Received September 7, 2020, accepted September 20, 2020, date of publication September 28, 2020, date of current version October 16, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3027357

SPADE 3: Supporting the New Generation of Multi-Agent Systems

JAVIER PALANCA, ANDRÉS TERRASA, VICENTE JULIAN, AND CARLOS CARRASCOSA
Valencian Research Institute for Artificial Intelligence (VRAIN), Universitat Politècnica de València, 46022 València, Spain

Corresponding author: Javier Palanca (jpalanca@dsic.upv.es)

This work was supported in part by the Spanish Government, under Project RTI2018-095390-B-C31-AR.

ABSTRACT Although intelligent agent-based systems have existed for several years, the progression in terms of real applications or their integration in the industry have not yet reached the expected levels. During the last two decades, many agent platforms have appeared with the aim of simplifying the development of multi-agent systems. Some of these platforms have been designed for general purposes, while others have been oriented towards specific domains. However, the lack of standards and the complexity associated with supporting such systems, among other difficulties, have hampered their generalised use. This article looks in depth at the current situation of existing agent platforms, trying to analyse their current shortcomings and their expected needs in the near future. The goal of the paper is to identify possible lines of work and some of the most crucial aspects to be considered in order to popularize the application of agent technology as a dynamic and flexible solution to current problems. Moreover, the paper presents SPADE 3, a new version of the SPADE middleware, which has been totally redesigned in order to conform to the identified challenges. Finally, a case study is proposed to illustrate how SPADE 3 is able to fulfill these challenges.

INDEX TERMS Multi-agent systems, intelligent agents, middleware.

I. INTRODUCTION

Multi-agent systems (MAS) technology allows for the development of autonomous software entities (intelligent agents) which are naturally designed to communicate with each other. This communication enables the formation of complex interaction spaces, from which higher-level social activities such as cooperation or collaboration may emerge. In order to facilitate the actual development of this type of systems, several frameworks have been proposed in recent years. Such frameworks, normally known as MAS platforms, usually offer certain facilities at the communication layer, the internal agent architecture, the set of development tools, etc. The initial trend where a multitude of MAS platforms were created and coexisted for some time has moved nowadays to a situation in which many of them are either no longer supported or have been adapted to new requirements and/or functionalities.

Current requirements in the area of multi-agent systems not only include that intelligent agents are able to interact with each other in an open context, but also that the supporting infrastructure provides them facilities to effectively build this context. In addition, the recent appearance of new areas such as Internet of Things, Ambient Intelligence, Cyber-Physical Systems, etc., has resulted in the emergence of new agent-based software specific to the needs of systems in such areas. However, most of these requirements are still not covered by the existing agent platforms. This gap has weakened many of the existing proposals, and so it is imperative that the requirements and functionalities of agent infrastructures be revised and reformulated, in order to support the implementation of current, and future, MAS applications.

In this sense, this article deepens into the current state of agent platforms, trying to analyze the existing trends, as well as their possible weaknesses, in order to propose an adequate support to the requirements of the next generation of multi-agent systems. As a result of this analysis, the paper intends to contribute with a set of open issues that may establish the foundations for current and future supporting infrastructures for agent-based solutions.

Following this idea, the paper also includes the proposal of a new version of the SPADE[1] software [1], a middleware for developing and executing multi-agent systems, written in Python. In particular, this article introduces SPADE 3, which has been completely redesigned and rewritten from scratch

The associate editor coordinating the review of this manuscript and approving it for publication was Zhan Bu.

[1]https://pypi.org/project/spade/

by the authors of this article, eliminating the functionalities that are no longer useful for modern multi-agent systems, and focusing on the incorporation of a careful selection of concepts and modern technologies which may cover some of the current open issues identified in the paper.

According to this, the paper presents three main contributions: the first one is a study of the current situation of agent platforms; the second one is an analysis of some relevant open issues for the development of modern and future multi-agent systems; and the third one is the presentation of SPADE 3, a completely redesigned multi-agent systems middleware that tries to cover the needs and shortcomings detected in this analysis. In particular, when compared with other proposals discussed in the paper, SPADE features a fully open, scalable and extensible development and execution environment which makes full use of a standard, well-known communication protocol; transparent integration of humans and agents; a modern and full-featured programming language such as Python, which is nowadays leading the rankings of use in most domains, and specifically in the Artificial Intelligence arena[2]; and a set of development mechanisms which facilitate the implementation of MAS applications.

The rest of the paper is structured as follows: Section II presents a view of the current situation of existing platforms for multi-agent systems development; Section III analyses some relevant open issues related to this development or support; Section IV introduces the SPADE 3 proposal, by comparing its functionalities with previously detected open issues; Section V describes a case of study which has been implemented in SPADE 3, with the aim of illustrating its main features regarding these open issues; finally, Section VI presents the conclusions of the paper.

II. STATE OF THE ART

Over the last few years, multi-agent systems (MAS) have greatly evolved in several facets by following different research lines, but this evolution has not yet achieved their full consolidation. One key aspect in this consolidation is the support given by MAS platforms, which are the pieces of software (or *middleware*) upon which MAS are built and executed. In several ways, the architecture, services, tools, etc., provided by a particular platform conditions the abilities of the MAS which can be developed and run on that platform and, in turn, this affects the types of problems which those MAS can solve.

In the research field of MAS platforms, some of the latest significant developments include, on the one hand, the evolution of some *classic*, well-known, general-purpose platforms. On the other hand, several new platforms have been proposed, many of which are oriented towards developing MAS in specific areas of interest which have become popular lately, such as the Internet of things (IoT), Ambient Intelligence (AmI), Cyber-Physical Systems (CPS) or Agent-Based Simulations (ABS). The reminder of the section will focus on revising these two types of recent results in MAS platforms, with the objective of identifying some important aspects which they have not yet solved or even tackled.

Among the general-purpose platforms, probably the most popular and established one is *JADE*[3] (JAVA Agent DEvelopment Framework) [2], which last version is from 2017. JADE has probably been the most widely used tool to develop multi-agent systems in the last two decades [3]. JADE implements a fully-compliant FIPA[4] platform [4] as a series of interconnected Java Virtual Machines called *containers*. Some of the most relevant late developments related to JADE are WADE and JADEL, which are now described. *WADE*[5] (Workflows and Agents Development Environment) [5] is an extension of JADE which adds the workflow abstraction to the agent concept, by means of a workflow engine. So, the MAS is designed from the viewpoint of a workflow, and the workflow tasks are assigned to the agents. *JADEL* [6] (JADE Language) is an initiative which aims to provide the MAS developer with a new, friendly-syntax language for JADE agent programming. However, as a language for programming agents, it is a limited proposal, and it is so linked to JADE that it does not cover all the functionalities of the platform; as a result, some actions (such as the instantiation of agents, for example) need to be carried out by directly using the facilities of the JADE platform.

Other popular proposals aimed to support MAS in general are platforms which have JavaBeans as their main underlying technology. This is for example the case of IBM's ABLE or MadKit. *ABLE* (Agent Building and Learning Environment) [7] was designed as a lightweight Java-based agent framework, and for some time it was widely used in several different domains, such as automotive diagnostics, system health monitoring, agent-based modeling and simulation, etc., but nowadays it seems not to be available anymore. *MadKit*[6] (Multi-agent development Kit) [8] on the other hand, is focused on providing support for agent organizations, but without imposing a predefined agent model. The *platform* is formed by a *micro-kernel* which supports the agents life-cycle managing, agent messaging, group managing, etc., and a set of system agents in charge of the platform services.

The last two items in this list of general-purpose platforms are JANUS and JaCaMo, and they are relevant to this review because of their recent advances and proposals. *JANUS*[7] [9] is a Java-based platform that includes not only organizational concepts but also holons (in the idea of a holon as a sub-structure composed itself of holons). Its main and most novel feature is its ability to natively manage the concepts of recursive agents and holons. The *JaCaMo+* [10] platform is founded on supporting the concept of agent organization. It integrates the Jason language [11] (a BDI agent language

[2]https://spectrum.ieee.org/computing/software/the-top-programming-languages-2019

[3]http://jade.tilab.com
[4]http://www.fipa.org
[5]https://jade.tilab.com/wadeproject/
[6]http://www.madkit.net
[7]http://www.janusproject.io

based on the AgentSpeak specification), CArtAgO [12] as a platform for implementing artifacts to define the environment of the agents, and MOISE [13] as the organizational model to structure the agents in the multi-agent system.

As stated above, there have also been recent developments in MAS platforms which have been designed to support applications in specific domains. In this group, there are platforms oriented to domains such as Internet of Things (IoT), Ambient Intelligence (AmI), Cyber-Physical Systems (CPS), Human-Agent Interaction (HAI) or Agent-Based Simulations (ABS). Some of these specific-domain platforms are now discussed.

In the AmI area, some relevant work can be found in [14], [15], which propose multi-agent solutions that facilitate the development of context-aware and dynamic applications in smart environments that take into consideration device heterogeneity. Since such proposals are ad-hoc developments to each particular application, the supporting platforms are hardly applicable to other domains or applications.

The field of IoT presents several MAS platform proposals. For example, [16] addresses the problem of applying Edge Computing (EC) to an IoT system, by proposing a multiagent EC platform as a flexible and scalable solution, where each platform component is controlled by a set of software agents. In another proposal, [17] introduces an agent platform for IoT systems oriented to tackling the limitations of current agent platforms in this domain area. Such limitations include, among others, how to support the heterogeneity of IoT environments, the lack of explicit support for the dissemination of data to a specific group of IoT nodes, or the fact of each solution being implemented and deployed for a specific agent platform with a particular communication protocol. The third platform in this list is the one presented in [18]. Here, the authors propose an agent platform to facilitate grid-interactive building operation with IoT devices. This platform is called BEMOSS (Building Energy Management Open Source Software) and it has been developed to improve sensing and control of equipment in small and medium-sized commercial buildings. BEMOSS aims to optimize electricity usage in order to reduce energy consumption and to facilitate the implementation of demand response (DR) programs in commercial buildings.

The work presented in [19] provides a MAS platform for Cyber-Physical Systems with specific features that allow cyber-agents to access and control real physical devices with ease. This platform supports rapid development and deployment of agents with mobility, as well as multi-threading programming, and it can be used in conjunction with some embedded devices. Another particularity of this platform is that it has been written in Prolog, whilst the majority of platforms discussed in this section have been coded in Java.

Finally, there are proposals focused on other areas. In Human-Agent Interaction, [20] presents an agent platform oriented to designing human-aware agents for negotiation processes. In Agent-Based Simulations, there are also some examples, such as *GAMA*[8] [21], a platform for agent-based simulations with special attention to the spatial dimension; *Anylogic*,[9] a well-known simulation tool that provides a combination of different modeling techniques including agent-based techniques; and *MATSim*,[10] a platform for agent-based simulations focused on large-scale transport scenarios.

III. OPEN ISSUES IN CURRENT AGENT PLATFORMS

The review presented in the previous section confirms that research in multi-agent platforms has been active over the last years, either on general-purpose platforms, or on specialized platforms oriented towards specific domain areas, where the provided support is normally tailored to the intended domain or particular application. However, despite all these advances, there are still unsolved or open issues in the area of multi-agent platforms, and this section tries to identify the most relevant ones.

The analysis of the solutions presented in the previous section reveals that, in general, multi-agent platforms have been traditionally conceived as closed environments designed to support the development and execution of MAS in a particular way. This has been clearly the case for the concrete, ad hoc platforms implemented to support applications in specific domains, but also for the more generic support provided by general-purpose platforms. In this latter case, platforms have normally been built on top of some general concepts such as the agent model and certain key services as the Agent Management Service, the Directory Facilitator, etc. Depending on the particular platform, MAS designers may have available a different set of tools, but the development and execution middleware is normally conceived as a fixed and, in many cases, proprietary framework.

This kind of proprietary support is not adequate for the next generation of MAS, which are likely to be open and massive, to require a great variety of very different services, and to incorporate agents developed by several developing teams under different approaches or requirements and executed from separate (even distant) locations. This flexible, massive, and *social* development of MAS requires a different kind of supporting middleware (platform), centered on some relevant aspects which are not considered by the solutions described in the previous section (at least, none of these solutions takes into account all the aspects). In the authors' opinion, there are four key issues among these aspects, which are now discussed.

In the first place, one of the foremost objectives of an agent platform is to offer the MAS with a simple and effective communication channel, which ideally should be as well-known and standardized as possible. Many of the existing proposals make use of closed or local communication protocols that hinder their interoperability. Conversely, the use of a

[8]https://gama-platform.github.io
[9]https://www.anylogic.com
[10]https://matsim.org

standard, sound and widespread communication protocol would effectively allow the multi-agent system to communicate not only with other MAS, but also with other types of computing elements (devices, software components, etc.) or even with humans. In this way, the use of an instant messaging protocol as the basis of communication in an agent platform would be a very good choice. Instant messaging protocols are nowadays used by millions of users on a daily basis, and they support almost any type of interaction between humans or between humans and computers.

The second crucial aspect when considering open and massive MAS is the elastic and scalable allocation of communication resources, in the same way that modern cloud computing systems are characterized by their ability to adjust the provided resources in order to dynamically meet the varying demands of users. Multi-agent platforms should offer elasticity in the communication process and in the resources related to that process, so that agents cannot be affected by an unexpected, significant growth in the number of messages or in the amount of entities interacting with the system at any particular moment. The platform should be able to allocate the required resources to cope with the increased workload without jeopardizing the integrity of the system, and in a way which is completely transparent to the application design and to the actual agents which may be running at that particular moment.

The third important open issue which is becoming more relevant nowadays is related to one of the specific domain areas mentioned in the previous section; in particular, the AmI area. The integration of humans and computation elements in the same system is going to be one key challenge in the next generation of intelligent systems, and specifically, in MAS. Multi-agent systems should allow for the development of applications where agents and humans can jointly provide services to other humans or agents, in an environment of full integration. This kind of *Human-Agent Societies* [22] will need to be supported at the communication and development levels by future agent platforms. Additionally, the ability to transparently communicate humans with agents will be key in the development of *fully open systems* where entities (humans, agents or third-party elements) can dynamically enter or exit the system in a way which is totally transparent for the system developer. This open system feature has traditionally been difficult to implement for real problems, but it will definitely be required in the near future. Again, the availability of an appropriate communication protocol and infrastructure may be decisive to tackle this feature.

The fourth and final issue is related to the interoperability of the system with IoT devices, which are now one of the most growing device markets in the world, and which are expected to be used in almost every human activity in the near future. In this context, interoperability means the ability of the system to interact with different types of low-powered, non-standard devices, but also the ability of agents to connect to the system independently of the device where they are running while preserving their identities. This, in turn, relies on the capacity of the system to support agents which may run not only on traditional computers but also on such devices, a feature which may be complicated depending on the size and complexity of the platform's middleware. An additional characteristic related to the execution of agents in different devices would be the ability of agents to migrate their execution from one device to another without restarting the system. This characteristic would be very useful not only in IoT, but also for many types of MAS, but it is difficult to achieve in the general case.

To sum up, the next generation of multi-agent platforms will probably need to provide a standard and effective communication protocol, elasticity in communication, full and transparent integration of humans and agents, support for open systems, and the ability of agents to connect to the system independently of their running device. By incorporating valid solutions to such open issues, platforms not only will provide an appropriate environment to build the type of MAS applications and domain areas mentioned above, but they will also be able to provide improved support to solve classic problems in the MAS area. Such problems include team formation [23], [24], task/resource allocation [25], [26], crowds dynamics [27], [28], MAS planning [29], or conflict resolution [30], [31], to mention a few of the most relevant ones at the moment.

IV. SPADE 3

SPADE 3 is a middleware for multi-agent systems that represents an evolution of the traditional multi-agent system platforms by means of incorporating a careful selection of concepts and modern technologies in the areas of distributed systems, instant messaging, asynchronous systems and open systems. SPADE 3 has been completely redesigned and its code rewritten from scratch, by focusing in the new functionalities and open issues pointed out in the previous section. For the sake of simplicity, the rest of this section will refer to this version as SPADE. The principal founding concepts and technologies of SPADE are now briefly discussed.

One of the most relevant concepts of any MAS middleware is its agent model. In the case of SPADE, its agent model is similar of those used in other platforms (such as Jade). The model is internally based on some simple abstractions and mechanisms, now described. The first one is the connection mechanism, by which each agent registers in SPADE by using a unique identifier (which format is "username@server") and a password. After registering, the agents may create one or several behaviors, which are independent tasks that execute the agent's actions. Behaviors can be of several types, with each type producing a particular execution pattern designed to support a typical execution requirement of agents in a multi-agent system. In particular SPADE supports five behavior types: Cyclic, One-Shot, Periodic, Time-Out and Finite State Machine. The third main mechanism is the message dispatcher which SPADE associates with each registered agent. This component acts as a mailman, redirecting any incoming message for the agent to the particular behavior(s) that may be expecting the message, and relaying the outgoing messages from any behavior to the SPADE's communication system.

Among the technologies, SPADE relies on the selection of a particular communication protocol as a paramount component of the multi-agent system, since it can bring very valuable features to intelligent, autonomous, social entities such as agents. In particular, SPADE incorporates XMPP [32] (eXtensible Messaging and Presence Protocol), which is an open protocol for instant messaging and presence notification. XMPP is a protocol based on XML that allows *entities* to exchange messages (with these entities being humans, agents, artifacts, etc.) and which also provides a presence notification mechanism by which any entity may have a list of other entities as *contacts*, and be notified when any of such contacts changes its state (e.g. when a contact is connected or disconnected, when it is busy, etc.). Furthermore, XMPP is defined as an extensible protocol, meaning that many of its features are proposed as extensions (called XEPs); it currently offers plenty of extensions for different purposes, and it is open to proposals from the community in order to make the protocol more flexible and useful. Because of this, XMPP is considered as the universal standard protocol for instant messaging by entities such as the IETF or the W3C, and it has a widespread use in the industry. Whatsapp, Google Talk, Facebook Messenger or Apple's iMessage are some examples of applications that use XMPP, or some variant of this protocol.

As a software project, SPADE has been publicly available for more than a decade. Recently, its complete redesign and re-implementation in order to create SPADE 3 has regained a growing interest from the community. According to the statistics available in the *PyPI* repository site,[11] SPADE 3 has been downloaded 369 times in June 2020 from a total of 1,635 times (from at least 9 different countries) in the first half of 2020, as it can be observed in Table 1. This community of SPADE users is using it to develop different applications, such as *SimFleet*[12] [33], [34] or *pygomas*,[13] or even to extend its functionalities as, for example, the *spade_bdi*[14] module which incorporates BDI-based behaviors to SPADE agents.

The next subsections present the key concepts on which SPADE is founded, many of them based on the XMPP standard. In each case, the section will stress how SPADE makes use of the related concept in order to provide an appropriate support to multi-agent systems.

A. AN OPEN, DECENTRALIZED, FEDERATED PROTOCOL

XMPP provides a federated, open server architecture (depicted in Figure 1), by which any XMPP server can communicate with any other running XMPP server in the Internet (in the same way than SMTP mail servers). By connecting to such federation of servers, SPADE enables any agent to communicate with any other agent, artifact or human in the world, pushing the concept of an open multi-agent system to a new level. In particular, the support that SPADE requires from a XMPP server may be broadly configured in three different ways, depending on the application requirements. First, a multi-agent system application running in SPADE may be configured to deploy its own public XMPP server (there are several open-source implementations of XMPP server software). This configuration makes the application self-sufficient while allowing its agents to communicate with other entities (agents or otherwise) connected to any other XMPP server over the Internet. A second, simpler configuration may be using any of the existing public XMPP servers which are freely available in the Internet. In this case, SPADE agents would directly register in this public server and then run naturally without any further infrastructure. Finally, a third possible configuration is to deploy a private XMPP server, without server-to-server connections, along with the SPADE application. In this case, the application would run privately, with no possible connections from outside entities.

Using this type of architecture to support multi-agent systems renders several advantages. Probably the most important

TABLE 1. Overview of SPADE's success in terms of downloads. Downloads are shown by country and total at June 2020 and the first half of 2020.

June 2020		1st half 2020	
Country	Downloads	Country	Downloads
US	169	ES	663
ES	92	US	385
CH	20	None	131
None	18	CH	101
HR	18	CN	84
RO	13	PL	73
PL	12	IN	69
IN	11	HR	58
AU	9	FR	43
RU	7	IT	28
Total	369	Total	1,635

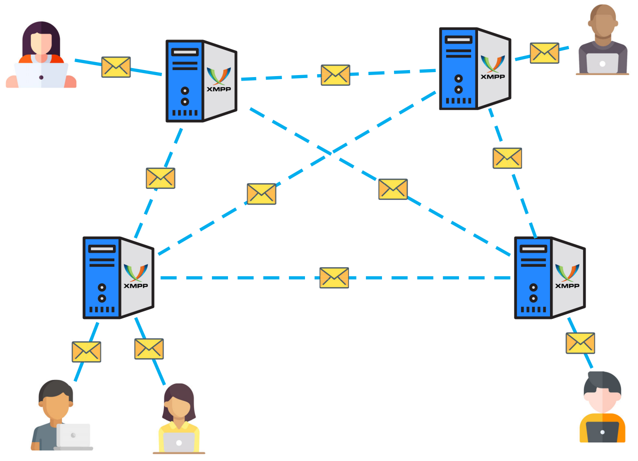


FIGURE 1. A representation of how federated XMPP servers interconnect with each other and with clients, by using server-to-server (dotted lines) and client-to-server (solid lines) mechanisms. In a federation of XMPP servers, messages are routed from any client to its destination client regardless of which servers such clients are connected to.

[11]http://pypi.stats
[12]https://pypi.org/project/simfleet/
[13]https://pypi.org/project/pygomas/
[14]https://pypi.org/project/spade-bdi/

one is flexibility: with the same architecture, SPADE can support from very large multi-agent systems with several interconnected servers able to distribute the messaging load, to much smaller, ad-hoc multi-agent systems where a single, tailored server can be deployed to provide the minimum required functionality. In addition, while the architecture is by nature distributed, hence providing the related advantages of load distribution, fault tolerance, etc., it can also provide some typical benefits from centralized systems, such as presence notification, persisting storage on the server side, strong authentication mechanisms, etc. This flexibility of supporting large or small, centralized or decentralized, open or private systems makes SPADE able to support multi-agent systems that are better adapted to the open issues commented in the previous section. Among other considerations, this architecture allows the multi-agent system to become *elastic*, since it can increase or decrease its size by adding new servers or removing them on demand, while running.

Finally, an additional advantage derived from this decentralized architecture is that SPADE provides agents with the ability of being independent from their physical location, that is, from the address of the computer where they are running. In many multi-agent platforms, the physical (IP) address of the computer where an agent is running must be known by the rest of agents in order to successfully deliver the messages they send to that agent. This is typically solved by including the address of the computer where the agent is running in the agent's name or identifier. But then, if the same agent wants to execute in a different computer, it needs a new identifier, which must also be broadcast to all its possible partners in order to be accessible again. This is tackled in SPADE in a much more convenient way, since agents are identified by means of the XMPP server where they are registered, not by the computer where they are running. So, as long as the XMPP server does not change its address, agents can migrate from running in one computer to another one, in a totally transparent way. This is similar, for example, to email addresses, which enable people to receiving messages regardless of the computer where they read their email. This location independence is a very interesting feature in big scale, open multi-agent systems, where it is common that each time a multi-agent application is executed, its agents may run in a different computer.

B. PRESENCE NOTIFICATION MECHANISM

Another important feature of XMPP which SPADE uses in its advantage is the presence notification mechanism provided by the protocol. By means of this feature, SPADE offers agents with the ability of managing their own presence status, including an automatic notification to their respective contacts when such status is changed. The concept of presence notification is straightforward and widely used nowadays in instant messaging applications, where users can check in real time whether their (human) contacts are online or not. In SPADE, this simple but powerful mechanism of XMPP is not limited to informing agents' contacts of their current connection status (online or offline), but it can be easily configured for an agent to broadcast any type of particular, relevant information to its contacts. In addition, SPADE uses this mechanism for other purposes, such as reporting changes in the internal status of an agent, which is useful for example when the logic of the agent is internally built a finite-state machine; or synchronizing the execution of agents by creating synchronization barriers. This latter is illustrated by an example, depicted in Figure 2, which shows how to use this mechanism in order to coordinate a group of agents which need to work in a common task. The top part of the picture shows the initial situation, where an agent called *Smith0* is waiting until a group of other agents which are its contacts (*Smith1* to *Smith4*) are ready to start working in the common task. Once *Smith0* checks in its contact list that all such contacts are ready, it opens the barrier by changing its status to *start!*, which in turn confirm them to start working. Then, each of the contact agents also changes its status to *working* (until it completes the task), allowing *Smith0* to check who is still working and who has finished the task (this is depicted in the bottom side of the picture).

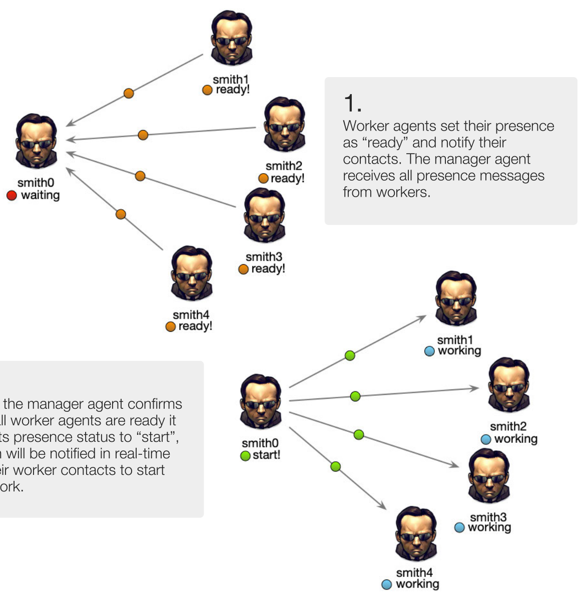

smith1 ready!
smith0 waiting
smith2 ready!
smith3 ready!
smith4 ready!
1.
Worker agents set their presence as "ready" and notify their contacts. The manager agent receives all presence messages from workers.
2.
Once the manager agent confirms that all worker agents are ready it sets its presence status to "start", which will be notified in real-time to their worker contacts to start the work.
smith0 start!
smith1 working
smith2 working
smith3 working
smith4 working

FIGURE 2. An example of how the XMPP presence notification mechanism can be used to coordinate the actions of a set of agents, by implementing a so-called synchronization barrier. The figure shows the two relevant steps in this particular scenario where the manager agent (Smith0) coordinates the execution of the other four agents (Smith1 to Smith4) to start working on a common task.

Related to this presence notification mechanism, each SPADE agent is provided with a subscription process by which it can manage its own contact list. Agents can request to be subscribed to the list of any particular agent, and such requests can be accepted or refused at the discretion of that agent. However, for simpler scenarios where this acceptance procedure is not required, SPADE also offers an automatic acceptance mechanism of contact requests, which frees agents from this task.

C. A MODERN AND COMMUNICATION-ORIENTED LANGUAGE

Providing a powerful and easy-to-use language, in accordance with the most recent technologies, is an important factor in the development of any type of software, and particularly, of multi-agent systems. According to this, SPADE has been developed in Python, and thus provides its natural API for implementing multi-agent systems in this language, which is one of the most widely used programming languages for Artificial Intelligence (AI) applications today [35]. As a result, developers of SPADE applications are provided with a programming language with a steep learning curve, supported by a large and active community which provides consistent help, and with many publicly available third-party libraries and *add-ons*, especially in the field of AI.

In addition, SPADE proposes the asynchronous programming paradigm to internally implement the code of the agents. This programming model is especially effective for applications with coexisting running entities which execute input/output operations often, and it is well defined in the Python ecosystem by means of a library known as *AsyncIO* (from Asynchronous I/O). In particular, applications implemented under this paradigm can optimize the waiting for input/output, improve resource management and leave more processor capacity for the computation tasks. This is especially convenient for multi-agent system applications, where the reasoning process of agents is normally driven by the messages they send or receive, and hence they naturally distribute their time between computation and communication tasks. As a result, SPADE applications can be internally designed to make an efficient use of the available computation resources (compared with other traditional multi-agent system platforms), hence allowing for running larger systems in the same hardware.

D. A SECURE ENVIRONMENT

Security is another important feature which is increasingly being expected for multi-agent system platforms nowadays. This is especially relevant for open multi-agent systems which need to run in the real world, where securing the application and its communications is essential. SPADE provides security at different levels, by making use of particular mechanisms available in the XMPP protocol. For example, XMPP provides certificates and encryption in client-to-server and in server-to-server communications, which SPADE uses for securing communications between entities (agents, humans, etc.). XMPP supports well-proven protocols as TLS [36] to encrypt communications and to sign messages in order to ensure that they are sent and received by reliable endpoints. Some of these supported standards are OMEMO [37] (XEP-0384) and OpenPGP [38] (XEP-0373 and XEP-0374). For this reason, SPADE enforces agents to initially log into a XMPP server by using a private identifier and a password. However, as most of the features of SPADE, this default configuration can be changed. If the SPADE application deploys its own XMPP server, the server can be configured to admit anonymous logins which do not require credentials, and to deactivate encryption if that level of security is not required.

E. AN EXTENSIBLE PROTOCOL

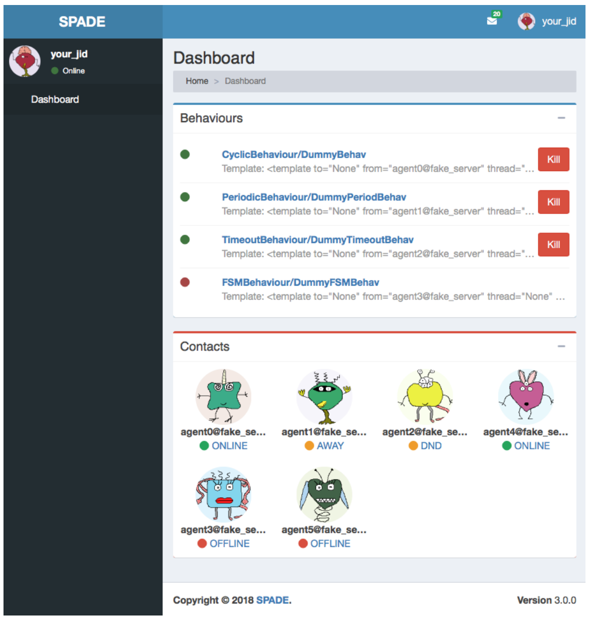
The growth, diversification and complexity that AI applications are experiencing, and are expected to do in the future, makes completely impossible to predict which features will be required for next generations of multi-agent systems. In this context, the best possible support for these systems will be one which can be extended as required without having to completely reconsider their design or implementation. SPADE can provide such feature, based on the natural extensibility of the XMPP protocol. As explained above, XMPP was defined to be extensible, in the sense that there is a standard procedure by which new functionality can be added to the protocol as *extensions* which are known as XEPs (XMPP Extension Proposal). There is a procedure by which the community can propose new XEPs to be officially added to the standard, after revision by the XMPP Standards Foundation.[15] In fact, XMPP already incorporates numerous extensions which are well-suited for multi-agent systems, such as multi-user chats, file transfer, user activity, HTTP gateways, etc. But, since this is an open protocol, developers can also implement new extensions in order to use them privately in their applications.

In many cases, new extensions only require the modification of the XML being sent (normally, by defining a new *namespace*). If, for example, a SPADE application required the FIPA [39] standard communication model in order to be able to interact with other multi-agent system platforms, this could be solved by adding a new message tag using the FIPA standard to XMPP. In other cases, new components need to be added to the XMPP server, but this is also well supported by the protocol, since there is an standardized procedure to do so, and XMPP includes a discovery service by which clients can ask which extensions are available on a particular server. All this, in turn, gives SPADE high adaptation capabilities for future demands of support by multi-agent systems.

F. INCLUDING HUMANS IN THE LOOP

The XMPP communication protocol, employed by SPADE as explained above, also gives support to many instant messaging applications for people-to-people communication. This puts SPADE agents and humans together in the same communication environment in a completely natural way, hence enabling a seamless integration of both types of communication entities. After connecting to a public (or private) XMPP server, SPADE agents can start communicating with other agents, as well as with humans or even many third-party applications (such as Telegram, SMS, email, etc.), which are

[15]The full list of extensions (including the ones currently proposed, accepted, deprecated, etc.) can be consulted in the URL https://xmpp.org/extensions/.



CyclicBehaviour/DummyBehav
Template: <template to="None" from="agent0@fake_server" thread="...
Kill

PeriodicBehaviour/DummyPeriodBehav
Template: <template to="None" from="agent1@fake_server" thread="...
Kill

TimeoutBehaviour/DummyTimeoutBehav
Template: <template to="None" from="agent2@fake_server" thread="...
Kill

FSMBehaviour/DummyFSMBehav
Template: <template to="None" from="agent3@fake_server" thread="None" ...

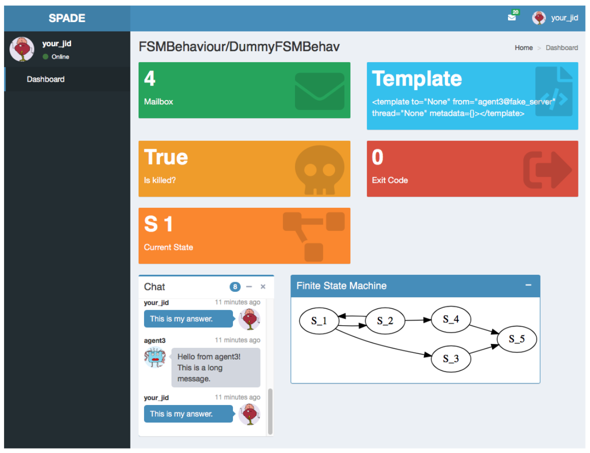
agent0@fake_se... ONLINE
agent1@fake_se... AWAY
agent2@fake_se... DND
agent4@fake_se... ONLINE
agent3@fake_se... OFFLINE
agent5@fake_se... OFFLINE

FIGURE 3. Example of a SPADE Web Interface Dashboard of an agent. It shows the list of behaviors of the agent and its list of contacts, along with their presence statuses. This dashboard can be used at any time to inspect the status of the agent.

accessible from XMPP servers by means of the so-called bridges or gateways.

Putting agents and humans in the same loop greatly simplifies the design of interfaces (e.g., interacting with SPADE agents can be as simple as chatting with them, in the same way people chat with their friends) as well as the creation of applications where humans and agents cooperate, as chat bots for example. Chat bots are a currently widespread technology that requires human interaction but also some degree of reasoning, or, at least, a reactive behavior of some kind (they are commonly used, for example, as shop assistants in e-commerce websites, or as help-desk bots). The SPADE architecture directly supports this type of applications, since the underlying communication support is not specific to agents, but open to any entity which can participate in the XMPP environment. However, it is worth noting that SPADE also supports multi-agent system applications which are as restricted to certain entities as required, by deploying a tailored, on-premise private server.

Another feature of SPADE which favors the human-agent interaction is a graphical interface which SPADE produces by default for each agent, if needed. This interface is available via web under the `/spade` path. By means of this interface, the agent may visualize any relevant internal data which may be useful to know by a human counterpart in an application; and it also allows the human to interact with the agent, if necessary. Figure 3 shows an example of this interface, available by default for each agent in SPADE. This interface displays the general view of an agent's internals (a typical *dashboard*) including the agent's behaviors and its current contacts with their respective statuses.



4
Mailbox

Template
<template to="None" from="agent3@fake_server" thread="None" metadata={}></template>

True
Is killed?

0
Exit Code

S 1
Current State

your_jid 11 minutes ago
This is my answer.

agent3 11 minutes ago
Hello from agent3! This is a long message.

your_jid 11 minutes ago
This is my answer.

FIGURE 4. Example of a behavior view in the SPADE Web Interface. This is a more detailed view of each property of an agent's behaviour. It shows properties such as the mailbox size, the template that filters messages to the behaviour (if set), whether the behaviour has finished or not, and properties which are particular of the type of behaviour (in this case, the behavior's state machine and the current state). It also shows the list of exchanged messages as a chat interface.

Figure 4 shows a more detailed view of a running agent, in this case, of one of its behaviors. In this view (web page), some important data of the execution status of the behavior is shown, such as its mailbox or the internal details of its execution (in this case, this is a finite-state-machine behavior, and the page shows both its structure and its current state). The view also allows the human inspecting the agent to check all the messages that have been sent to, and received by, the behavior, by means of the chat box.

In addition to this basic, standard interface by which SPADE provides inspection of (and interaction with) agents, it is also possible for each agent to incorporate new, alternative web interfaces. Such interfaces can be used by the agent to present visualization or interaction displays adapted to the needs of the application (e.g., input forms in a mobile interface, graphs of the results the agent is calculating, etc.). In order to do this, SPADE offers a straightforward programming scheme based on the Model View Controller (MVC) design paradigm, where the model is the agent's knowledge, the view is a HTML template (following the Jinja[16] syntax), and the controller is an asynchronous function implemented within the agent's code. Thus, the agent developer may easily create interfaces, adapted to the particular type of human-agent interaction required by the application, which access the internal information of agents by means of the controllers and display this information by means of the templates.

V. CASE STUDY: VOLUNTARY DISTRIBUTION OF ESSENTIAL GOODS

In the first half of 2020, many countries have faced confinement situations produced by the COVID-19 disease. In situations like these, the efficient and effective distribution of essential goods and medicines for the elderly or vulnerable

[16]https://palletsprojects.com/p/jinja/

people has become one of the crucial aspects in the disease management.

From a logistics point of view, this issue can be seen as a distribution problem similar to the so-called Last Mile Delivery (LMD) problem. The LMD problem can be defined as the last movement of goods from a transportation center to the final destination of the delivery. The main objective of last-mile logistics is to make the delivery of the goods as fast and efficient as possible. In this context, this section presents an application of the LMD problem to tackle the distribution of necessity goods to vulnerable people by a group of volunteers, implemented in SPADE. The aim of this section is to show how SPADE can provide an appropriate support for a large-scale, distributed, intelligent application designed for an open environment.

In this proposed solution, the goods to be transported are necessity goods (food, medicines, etc.), the transporters are volunteer workers who offer themselves to distribute the goods, and the customers are people which cannot leave their homes because of being vulnerable to the virus. In this situation, the volunteers (transporters), even though they may not know each other, need to collectively organize themselves in order to efficiently manage the distribution process. In cases like this, an intelligent management and planning solution of the distribution problem can significantly help volunteers to analyze and improve their delivery routes as well as to optimize their personal strategies. On the other hand, the customers may be anxious to receive their orders due to the situation, and they may become frustrated as they wait for their goods. In this case, the use of an intelligent last-mile tracking system would allow them to see the current location of their respective packages in real time, and even be notified of their estimated arrival times, which would contribute to relieve their anxiety. Overall, the use of multi-agent system technology in order to develop such a distributed, intelligent planning and tracking application of last mile delivery is more than adequate.

In [40], a new approach to the LMD problem which considers a crowdsourcing solution was presented. This solution is based on an open fleet of *temporary* transporters who do not take a specific route to deliver each package from the customer's pickup point to the final destination, but on the contrary, they make use of their usual routes in order to carry a package, either to its final destination or to a point where another transporter can pick it up and continue with the delivery process. This crowdsourcing approach can be also applied to this case of distributing essential goods for vulnerable customers by means of a virtual fleet consisting of a group of volunteers.

In particular, the proposed approach is presented as a mobile service (or app) to the two types of users: volunteers which offer themselves to occasionally deliver some kind of essential goods, and vulnerable customers which need these goods but are confined in their homes, typically in an urban area. Once the users register in the application by using their mobile devices, the system will be able to locate them in real time, and to share their current location to other users when necessary. From that moment on, whenever a customer issues a delivery request, a dynamic network analysis (explained in [40]) uses the fleet of geo-localized volunteers in order to calculate a particular path for delivering the goods to the customer's location. It is important to note that, in order to optimize each delivery, the system builds the complete delivery path as a chain of collaborative deliverers (the volunteers) in which each volunteer carries the package over some part of the path (a sub-path) and then passes it to the next volunteer. Figure 5 presents a general view of the proposed approach.

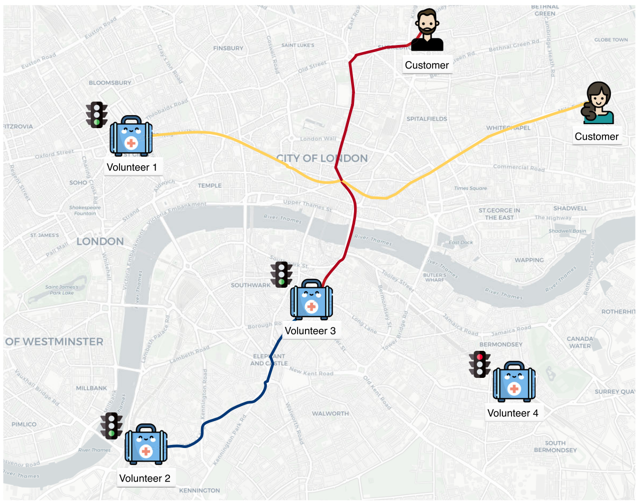


FIGURE 5. General view of the Last Mile Delivery case study where a set of volunteers (represented by blue suitcases) transport goods to vulnerable customers (people icons) by following their respective habitual routes through the city. In particular, the figure shows a volunteer (Volunteer 1) that makes a direct delivery to a customer (yellow path) and also a group of two volunteers who coordinate to serve another delivery request: Volunteer 2 first takes the goods to Volunteer 3 (blue path), who then delivers them to the customer (red path). The figure also depicts the current availability status of each volunteer as a green or red light (e.g., Volunteer 4 has decided not to be available for transporting goods at this moment, as indicated by its red light).

In order to develop and test a prototype of this system, a fleet simulator on an urban area called SimFleet, which was already implemented on SPADE, has been used. This tool simulates the environment (typically, a city) where the users may be geo-located and the delivery paths are depicted. It is worth noting that, although the current prototype of the system described below does use this simulation framework as a front-end, the system has been designed in order to be run in the real world once this prototype version is fully tested.

The following subsections explain the internal design of this prototype and some preliminary empirical results which have been obtained in several executions of the system over the simulation environment.

A. SYSTEM ARCHITECTURE

In essence, the system keeps track of a pool of pending deliveries, which have been issued by the vulnerable customers and have to be assigned to the volunteer transporters which may be currently available, in the most efficient way

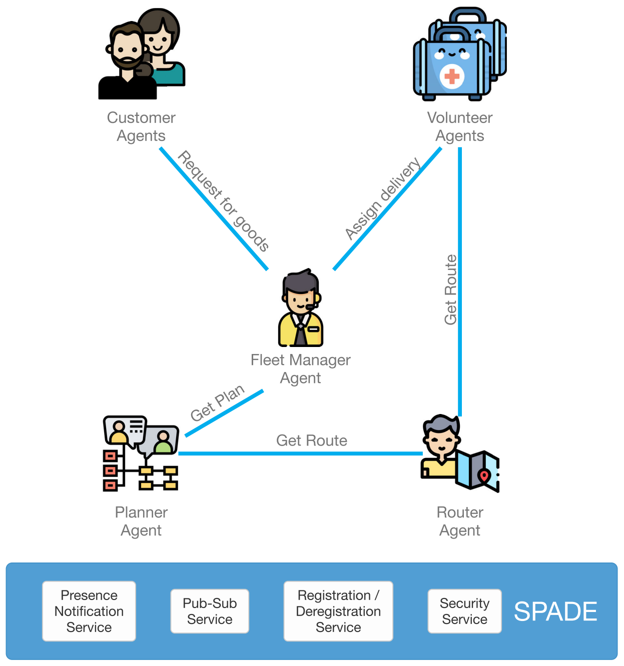


FIGURE 6. Internal architecture of the proposed system. This figure shows how each type of agent in the case study interacts with each other to solve the delivery problem. At the bottom, it also shows the most relevant SPADE services that the agents use to do their job.

possible. In the implementation of this system on SPADE, the following main types of agents have been defined (they are also depicted in Figure 6):

- Volunteer agent: it represents a potential volunteer in the system. Its interface with the human volunteer is a web app by which it is possible to enter or exit the system at any time, change the volunteer's availability through the presence notification mechanism, and accept deliveries by means of a subscription protocol.
- Customer agent: it represents a vulnerable person who needs some essential goods to be delivered. Its interface is also a web app by which the customer may issue the request of goods as well as to monitor the ongoing request in real time.
- Planner agent: this agent is in charge of computing optimized delivery paths, given the pending deliveries and the volunteers' habitual routes, and then proposing such paths to the volunteers. In order to do so, this agent uses the crowdsourcing algorithm proposed in [40]. As an example, in Figure 5, the Planner has produced two delivery paths: one path for Volunteer 1 to directly deliver the goods to its customer, and another one with two sub-paths, where Volunteer 2 first transports the goods up to a point and then passes them to Volunteer 3, who finally delivers them to the customer.
- Router agent: its main goal is the calculation and reconfiguration of routes within the city, taking into account different transport forms (such as bicycle, car, bus, etc.).
- Fleet Manager agents: these agents allow for the federalization of the system. Fleet Manager agents may be used to create and manage different groups of transporters (volunteers) according to different criteria. For example, it could be used to subdivide the city in different areas, each managed by a fleet agent, hence turning the global delivery process into a federated system.

In the proposed solution, these agent types make use of some services of SPADE. The most important of such services, also depicted in Figure 6, are now discussed. The first one is the Presence Notification service, which is used by volunteer agents to set their availability state and let their contacts know about it without explicitly sending any messages. The possible states are: *inactive* (not available), *active* (available to be assigned deliveries), and *on_route* (following a route to deliver some assigned goods). This way, the planner agent and the fleet manager agent may know in real time which volunteers are available to be assigned to each delivery.

Another service that is used by the application is the Pub-Sub service, which is a standard extension of XMPP that implements the Publish-Subscribe protocol.[17] By using this service, agents in SPADE can subscribe to a type of event and get automatic notifications whenever any other agent publishes an event which belong to that type. In this application, this service is used to issue delivery requests. In particular, customer agents publish an event of this type whenever they need to request a new delivery, effectively communicating this request to the appropriate agents, which are subscribed to it (i.e., the Planner agent, or the Fleet agent corresponding to this customer). This is an example of how SPADE can be extended by introducing protocol extensions (XEPs) that are standard and publicly available.

The last two main services used by the application are the Registration/Deregistration service and the Security service. Both are basic services which are part of the core of SPADE. The registration is mandatory, since every user must be authenticated in the system with a previous registration. Regarding security, this application uses some standard encryption protocols in order to ensure that all communications preserve the privacy of customer's health-related data (in particular, SSL for encrypting communications and SASL for securing authentication).

Finally, each agent in the system, as a SPADE agent, has its own web server, as explained in the previous section. This web server may visualize several pieces of data, for example, information related to the messages sent to the agent. In this sense, this web server may be used as the user interface for the agent, which can be accessible through any kind of device with internet connection.

B. EVALUATION

As explained above, the current version of the system has been implemented and tested in SPADE 3 as a prototype integrated into a fleet simulator on urban areas (in particular, using the city of Valencia, Spain). Extending the system to

[17]https://xmpp.org/extensions/xep-0060.html

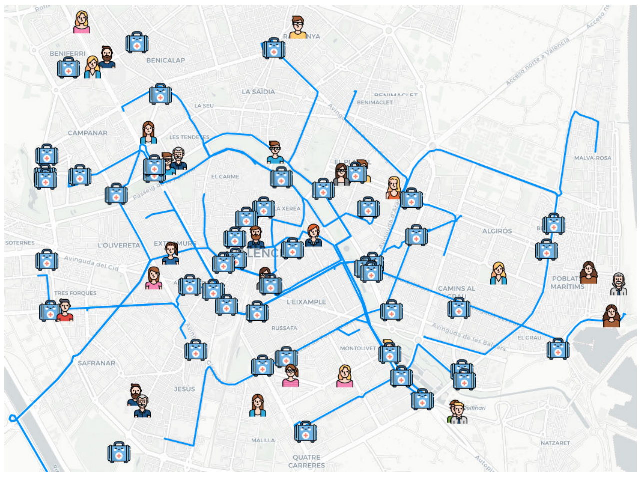


FIGURE 7. **A snapshot of the simulation environment while running one of the tests in the experiment. The snapshot displays the current geo-location of each volunteer and customer agents on the city map, as well as the delivery paths taken by volunteers to pick up and hand over the goods (blue lines).**

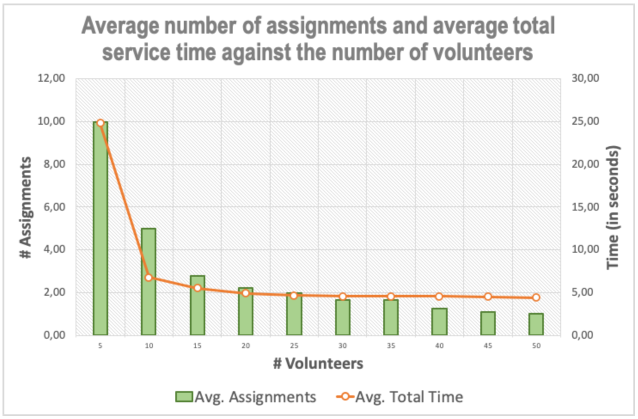


FIGURE 8. **Graph with the simulation results of the experiment of 50 randomly distributed delivery requests and varying number of available volunteers (each combination presents the average values of 10 single tests). For each combination, the graph shows the average number of assignments that each volunteer carried out (in green bars) and the average total delivery time needed for serving all the 50 requests (in an orange line with dots).**

other bordering cities would be as easy as including additional planner agents in charge of computing the delivery routes on these cities and then accepting volunteers and customers geo-located on these areas. Such city planners could be federated or not, and all these agents could be connected to a single or several distributed XMPP servers, depending on the total number of agents involved, the geographic distribution of users, and certain security and performance criteria.

A series of experiments (simulations) have been designed and executed, with the objective of demonstrating the capabilities of SPADE 3 and the usefulness of the proposed system. For example, a particular experiment was carried out in order to analyze the number of volunteers that would be required in order to guarantee a given quality of service, in terms of average delivery time to customers. In the experiment, the number of customers was set to 50, each one issuing a single delivery request. A Poisson distribution was used in order to randomly setting the arrival times of the requests during the simulation. With this, several tests were configured with an increasing number of volunteers, from 5 to 50 in steps of 5. In particular, for each number of volunteers, 10 tests with randomly-generated geo-locations for customers and volunteers were executed, with two metrics computed for each test: the number of deliveries each volunteer had to perform during the test (in order for the system to serve all 50 requests), and the average delivery time of all 50 deliveries (from the time the customer issued the request to the time when the goods were delivered). As a sample of how the system works, Figure 7 shows a snapshot of the simulation environment while running one test of this experiment. In the snapshot, the current location of each volunteer and customer is positioned on the city map and blue lines mark the delivery routes made by volunteers in order to pick up and hand over the goods. The simulation shows all the registered volunteers (whether being currently available or not) but it only displays the customers who presently have pending requests.

Figure 8 presents a graph which summarizes the results of the experiment. In the graph, the x-axis represents the number of volunteers in the tests, while the two y-axis represent the average values of the two metrics computed for each test, in all the tests with the same number of volunteers. The graph clearly shows that, for this number of customers and their expected distribution of requests, there is a very small gain in service time from 15 volunteers up. On the other hand, another way of analyzing the results is from the perspective of the volunteers, who are not usual transporters in the sense that they do not perform the deliveries for money. From this perspective, it is possible to establish the number of volunteers which should be available in order to avoid overloading them with more than N deliveries per day. The graph shows that the number of available volunteers should be increased at least to 20 in order to assign most of them a maximum of 2 deliveries per day.

It is worth noting that the utility of simulations as the one presented above is to study the viability and usefulness of a particular approach before implementing a fully operational application. In this case, the experiment shows that the final application, working with real data about customer requests and volunteers, could be used in order to establish, in a very simple way, an estimate of the number of volunteers a city may need in order to offer a balance between a certain quality of service and a reasonable use of volunteers in the distribution of essential goods. The simulation would also be useful in testing some typical delivery scenarios in order to set some parameters that would work better in the final application, such as different ways of grouping volunteers or alternative matching (or negotiation) algorithms for assigning requests to volunteers, for example.

VI. CONCLUSION

This article has presented an analysis of the current state of agent platforms, by which it has been possible to identify a set of challenges or open issues that platforms experience when facing some relevant problems and domains of interest, nowadays and also expected in the near future. In short, these challenges can be summarized as follows: to be able to offer a simple and effective communication channel, as extensible and standardized as possible; to possess a high degree of elasticity in communication (in the sense of being unaffected by unexpected growths in the number of messages or entities participating in the communication); to incorporate effective mechanisms which allow humans to be "in the loop"; to enable the system to be fully open; and to provide support independently of the running device of each agent.

Moreover, the paper has also introduced SPADE 3, the new re-envisioning of the SPADE middleware based on those open issues. SPADE agents were conceived from the perspective of integrating them in a communication environment initially designed for humans only, as it is the XMPP protocol. This protocol is one of the cornerstones of SPADE. Not only it provides a natural human-in-the-loop integration, but also an open, decentralized, federated protocol that is extensible and offers some interesting utilities (e.g., the presence notification mechanism). SPADE makes full use of all these features in order to provide its support.

Lastly, the paper has presented a case of study where these features have been applied for developing a system to manage the distribution of essential goods to vulnerable people by volunteers, which may be useful in the current pandemic situation. This system has applied a crowdsourcing approach of the last mile delivery problem in order to dynamically allocate volunteers which may deliver the essential goods while making their own habitual trips through the city. A prototype of the system running inside a urban fleet simulator has also been proposed, as a prototype and test bed for the real-world system, which will be developed in the near future.

Using SPADE in this scenario has provided several advantages. First, the use of an extensible platform allows the system to be easily adapted to a sudden growth in the number of users. Second, the possibility of federalization when managing different groups of volunteers (or "fleets" in the implemented prototype) permits an elastic scaling of the system in the case of running large experiments in the simulator (or running the real system in big cities). Third, the dynamic registration and deregistration mechanism provided by the system ensures a completely open system, which is essential in a fully collaborative scenario as the one proposed here. Fourth, this case required a straightforward yet consistent synchronization technique in order to determine the availability of volunteers, which has been efficiently achieved by means of the presence notification mechanism. Fifth, the possibility of extending the communication protocol with standard add-ons allowed the system to easily incorporate new support, such as the publish-subscribe protocol which was used to receive the delivery requests by customers. And finally, it is also worth noting that by using a protocol that allows for the integration of people-to-people and people-to-agent communication, moving this application from the simulator to a real environment is quite simple. In this case, the communication protocol with the agents will be maintained, and only an app (maybe as simple as a chat app) will be needed for the humans to communicate with the agents. In scenarios like this one, where humans are an important component of the system, the integration of people-to-people communication in SPADE enables the incorporation of situations where communication between humans and agents, or even exclusively between humans, is required.

REFERENCES

[1] M. E. Gregori, J. P. Cámara, and G. A. Bada, "A jabber-based multi-agent system platform," in *Proc. 5th Int. Joint Conf. Auto. Agents Multiagent Syst. AAMAS*, 2006, pp. 1282–1284.

[2] F. Bellifemine, A. Poggi, and G. Rimassa, "JADE—A FIPA-compliant agent framework," in *Proc. PAAM*, London, U.K., vol. 99, nos.97–108, 1999, p. 33.

[3] F. Bergenti, G. Caire, S. Monica, and A. Poggi, "The first twenty years of agent-based software development with JADE," *Auto. Agents Multi-Agent Syst.*, vol. 34, no. 2, Oct. 2020.

[4] S. Poslad, P. Buckle, and R. Hadingham, "The fipa-os agent platform: Open source for open standards," in *Proc. 5th Int. Conf. Exhib. Practical Appl. Intell. Agents Multi-Agents*, vol. 355, 2000, p. 368.

[5] G. Caire, D. Gotta, and M. Banzi, "WADE: A software platform to develop mission critical applications exploiting agents and workflows," in *Proc. 7th Int. Joint Conf. Auto. Agents Multiagent Syst., Ind. Track*, 2008, pp. 29–36.

[6] F. Bergenti, "An introduction to the JADEL programming language," in *Proc. IEEE 26th Int. Conf. Tools With Artif. Intell.*, Nov. 2014, pp. 974–978.

[7] J. P. Bigus, D. A. Schlosnagle, J. R. Pilgrim, W. N. Mills III, and Y. Diao, "ABLE: A toolkit for building multiagent autonomic systems," *IBM Syst. J.*, vol. 41, no. 3, pp. 350–371, 2002.

[8] O. Gutknecht and J. Ferber, "The madkit agent platform architecture," in *Proc. Workshop Infrastruct. Scalable Multi-Agent Syst. Int. Conf. Auto. Agents*. Berlin, Germany: Springer, 2000, pp. 48–55.

[9] S. Galland, N. Gaud, S. Rodriguez, and V. Hilaire, "JANUS: Another yet general-purpose multiagent platform," in *Proc. 7th AOSE Tech. Forum*, Paris, France, 2010, pp. 2–55.

[10] M. Baldoni, C. Baroglio, F. Capuzzimati, and R. Micalizio, "Commitment-based agent interaction in JaCaMo+," *Fundamenta Informaticae*, vol. 21, pp. 1001–1030, Jan. 2016.

[11] R. H. Bordini and J. F. Hübner, "Bdi agent programming in agentspeak using Jason," in *Proc. Int. Workshop Comput. Log. Multi-Agent Syst.* Berlin, Germany: Springer, 2005, pp. 143–164.

[12] A. Ricci, M. Viroli, and A. Omicini, "Cartago: A framework for prototyping artifact-based environments in mas," in *Proc. Int. Workshop Environ. Multi-Agent Syst.* Berlin, Germany: Springer, 2006, pp. 67–86.

[13] M. Hannoun, O. Boissier, J. S. Sichman, and C. Sayettat, "Moise: An organizational model for multi-agent systems," in *Advances in Artificial Intelligence*. Berlin, Germany: Springer, 2000, pp. 156–165.

[14] F. Piette, C. Caval, C. Dinont, A. E. F. Seghrouchni, and P. Tailliert, "A multi-agent solution for the deployment of distributed applications in ambient systems," in *Proc. Int. Workshop Eng. Multi-Agent Syst.* Berlin, Germany: Springer, 2016, pp. 156–175.

[15] A. E. Fallah-Seghrouchni, F. Piette, C. Caval, and P. Taillibert, "A multi-agent platform for the deployment of ambient systems," *Int. J. Agent-Oriented Softw. Eng.*, vol. 6, nos. 3–4, pp. 369–401, 2018.

[16] T. Suganuma, T. Oide, S. Kitagami, K. Sugawara, and N. Shiratori, "Multiagent-based flexible edge computing architecture for IoT," *IEEE Netw.*, vol. 32, no. 1, pp. 16–23, Jan. 2018.

[17] I. Ayala, M. Amor, and L. Fuentes, "The sol agent platform: Enabling group communication and interoperability of self-configuring agents in the Internet of Things," *J. Ambient Intell. Smart Environ.*, vol. 7, no. 2, pp. 243–269, 2015.

[18] M. Pipattanasomporn, M. Kuzlu, W. Khamphanchai, A. Saha, K. Rathinavel, and S. Rahman, "BEMOSS: An agent platform to facilitate grid-interactive building operation with IoT devices," in *Proc. IEEE Innov. Smart Grid Technol. Asia (ISGT ASIA)*, Nov. 2015, pp. 1–6.

[19] T. Semwal, S. Nikhil, S. S. Jha, and S. B. Nair, "Tartarus: A multi-agent platform for bridging the gap between cyber and physical systems," in *Proc. Int. Conf. Auto. Agents & Multiagent Syst.*, 2016, pp. 1493–1495.

[20] J. Mell and J. Gratch, "Grumpy & pinocchio: Answering human-agent negotiation questions through realistic agent design," in *Proc. 16th Conf. Auto. Agents Multiagent Syst.*, 2017, pp. 401–409.

[21] P. Taillandier, B. Gaudou, A. Grignard, Q.-N. Huynh, N. Marilleau, P. Caillou, D. Philippon, and A. Drogoul, "Building, composing and experimenting complex spatial models with the GAMA platform," *GeoInformatica*, vol. 23, no. 2, pp. 299–322, Apr. 2019.

[22] A. Sanchis, V. Julián, J. M. Corchado, H. Billhardt, and C. Carrascosa, "Using Natural Interfaces for Human-Agent Immersion," in *Highlights of Practical Applications of Heterogeneous Multi-Agent Systems*. Cham, Switzerland: Springer, 2014, pp. 358–367.

[23] M. E. Gaston and M. desJardins, "Agent-organized networks for dynamic team formation," in *Proc. 4th Int. Joint Conf. Auto. Agents Multiagent Syst. AAMAS*, 2005, pp. 230–237.

[24] P. R. Cohen, H. J. Levesque, and I. Smith, "On team formation," in *Contemporary Action Theory*, vol. 2, G. Holmstrom-Hintikka and R. Tuomela, Eds. Dordrecht, The Netherlands: Kluwer, 1997, pp. 87–114.

[25] Y. Jiang and J. Jiang, "Contextual resource negotiation-based task allocation and load balancing in complex software systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 5, pp. 641–653, May 2009.

[26] Y. Jiang, Y. Zhou, and W. Wang, "Task allocation for undependable multiagent systems in social networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 8, pp. 1671–1681, Aug. 2013.

[27] B. Ulicny and D. Thalmann, "Crowd simulation for interactive virtual environments and vr training systems," in *Comput. Animation Simulation*. Berlin, Germany: Springer, 2001, pp. 163–170.

[28] J. E. Almeida, R. J. F. Rosseti, and A. Leça Coelho, "Crowd simulation modeling applied to emergency and evacuation simulations using multi-agent systems," 2013, *arXiv:1303.4692*. [Online]. Available: http://arxiv.org/abs/1303.4692

[29] M. Georgeff, "Communication and interaction in multi-agent planning," in *Readings in Distributed Artificial Intelligence*. Amsterdam, The Netherlands: Elsevier, 1988, pp. 200–204.

[30] S. Resmerita and M. Heymann, "Conflict resolution in multi-agent systems," in *Proc. 42nd IEEE Int. Conf. Decis. Control*, Dec. 2003, pp. 2537–2542.

[31] R. Aydogan, V. Sanchez, V. Julian, J. Broekens, and C. Jonker, "Computational approaches for conflict resolution in decision making: New advances and developments," *Cybern. Syst.*, vol. 45, no. 3, pp. 217–221, Apr. 2014.

[32] P. Saint-Andre, *Extensible Messaging and Presence Protocol (XMPP): Core*, document RFC 6120, Internet Requests for Comments, RFC Editor, Mar. 2011. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6120.txt

[33] J. Palanca, A. Terrasa, C. Carrascosa, and V. Julián, "SimFleet: A new transport fleet simulator based on MAS," in *Highlights of Practical Applications of Survivable Agents and Multi-Agent Systems. The PAAMS Collection*. Berlin, Germany: Springer, 2019, pp. 257–264.

[34] J. Palanca, A. Terrasa, C. Carrascosa, and V. Julián, "Improving the programming skills of students in multiagent systems master courses," *Comput. Appl. Eng. Edu.*, vol. 27, no. 4, pp. 836–845, Jul. 2019.

[35] TIOBE Software. (Jun. 2020). *TIOBE Programming Community Index*. TIOBE Software. Accessed: Jun. 18, 2020. [Online]. Available: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

[36] T. Dierks and E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2*, document RFC 5246, Internet Requests for Comments, RFC Editor, Aug. 2008. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5246.txt

[37] A. Straub, D. Gultsch, T. Henkes, K. Herberth, P. Schaub, and M. WiBfeld, *XEP-0384: OMEMO Encryption*, document XEP 0384, XMPP Extension Protocol, Jun. 2009. [Online]. Available: https://xmpp.org/extensions/xep-0384.html

[38] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer, *OpenPGP Message Format*, document RFC 4880, Internet Requests for Comments, RFC Editor, Nov. 2007. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4880.txt

[39] P. D. O'Brien and R. C. Nicol, "FIPA—Towards a standard for software agents," *BT Technol. J.*, vol. 16, no. 3, pp. 51–59, 1998.

[40] A. Giret, C. Carrascosa, V. Julian, M. Rebollo, and V. Botti, "A crowdsourcing approach for sustainable last mile delivery," *Sustainability*, vol. 10, no. 12, p. 4563, Dec. 2018.

JAVIER PALANCA received the Ph.D. degree from the Universitat Politècnica de València (UPV), in 2012. He is currently a Senior Researcher with UPV. He has participated in several research projects related to multi-agent systems, artificial intelligence, recommender systems, smart cities, and social network analysis.

ANDRÉS TERRASA received the B.S. and Ph.D. degrees in computer science from the Universitat Politècnica de València (UPV), Spain, in 1995 and 2001, respectively. He is currently working as an Associate Professor with the Department of Information Systems and Computation (DSIC), UPV. He is also a member of the Group of Information Technology/Artificial Intelligence (GTI/IA) Research Group. His research interests include real-time systems, realtime artificial intelligence, and multi-agent systems.

VICENTE JULIAN received the Ph.D. degree in computer science from the Valencia University of Technology, Spain, in 2002. He is currently a Full Professor with the Departament de Sistemes Informàtics i Computació, Universitat Politècnica de València and a Researcher with the GTI-IA Research Group, Universitat Politècnica de València. His research interests include multiagent systems, agent architectures, agent organizations, multiagent system methodologies, and real-time agents. He is the coauthor of over 200 book chapters, journal articles, technical reports, and so on about these topics.

CARLOS CARRASCOSA was born in Valencia, Spain. He received the M.S. degree in computer science from the Universitat Politècnica de València (UPV), in 1995, and the Ph.D. degree from the Departamento de Sistemas Informáticos y Computación, UPV. He is currently a Lecturer involved in teaching several AI-related subjects at UPV. His research interests include MAS, social emotions, consensus in MAS, intelligent virtual environments, learning, serious games, information retrieval, and real-time systems.