Received: 21 February 2018 | Revised: 30 July 2018 | Accepted: 9 October 2018

DOI: 10.1002/spe.2657

SHORT COMMUNICATION

WILEY

A platform for P2P agent-based collaborative applications

Daniela Briola | Daniela Micucci | Leonardo Mariani

Department of Informatics, Systems and Communication, University of Milano - Bicocca, Milan, Italy

Correspondence
Daniela Briola, Department of Informatics, Systems and Communication, University of Milan Bicocca, Milan, Italy.
Email: daniela.briola@unimib.it

Funding information
EU H2020; ERC Consolidator, Grant/Award Number: ERC 646867

Summary

The operational environment can be a valuable source of information about the behavior of software applications and their usage context. Although a single instance of an application has limited evidence of the range of the possible behaviors and situations that might be experienced in the field, the *collective knowledge* composed by the evidence gathered by the many instances of a same application running in several diverse user environments (eg, a browser) might be an invaluable source of information. This information can be exploited by applications able to autonomously analyze how they behave in the field and adjust their behavior accordingly. Augmenting applications with the capability to collaborate and directly share information about their behavior is challenging because it requires the definition of a fully *decentralized* and *dependable* networked infrastructure whose nodes are the user machines. The nodes of the infrastructure must be *collaborative*, to share information, and *autonomous*, to exploit the available information to change their behavior, for instance, to better accommodate the needs of the users to prevent known problems. This paper describes the initial results that we obtained with the design and the development of an infrastructure that can enable the execution of collaborative scenarios in a fully decentralized way. Our idea is to combine the *agent-based paradigm*, which is well suited to design collaborative and autonomous nodes, and the *peer-to-peer paradigm*, which is well suited to design distributed and dynamic network infrastructures. To demonstrate our idea, we augmented the popular JADE agent-based platform with a software layer that supports both the creation of a fully decentralized peer-to-peer network of JADE platforms and the execution of services within that network, thus enabling JADE multi-agent systems (MASs) to behave as peer-to-peer networks. The resulting platform can be used to study the design of collaborative applications running in the field.

KEYWORDS

agent-based paradigm, diagnosis, distributed multiagent system, JADE, MAS, peer-to-peer systems

1 | COLLABORATIVE APPLICATIONS

Modern software applications have often to face complex and unexpected situations while running in the field. For instance, applications must behave efficiently and effectively despite deployment of upgrades, installation of new add-ons, and changes to configurations. Since it is impossible to predict every possible scenario in advance, the ability of the software to extract information from the field to improve its ability to autonomously respond to changes and unexpected situations is extremely important.

While a single instance of an application may hardly accumulate knowledge quickly, the many instances of a same application running in the field can all together quickly accumulate knowledge, as long as they are in the condition of communicating and interacting efficiently. For example, the knowledge of error-prone behaviors might be exploited to prevent the repeated occurrence of the same failures[1] and to activate collaborative diagnosis and repair procedures.[2,3]

Enabling communication between the instances of a same application requires the availability of a fully *decentralized* network whose nodes are user machines running instances of a same application. The network is clearly highly dynamic, with nodes continuously entering and exiting the system, coherently with the activity performed by the end users. Moreover, the design of distributed procedures requires nodes to be *autonomous*, to take decisions locally, and *collaborative*, to exchange information with other nodes.

There exist architectures that satisfy many of the requirements discussed above; however, there is no single solution that satisfies all of them. For instance, peer-to-peer architectures can be used to design decentralized distributed systems with dynamic topology, but unfortunately, they provide little support to the design of autonomous and collaborative behaviors[4,5]; while agent-based systems can be used to design decentralized systems composed of autonomous and collaborating components, actual platforms provide limited support to continuously changing network topologies.[6,7]

To obtain a decentralized system with dynamic network topology whose nodes are able of sophisticated behaviors, we combined the agent-based and peer-to-peer (P2P) paradigms. We selected the JADE agent-based platform[7] for implementing our P2P layer since JADE is probably the most adopted MASs development platform both in academia and in industry. In particular, we extended JADE with a new layer that transparently enriches the platform with the capability to both create a dynamic P2P network of JADE platforms and invoke services in a P2P fashion, that is, invoking services by iteratively propagating requests to neighbors and incrementally collecting responses. Interestingly, our extension leaves unaltered the JADE platform and makes the already existing MASs developed with JADE executable in a P2P fashion. We experienced our platform with a sample scenario relative to the collaborative detection of problematic configurations and the prevention of the corresponding failures.

The article is organized as follows. Section 2 introduces the JADE Platform and its limitations that motivate the proposed extension. Sections 3 and 4 present the resulting platform and example usages, respectively. Section 5 discusses related work. Finally, Section 6 provides final remarks.

2 | THE JADE PLATFORM AND ITS LIMITATIONS

JADE[7] is a platform for implementing distributed MASs in Java, developed from an industrial project and then widely adopted both in industry and in academia, and today improved with a large set of tools and plugins. Each instance of the JADE platform can execute multiple agents that can communicate with other agents running in the same or other JADE platforms.

Communication between agents in different platforms is possible only if the network addresses of the platforms are known. Otherwise, any kind of communication between agents on different and unknown platforms is impossible. Indeed, JADE does not foresee a mechanism to automatically discover platforms over a network, instead, it assumes that the information about the existing platforms is known and available to the agents that need to communicate.

When a platform is started, the agents that provide the core services of the platform are automatically created. For our purpose, it is worth mentioning the *Directory Facilitator Agent* (DF) and the *Agent Management System* (AMS).

The DF manages a "yellow pages" service that indexes the services provided by the agents in the platform.

Similarly, the AMS maintains a list of all the agents that have been created in the platform. Each agent can be identified by a globally unique name in the form ⟨`AgentName`⟩`@`⟨`PlatformName`⟩ plus the physical address of the platform. So, the AMS agent in platform P1 running on a physical devices with IP address 149.132.149.72 can be globally identified by the unique name `AMS@P1` on 149.132.149.72.

The unique name of an agent can be used when an agent needs to communicate with another agent hosted on a different platform.

The JADE platform and the agent-based paradigm in general do not satisfy two key requirements to achieve a networked architecture with dynamic and unmanaged topology:

1. *Network discovery*: nodes, that is, the JADE Platforms, are not known a priori but must be automatically and dynamically discovered, flexibly adapting the shape of the network to the available nodes.
2. *Requests propagation*: services must be invoked without requiring the knowledge of the agents that provide them. This is usually achieved by propagating requests and responses within the P2P network following links to neighbors.[4,5]

With our extension, we aim at providing a layer overcoming these limitations to JADE and consequently to the large community developing MASs with these needs.

3 | HYBRID PEER-TO-PEER AGENT-BASED SOLUTION

In order to achieve a networked architecture supporting dynamic topology while exploiting the agent-based paradigm, we designed a P2P layer managed by two agents that can be used as part of an agent-based system to make it a hybrid agent/P2P system. We illustrate the main concepts in the context of JADE, but our ideas can be easily adapted to any other agent-based platform that runs yellow page services that track the existing agents and services, which are common in several platforms.

The P2P layer that we defined consists of two agents: the *Peer2Peer Agent Management System* (P2PAMS), which handles the dynamic topology of the network by incrementally and regularly discovering platforms (the peers of the network), and the *Service Manager* (SM), which enables running any service in a P2P fashion by handing the propagation of the requests and the collection of the responses. The following two subsections describe these two agents more in details.

3.1 | P2PAMS agent: Network creation and maintenance

The P2PAMS agent is available on every platform and is responsible for dynamically discovering and maintaining the network structure. As in most P2P systems, it is assumed the existence of a publicly available list of platforms that are likely to be up and running at any time.[4] These platforms are the entry points for any other platform joining the system for the first time.

Once a JADE platform is part of the network, its P2PAMS agent proactively starts running the discovery protocol: the agent interacts with the P2PAMS agents running in the other platforms to mutually exchange the list of the known peers, thus increasing the set of peers known by each platform. This mechanism lets each platform to quickly obtain a partial but nontrivial view of the peers available in the network.

The platform discovery protocol can be configured according to the following four parameters depending on the specific characteristics of the network and the kinds of services that must be provided.

- *Discovery rate*, which defines how frequently the P2PAMS agent must execute the protocol to discover new platforms by exchanging the list of known platforms with other P2PAMS agents.
- *Flooding rate*, which defines the maximum number of peers that can be contacted by the P2PAMS at each round. This is used to prevent the P2PAMS from flooding the network with too many messages. If the number of known platforms is higher than the flooding rate, the peers to be contacted can be selected according to different criteria (eg, randomly or cyclically).
- *List size*, which limits the number of platforms that a peer may know to a user-defined value.
- *Unreachability threshold*, which defines when a peer is considered to be offline. Each platform regularly pings the peers in the list of known platforms. If a peer does not respond for the number of times defined by this parameter, it is assumed to be offline and is dropped from the list of known platforms. This parameter is defined based on the evolution rate of the network. Since this protocol may lead to oscillations, the agent maintains a list of "Offline Platforms" and the P2PAMS will not contact again a platform if it appears in that list. The *unreachability threshold* parameter is used to clean this list too: a platform will remain in that list only for *unreachability threshold* executions of the protocol, then it will be removed (and can be contacted again).

Each P2PAMS agent can be configured in different ways, depending for example on specific strategies and resources availability.

To optimize the discovery of the network, the information obtained by the P2PAMS is persisted regularly so that, if a peer goes offline and then comes back online again, it can start the discovery procedure from the known set of peers. Of course, several of these peers might be offline at that point, but the presence of several other running peers in the list at start-up will speed up the process of joining the network.

3.2 | SM agent: Handling distributed service invocations

Network discovery and management are fundamental functionalities to run an agent-based system within a P2P network. However, they are not enough: it is also necessary that the services provided by each agent in each platform are invocable from the other platforms exploiting the P2P network. The P2P layer nonintrusively achieves this result by means of the *Service Manager* (SM) agent, which is responsible for offering a new type of service request style to the agents in the platform. Moreover, SM allows an agent to obtain the required answers from the other agents in the network transparently, exploiting the connections of the P2P network. In particular, the SM agent propagates the request on the network reaching the agents that may respond without requiring the agent that has originated the request to be aware of the identity of the responding agents.

The behavior of the SM exploits both the existence of the standard DF JADE agent, which stores the list of the services offered locally to a platform, and ontologies shared between the network nodes, which specify how services can be invoked. Note that this solution implements the P2P paradigm *exploiting the agent-based paradigm without introducing any change to the underlying platform*, thus also retaining all the original capabilities of the platform.

In a nutshell, when an agent needs to request a service to other peers, the SM agent propagates the request of the agent within the network until the required number of answers is collected. Since the SM might be overloaded with requests and might have to manage many parallel conversations with many platforms, our design naturally splits the workload among *Assistants* that are dynamically created by the SM one per conversation that must be managed.

The protocol for propagating service requests is graphically illustrated in Figure 1. The protocol starts from a Source Agent that asks the SM for a service `S` with parameters `[Params]` (if any), expecting `#Answ` responses. This request is received by the SM agent which creates the Assistant that then first checks whether the request can be satisfied by the locally available agents. If not, the Assistant interacts with the P2PAMS agent to extract the set of known platforms, selects a subset of the platforms, and forwards the request to them by contacting the SM agent running at those platforms (equally dividing the `#RemainingAnsw` between the contacted platforms). All the agents requested to perform the service `S` are asked to send the result back to the Assistant. If enough responses are collected, the set of responses is sent back to the Source Agent; otherwise, the protocol is iterated until enough responses have been collected, selecting a different subset of platforms at each iteration. Since the agents that receive the requests do not propagate them to their neighbors but is

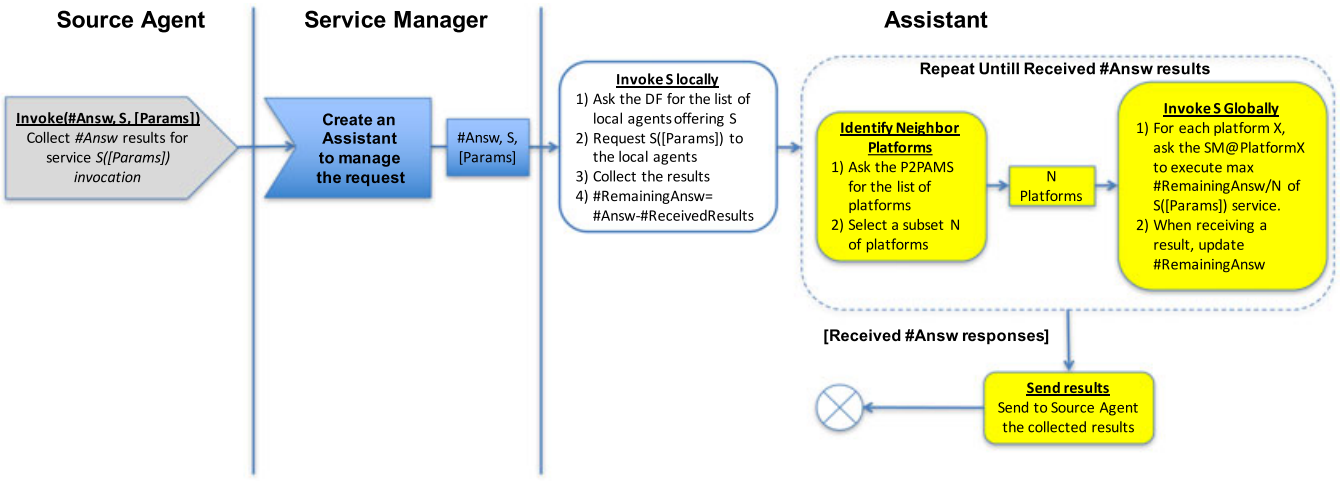

FIGURE 1 The agent-based protocol to propagate requests for services [Colour figure can be viewed at wileyonlinelibrary.com]

the assistant that simply iterates its protocol contacting different agents until collecting enough results, the network is very unlikely to be flooded with redundant requests.

Both the policy for selecting the platforms to interact with at each iteration and the time before a new iteration is started can be configured individually for each platform.

The Source Agent can request the SM to perform this service in "batch-mode," thus waiting for only one answer with the complete result (as described before), or "incrementally," which requires the Assistant sending back each response as soon as it is received.

4 | SCENARIOS: NETWORK CREATION AND FAILURE PREVENTION

This section presents sample usage scenarios of the two key features of the resulting hybrid architecture: the network creation and the requests propagation.

4.1 | Network creation

In this case, we illustrate how platforms discover each other with the network discovery protocol. In the example, `Platform1` is always up and running, and it is listed in the publicly available list "AlwaysUpAndRunningPlatforms List" on a known website. Initially, `Platform1` does not know any other platform. *Flooding rate* and *list size* parameters of all the involved platforms are set to 100. The *discovery rates* for `PlatformA` and `PlatformB` are set to 10 and 100 milliseconds, respectively.

`PlatformA` and `PlatformB` are JADE platforms augmented with our P2P layer. The two platforms are created and started on different physical nodes. Figure 2 shows the logical and temporal flow of the messages exchanged between the involved P2PAMS agents on the platforms. Each P2PAMS, when created, reads from the "AlwaysUpAndRunningPlatforms List" the list of known alive platforms, which consists of *Platform1* only (messages 2 and 6).

Then, cyclically, after *discovery rate* milliseconds, every P2PAMS agent starts the network discovery protocol by sending an "AskForKnownPlatforms" message to *flooding rate* known platforms. In this example, the first time `PlatformA` searches for new platforms (message 4), it can only ask to the P2PAMS agent on `Platform1` because it is the only known platform. The same happens for `PlatformB` (message 8). Since `Platform1` does not know any other running platform, it answers to the request by `PlatformA` with an empty list (return message 4.1) and then it adds `PlatformA` in its list of known platforms (message 4.2). When `Platform1` receives the request from `PlatformB` (message 8), it replies with a list containing *PlatformA* (return message 8.1). Then, `Platform1` adds `PlatformB` to its list of known Platforms (message 8.2), and similarly, `PlatformB` adds `PlatformA` to its list (message 9).

At the second round of the protocol, `PlatformA` still knows only `Platform1` and asks it again for the known platforms (message 10). At this time, `Platform1` replies with a list containing *PlatformB* (message 10.1), and `PlatformA` adds it to its list of known platforms (message 11). In the next round, `PlatformA` will be able contact both `Platform1` (message 12) and `PlatformB` (message 13) for known platforms.

At the second round of the protocol (ie, after 100 milliseconds), `PlatformB` also contacts `Platform1` and `PlatformA` for known platforms (not shown in the figure), and so on. It is possible to notice that, in this simple example, after two rounds of the protocol, all the platforms discovered one the other.

4.2 | Service discovery and invocation for failure prevention

In this scenario, we illustrate how agents invoke services exploiting the P2P network to share information about crashes and problems related to FireFox.[8] This collective knowledge may support an agent in preventing the failures already experienced by the other Firefox users.

To this end, we defined the *FireFox Failure Prevention* (FFP) agent that is in charge of supervising at runtime the execution of a FireFox instance, preventing failures when possible. The agent can access both the actual FireFox configuration, including the list of installed plug-in, and the crash report of the application when a crash is reported. When the FFP agent detects a crash that involves a plugin (eg, the plugin occurs in the stack trace), it assumes the plugin might be responsible for the observed problem. To confirm this guess, the FFP agent interacts with other FFP agents looking for similar evidence. If this is evidence is confirmed, the FFP agent suggests the user to uninstall or replace the plug-in to improve stability of the environment. Below we show how this scenario can be implemented on the proposed platform.

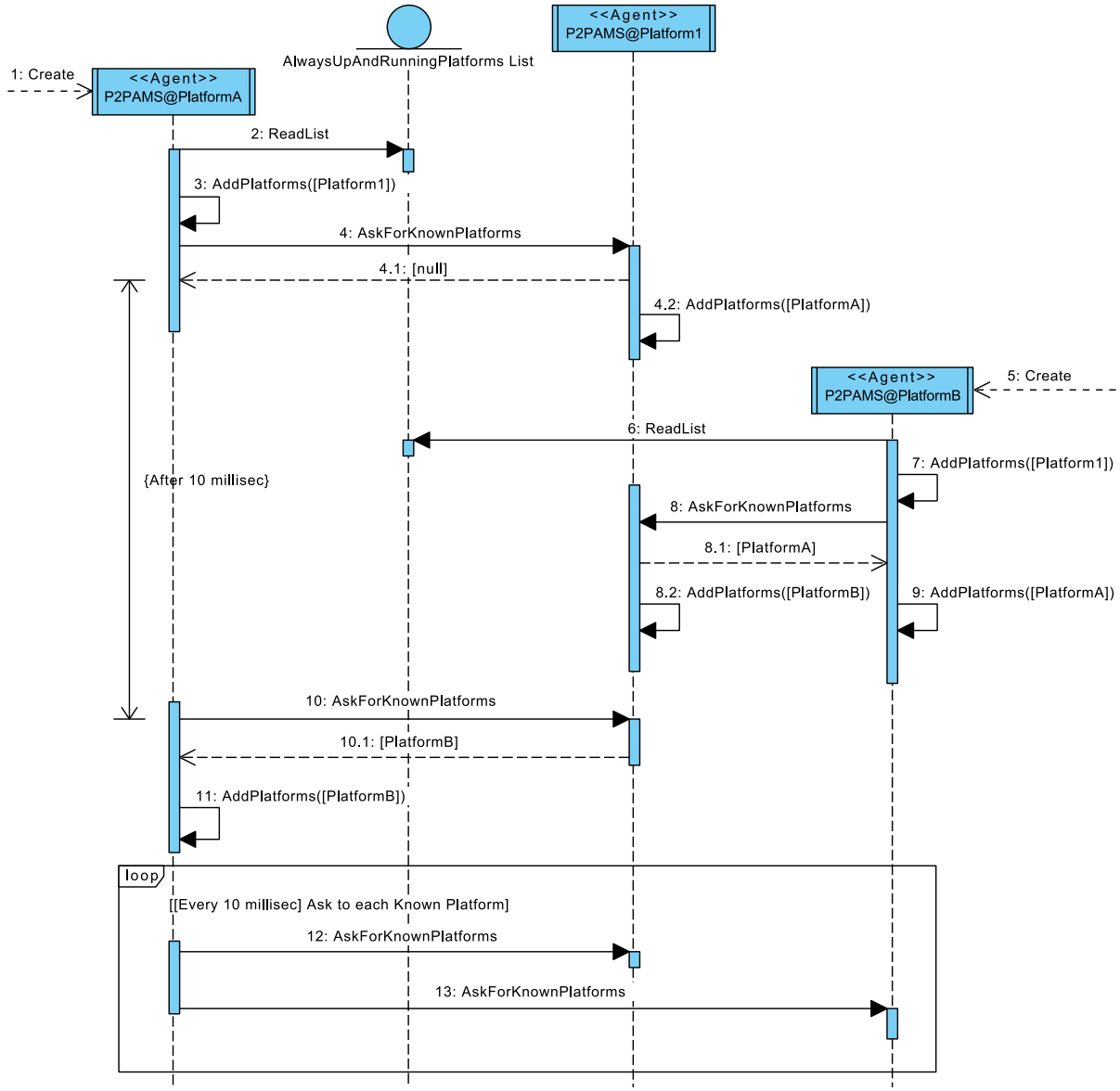

FIGURE 2 Sample execution required to build a fully connected network of three platforms [Colour figure can be viewed at wileyonlinelibrary.com]

Suppose to have the three platforms resulting from the previous scenario (`Platform1, PlatformA, PlatformB`) and that they already discovered one the other using the network discovery protocol. The three platforms offer services by means of dedicated agents. In particular, they all include the FFP agent, which provides the `IsPlugInSuspicious` service that, given a plug-in, returns a boolean value indicating if the agent considers the specified plug-in as likely responsible for failures in FireFox.

Figure 3 shows the platforms and the hosted agents (`PlatformC` is already shown in the Figure 3, but it will join the network later, as described below). In each platform, we distinguish three logical layers. The Native JADE layer hosts the original agents of the platform, that is, the agents provided by JADE (in this specific example, we show the DF agent only). The P2P layer hosts the agents that handle the P2P network and enable interactions in a P2P style, that is, the SM and P2PAMS agents. The Application layer hosts the agents offering services and implementing the application logic. In this specific example, the Application layers hosts FFP agents.

Suppose that the FireFox instance on `Platform1` crashes, and that `FFP1` (an FFP agent), analyzing the crash report and the stack trace, identifies a reference to a specific plug-in (`BuggyPlug-In`) installed on FireFox involved in the crash.

The `FFP1` agent records this information and asks its neighbors (other FFP agents running on other platforms) if they are aware of crashes involving `BuggyPlug-In`. When receiving a sufficient number of replies (eg, three in this example),

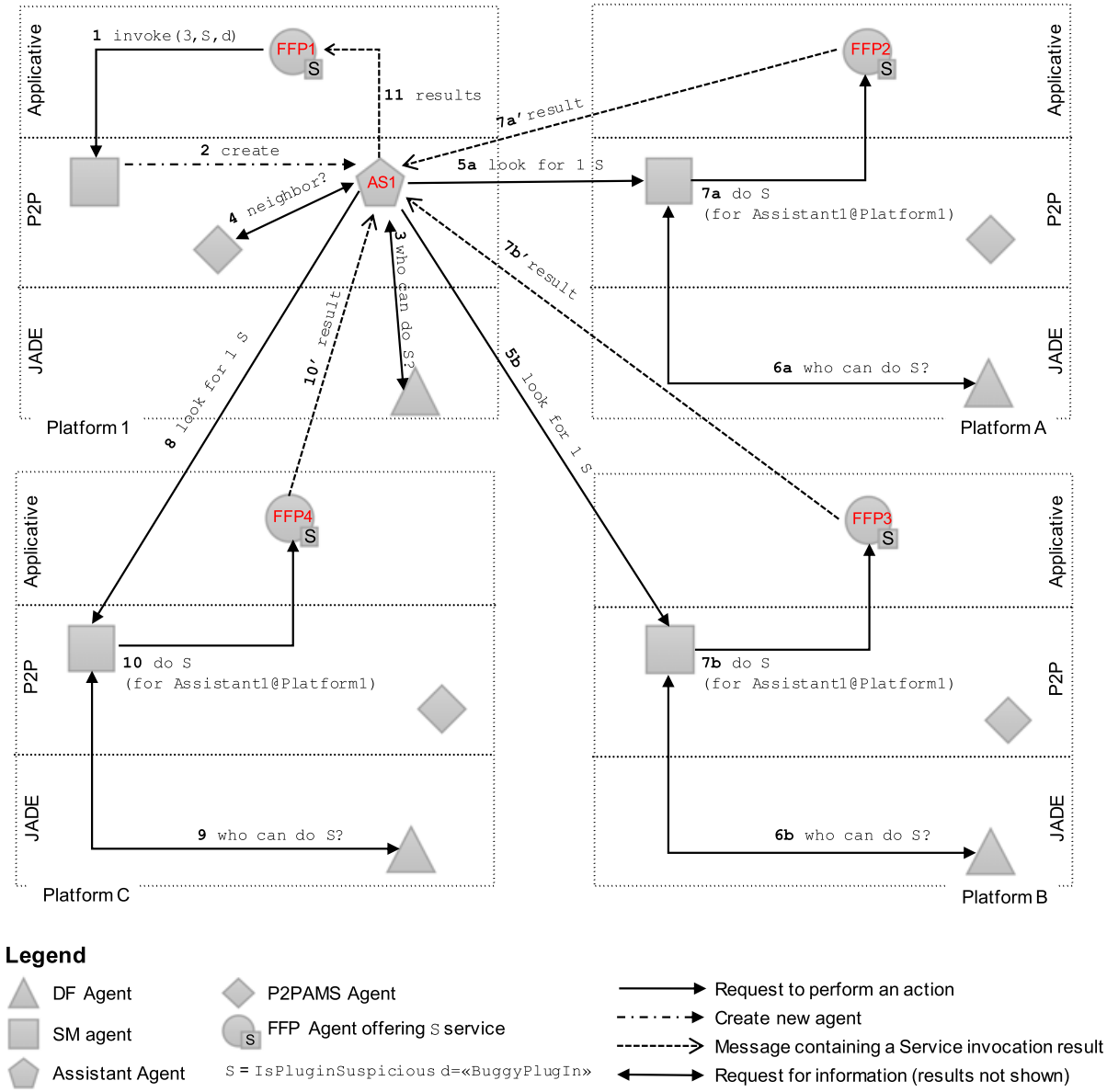

FIGURE 3 Exchanged messages and main steps needed to invoke `IsPlugInSuspicious` in a peer-to-peer P2P style. DF, Directory Facilitator Agent; FFP, FireFox Failure Prevention; P2PAMS, Peer-to-Peer Agent Management System; SM, Service Manager [Colour figure can be viewed at wileyonlinelibrary.com]

the agent can infer that the plugin is unsafe and suggest the user to uninstall it. Of course, more sophisticated criteria can be used to associate a plugin with the stability of the hosting application, yet this does not affect how the service discovery and invocation protocols can be used.

To this end, `FFP1` asks for `invoke(3, IsPlugInSuspicious, ["BuggyPlug-In"])` to the local `SM` agent (message 1 in Figure 3), which in turn creates the `AS1` Assistant dedicated to serve this request (message 2).

The dedicated Assistant `AS1` starts from the agents available locally to the platform, thus it asks the local `DF` agent for the list of the local agents offering the service `IsPlugInSuspicious` (message 3). The returned list contains only `FFP1`, which is ignored by `AS1`.

To satisfy the request by `FFP1` agent, `AS1` still has to collect the three responses, so it asks its local `P2PAMS` agent for a list of neighbors (message 4). The `P2PAMS` agent returns a list of neighbors containing `PlatformA` and `PlatformB`. The `AS1` agent then asks each `SM` agent on the two platforms to search locally for one agent that offers the service `IsPlugInSuspicious` and to ask them, if any, to perform the service (messages 5a and 5b). Note that, to avoid requesting too many service executions (that should be thrown away if the requested number has been already collected), the `AS1` asks only one result from each platform.

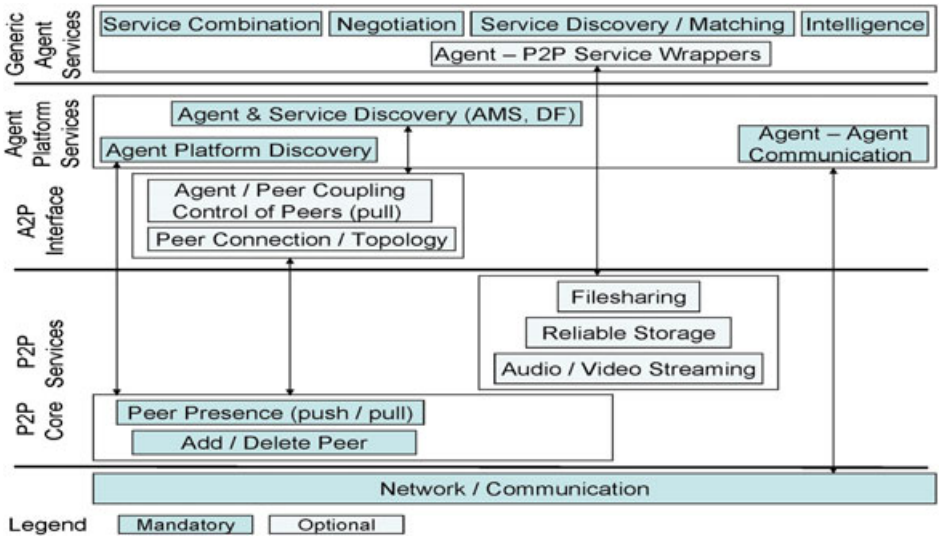

FIGURE 4 The Foundation for Intelligent Physical Agents architecture with integrated peer-to-peer (P2P) capabilities for MASs (from the work FIPA working group for the integration of P2P and agents[10]) [Colour figure can be viewed at wileyonlinelibrary.com]

Thus, `SM@PlatformA` asks its local `DF` for agents performing the `IsPlugInSuspicious` service (message 6a). The `SM` replies with a list containing `FFP2`. The `SM@PlatformA` then asks `FFP2` to perform service `IsPlugInSuspicious(BuggyPlug-In)` (message 7a) and to send back the result to `AS1@Platform1`. Similarly, `SM@PlatformB` agent asks its local `DF` for agents performing the service `IsPlugInSuspicious` and then asks the `FFP3` to do the same (messages 6b and 7b). At the end of this process, the `AS1@Platform1` agent receives two responses, one from `FFP2@PlatformA` and one from `FFP3@PlatformB` (messages 7a′ and 7b′ respectively) but still miss one result to reach three.

Now, assume that a new platform (`PlatformC`) running the P2P layer and with a FFP agent (`FFP3`) connects to the network. In short time, the `PlatformC` platform will be connected to the network (as described in the previous section) so that, when the `AS1` iterates the protocol to collect the missing answer, the agent will be able to receive it from `FFP3` (messages 8, 9, 10, and 10′).

If the collected answers show that `BuggyPlug-In` is indeed likely to be responsible for several failures, the `FFP1` suggests the user to uninstall it.

5 | RELATED WORK

The Foundation for Intelligent Physical Agents (FIPA) consortium,[9] which is the reference point for the standards in MASs, has been working for a while on a standard for augmenting FIPA compliant MASs with P2P capabilities.[10] Today, the document is still a draft, and no implementation is available.

The architecture and implementation proposed in this paper have connections with the FIPA proposal. With reference to the FIPA proposal shown in Figure 4, the P2PAMS agent in our P2P layer manages both the "FIPA Agent platform discovery" activity in the "Agent Platform Services" layer and the services in the "A2P Interface." Differently form the FIPA proposal, in which each agent could be a peer, our architecture assumes that a peer corresponds to a platform and multiple agents can be running on a same peer (ie, platform). Moreover, our SM agent in the P2P layer offers a "P2P-based service invocation," which, in the FIPA vision, is similar to the "Agent - P2P service wrapper" and the "Generic Agent Services" layer.

Idreos et al propose a similar design approach by developing a P2P platform on top of the multiagent platform DIET.[11,12]

Our proposal differs from the one by Idreos et al on several aspects. Our extension focuses on supporting the distributed discovery and invocation of complex services, whereas Idreos et al focus on supporting the search/subscription to query over textual described documents. Our approach consists of enriching the agent-based paradigm, and thus extending existing platforms, with P2P capabilities, whereas Idreos et al defined a new P2P platform by exploiting a multiagent platform. Finally, differently from Idreos et al, our proof of concept has been implemented in the JADE platform, which is a widely used (probably the most used) platform to develop real MASs.

The RETSINA MAS infrastructure system adopts a multicast protocol both to dynamically discover new agents (peers) and available services.[13] However, in RETSINA, the agents are requested to manage the network and services discovery and management by themselves, whereas we intentionally delegated all these aspects to a specific set of agents so that the burden of managing the topology is not on the normal agents.

Poggi and Tomaiuolo[14] modified JADE to integrate some P2P capabilities from JXTA.[15] Following the FIPA recommendations, authors added an implementation of a global index of resources and services by enabling the JADE DF with the JXTA discovery service. Then, each JADE platform involved in the network is connected to the Agent Peer Group (the one registering all the peers in the network) and to other peer groups (those created by each platform, to manage application specific tasks), reflecting the JXTA network and group organization. This JADE extension was mainly used to develop the RAIS system (Remote Assistant for Information Sharing), a P2P MAS supporting information and document sharing among a community of Internet users.

While the extension was useful, it required modifying the JADE implementation, producing a new version of the platform. Since JXTA is no longer supported by any stable project (its website no longer exists), the new version of the platform can hardly be used now. Our approach proposes a much seamless integration that does not require changing the agent-based paradigm or the platform but rather exploits it to deliver network management and service invocation over a P2P network. Note that we achieve a clear separation between the existing services, which can still be used as they were used before introducing the P2P layer and the new services that enable the hybrid architecture.

6 | CONCLUSIONS

In this paper, we proposed an extension to the JADE platform, which is elegantly achieved by only adding specific agents that provide the appropriate P2P features, in a classic agent oriented design. Our implementation leaves unaltered the JADE platform; thus, it nonintrusively adds capabilities without limiting the existing functionalities. Indeed, it even enables the coexistence of P2P interactions with regular agent-based interactions and is directly usable by new or already existing JADE MASs. The JADE platform was chosen first due to its ample adoption and to its support in the definition and exploitation of agent services, but the proposed protocols may be integrated in other MAS platforms too, for example, in Jason.[16]

We are currently working on evaluating our Network Discovery and Management algorithm to empirically study how it performs in a simulated environment with thousands of platforms. We also plan to make a comparison with other related protocols from the state of the art, for example, with the UpNP protocol[17] or the one proposed in the RETSINA system.[13]

As part of future work, we plan to exploit the extended version of JADE to build distributed applications operating directly in the end-user environment, further demonstrating the suitability of the combined agent- and peer-to-peer paradigms for the development of collaborative applications that assist end-users in case of problems. With this perspective, we will exploit MAS runtime verification techniques (for example, see the works of Briola et al[18,19]) and runtime enforcement techniques (for example, see the works of Riganelli et al[20,21]) to improve the reliability of our layer, and we will further exploit and improve the ontological support to the services modeling and invocation (as, for example, done in the work of Briola et al[22]).

ACKNOWLEDGEMENT

This work has been partially supported by the EU H2020 "Learn" project, which has been funded under the ERC Consolidator Grant 2014 program (ERC Grant Agreement no. 646867).

ORCID

Daniela Briola http://orcid.org/0000-0003-1994-8929
Daniela Micucci http://orcid.org/0000-0003-1261-2234
Leonardo Mariani http://orcid.org/0000-0001-9527-7042

REFERENCES

1. Wang B, Passos L, Xiong Y, Czarnecki K, Zhao H, Zhang W. SmartFixer: fixing software configurations based on dynamic priorities. In: Proceedings of the International Software Product Line Conference (SPLC); 2013; Tokyo, Japan.

2. Ohmann P, Brown DB, Neelakandan N, Linderoth J, Liblit B. Optimizing customized program coverage. In: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE); 2016; Singapore.
3. Liblit B, Aiken A, Zheng AX, Jordan MI. Bug isolation via remote program sampling. In: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI); 2003; San Diego, CA.
4. Androutsellis-Theotokis S, Spinellis D. A survey of peer-to-peer content distribution technologies. *ACM Comput Surv*. 2004;36(4):335-371.
5. Paoli FD, Mariani L. Dependability in peer-to-peer systems. *IEEE Internet Comput*. 2004;8(4):54-61.
6. Wooldridge M, Jennings NR. Intelligent agents: theory and practice. *The Knowl Eng Rev*. 1995;10(2):115-152.
7. Bellifemine FL, Caire G, Greenwood D. *Developing Multi-Agent Systems with JADE*. England, UK: Wiley; 2007.
8. Firefox. https://www.mozilla.org. Accessed July 21, 2018.
9. Fipa Consortium. http://www.fipa.org. Accessed July 21, 2018.
10. Fipa working group for the integration of p2p and agents. http://www.fipa.org/subgroups/P2PNA-WG.html. Accessed November 13, 2017.
11. Idreos S, Koubarakis M, Tryfonopoulos C. P2p-diet: an extensible p2p service that unifies ad-hoc and continuous querying in super-peer networks. In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD '04); 2004; Paris, France. http://doi.acm.org/10.1145/1007568.1007704
12. Idreos S, Koubarakis M. P2P-DIET: ad-hoc and continuous queries in peer-to-peer networks using mobile agents. Paper presented at: Third Hellenic Conference on Artificial Intelligence (SETN); 2004; Samos, Greece. https://doi.org/10.1007/978-3-540-24674-9_4
13. Sycara K, Paolucci M, Van Velsen M, Giampapa J. The RETSINA MAS infrastructure. *Auton Agents Multi-Agent Syst*. 2003;7:29-48. https://doi.org/10.1023/A:1024172719965
14. Poggi A, Tomaiuolo M. Integrating peer-to-peer and multi-agent technologies for the realization of content sharing applications. *Information Retrieval and Mining in Distributed Environments*. Berlin, Germany: Springer; 2011:93-107.
15. JXTA. https://en.wikipedia.org/wiki/JXTA. Accessed December 13, 2017.
16. Bordini RH, Hübner JF, Wooldridge M. *Programming Multi-Agent Systems in AgentSpeak Using Jason*. Hoboken, NJ: John Wiley & Sons; 2007.
17. The UPnP protocol. https://openconnectivity.org/developer/specifications/upnp-resources/presentations-whitepapers. Accessed November 13, 2017.
18. Briola D, Mascardi V, Ancona D. Distributed runtime verification of jade and jason multiagent systems with prolog. In: Proceedings of 29th Italian Conference on Computational Logic (CILC); 2014; Turin, Italy.
19. Mascardi V, Briola D, Ancona D. On the expressiveness of attribute global types: the formalization of a real multiagent system protocol. Paper presented at: International Conference of the Italian Association for Artificial Intelligence; 2013; Turin, Italy. https://doi.org/doi:10.1007/978-3-319-03524-6_26
20. Riganelli O, Micucci D, Mariani L. Policy enforcement with proactive libraries. In: Proceedings of the 12th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'17); 2017; Buenos Aires, Argentina.
21. Riganelli O, Micucci D, Mariani L, Falcone Y. Verifying policy enforcers. In: Proceedings of the 17th International Conference on Runtime Verification (RV'17); 2017; Seattle, WA.
22. Briola D, Deufemia V, Mascardi V, Paolino L. Agent-oriented and ontology-driven digital libraries: the IndianaMAS experience. *Softw: Pract Exp*. 2017;47(11):1773-1799.

How to cite this article: Briola D, Micucci D, Mariani L. A platform for P2P agent-based collaborative applications. *Softw: Pract Exper*. 2018;1–10. https://doi.org/10.1002/spe.2657