, [retrieved: July 2017].

[42] NSSecureCoding, Available at: https://developer.apple.com/documentation/foundation/nssecurecoding, [retrieved: July 2017].

[43] Crypto Swift, Available at: http://cocoadocs.org/docsets/CryptoSwift/0.5.2, [retrieved: July 2017].

 ISBN: 978-1-61208-590-6