Gajim For SPADE

Aida Ben Aziza[1][2]

Higher Institute of Applied Science and Technology of Sousse.[1]
aidabenaziza96@gmail.com[2]
https://orcid.org/0000-0002-8458-1783

Abstract. Although agent-based systems have been in existence for several years, progress in terms of actual applications or their integration into industry has not yet reached the expected levels. Over the past two decades, multiple agent platforms have emerged to streamline the development of multi-agent systems. Some of these platforms have been developed for general purposes, while others have been targeted to specific areas. However, the absence of standards and the complexity associated with supporting such systems, among other challenges, have hindered their widespread use. 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 fiexible solution to current problems.Furthermore, the document presents SPADE the middleware, whith the collaboration of Gajim the instant messaging client for the XMPP protocol. Finally, a case study is provided to illustrate how SPADE with Gajim is capable of addressing these challenges.

Keywords: Gajim, SPADE , Multi-agent systems,middleware.

1 Introduction

Distributed artificial intelligence (DAI) is a subfield of Artificial Intelligence [Jennings et al., 1998] that has gained considerable importance due to its ability to solve complex real-world problems. The primary focus of research in the field of distributed artificial intelligence has included three different areas. These are parallel AI, Distributed problem solving(DPS) and Multi-agent systems (MAS). Parallel AI primarily refers to methodologies used to facilitate classical AI [Jain and Jain, 1997] [Teodorescu et al., 1998] techniques when applied to distributed hardware architectures like multiprocessor or cluster based computing. The main aim of parallel AI is to increase the speed of operation and to work on parallel threads in order to arrive at a global solution for a particular problem. Distributed problem solving is similar to parallel AI and considers how a problem can be solved by sharing the resources and knowledge between large number of cooperating modules known as Computing entity. In the resolution of distributed problems, communication between IT entities, the amount of shared information is predetermined and incorporated into the design of the IT entity.

Distributed problem solving is inflexible because of integrated strategies and therefore provides little or no flexibility.
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.
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 multiagent 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 SPADE1 software [1], a middleware for developing and executing multi-agent systems, written in Python with the collaboration of the instant messaging client for the XMPP protocol Gajim.
According to this, the paper presents three main contributions: the first one is a study of the current situation of agent platforms and the second one is a analysis of some relevant open issues for the development of modern and future multiagent systems ; and the third one is the presentation of SPADE with Gajim .

2 Theoretical Background

2.1 SPADE

The idea of an XMPP-based agent platform appeared one night at 4 A.M [Palanca et al., 2020]. when, studying the features of the Jabber architecture, we found out great similarities with the ones of a FIPA-compliant agent platform. The XMPP protocol offered a great architecture for agents to communicate in a structured way and solved many issues present when designing a platform, such as authenticating the users (the agents), provide a directory or create communication channels.
We started to work on our first prototype of this Jabber-powered platform and within a week we had a small working proof of concept by the name of Fipper which eventually allowed for dumb agents to connect and communicate through a common XMPP server.
Since that day, things have changed a bit. The small proof of concept evolved

into a full-featured FIPA platform, and the new SPADE name was coined. As usual, we later had to find the meaning of the beautiful acronym. We came up with Smart Python multi-Agent Development Environment, which sounded both good and geek enough.
The years passed, and everything evolved except the platform. Python reached version 3, which came with lots of interesting changes and improvements. We also became better programmers (just because of the grounding and the experience that the years give), we met the PEP8 and the Clean Code principles and they opened our eyes to a new world. That's why in 2017 SPADE was fully rewritten in Python 3.6, using asyncio and strictly following PEP8 and Clean Code principles.
We hope you like this software and have as much fun using it as we had writing it. Of course we also hope that it may become useful, but that is a secondary matter.
The Agent Model is basically composed of a connection mechanism to the platform, a message dispatcher, and a set of different behaviours that the dispatcher gives the messages to. Every agent needs an identifier called Jabber ID a.k.a. JID and a valid password to establish a connection with the XMPP server [Palanca et al., 2020].
The JID (composed by a username, an @, and a server domain) will be the name that identifies an agent in the platform, e.g. myagent@myprovider.com.

2.2 Connection to the platform

Communications in SPADE are handled internally by means of the XMPP protocol. This protocol has a mechanism to register and authenticate users against an XMPP server [Palanca et al., 2020].
After a succesful register, each agent holds an open and persistent XMPP stream of communications with the platform. This process is automatically triggered as part of the agent registration process.

2.3 The message dispatcher

Each SPADE agent has an internal message dispatcher component. This message dispatcher acts as a mailman: when a message for the agent arrives, it places it in the correct "mailbox" (more about that later); and when the agent needs to send a message, the message dispatcher does the job, putting it in the communication stream. The message dispatching is done automatically by the SPADE agent library whenever a new message arrives or is to be sent [Palanca et al., 2020].

2.4 The behaviours

An agent can run serveral behaviours simultaneously. A behaviour is a task that an agent can execute using repeating patterns. SPADE provides some predefined behaviour types: Cyclic, One-Shot, Periodic, Time-Out and Finite State

Machine [Palanca et al., 2020]. Those behaviour types help to implement the different tasks that an agent can perform. The kind of behaviours supported by a SPADE agent are the following:

- Cyclic and Periodic behaviours are useful for performing repetitive tasks.
- One-Shot and Time-Out behaviours can be used to perform casual tasks.
- The Finite State Machine allows more complex behaviours to be built.

Every agent can have as many behaviours as desired. When a message arrives to the agent, the message dispatcher redirects it to the correct behaviour queue. A behaviour has a message template attached to it. Therefore, the message dispatcher uses this template to determine which behaviour the message is for, by matching it with the correct template. A behaviour can thus select what kind of messages it wants to receive by using templates [Palanca et al., 2020].

2.5 Gajim

To show the feasibility of the SPADE, we implemented a prototype to support SPADE based on the Gajim [1] XMMP instant messaging platform.
Gajim is an instant messaging client for the XMPP protocol which uses the GTK toolkit. The name Gajim is a recursive acronym for Gajim's a jabber instant messenger.

2.6 Multi-Agent Systems

An MAS is a system consisting of a large number of agents interacting autonomously in a dynamic environment [Cavalcante et al., 2012]. The roles and actions of the agents must be clearly defined to solve one or more defined problems.
An agent is a real or virtual entity that can act autonomously and flexibly to achieve its goals in its environment [Baig, 2012]. An agent can be characterized by six properties:

- **Autonomy:**An agent should have the ability to act and cooperate without human intervention;
- **Reactivity:**An agent should be able to perceive the environment and react to changes, for instance in terms of the modification of the defined objectives or the resources available;
- **Proactive:** An agent must be able to identify the purpose of a directed behavior by taking the initiative;
- **Sociability and communication:**An agent should have an ability to interact with other agents. The most common agent communication protocol is Agent Communication Language (ACL);
- **Learning:**An agent should be able to memorize experiences and adapt its behavior accordingly;
- **Mobility:** An agent should be able to move from one machine to another.

[1] `http://www.gajim.org`

3 State of the art

In recent years,[Palanca et al., 2020] multi-agent systems (MAS) have evolved considerably in several respects along various lines of research, but this evolution has not yet reached full consolidation. A key aspect of this consolidation is the support provided by the MAS platforms, which are the software (or middleware) components on which MAS is built and executed. In many ways, the architecture, services, tools, etc., provided by a particular platform determines the capabilities of the MAS which can be developed and executed on this platform.and, in turn, this affects the types of problems that these MAS can solve [Palanca et al., 2020].

In the field of research on MAS platforms [Palanca et al., 2020], some of the latest significant developments include, on the one hand, the evolution of some traditional, well-known and versatile 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[2] (JAVA Agent Development Framework) [Bellifemine et al., 1999], which last version is from 2017. JADE has probably been the most widely used tool to develop multiagent systems in the last two decades [Bergenti et al., 2020]. JADE implements a fully-compliant FIPA[3] platform [Poslad et al., 2000] 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[4] (Workflows and Agents Development Environment) [Caire et al., 2008] 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 [Bergenti, 2014] (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 Envi-

[2] `http://jade.tilab.com`

[3] `http://www.fipa.org`

[4] `https://jade.tilab.com/wadeproject/`

ronment) [Bigus et al., 2002] 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. Mad- Kit6 (Multi-agent development Kit) [Gutknecht and Ferber, 2000] 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. JANUS7 [Galland et al., 2010] 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+ [Baldoni et al., 2018] platform is founded on supporting the concept of agent organization. It integrates the Jason language [Bordini and Hübner, 2005] (a BDI agent language based on the AgentSpeak specification), CArtAgO [Ricci et al., 2006] as a platform for implementing artifacts to define the environment of the agents, and MOISE [Hannoun et al., 2000] as the organizational model to structure the agents in the multi-agent system.

As stated above, there have also been recent developments inMASplatforms 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),Big Data, Intrusion Detection System(IDS)

This part gives a survey of literature work done by other researchers. I've learned some existing techniques from their research work, few of them are discussed below.

The authors of [Zhai Shuang-Can and Weiming, 2014] proposed a collaborative multi-agent IDS (CIDS) model based on the backpropagation neural network. This model adopts the collaborative detection and distributed response modes, where the agents were relatively independent. The advantages of the proposed system are: reduction of the mobile data process, load equalization, good error tolerance and effective detection of collaborative intrusions. The system was tested on the KDD Cup 99 dataset, and the results show that the system could improve detection accuracy efficiency and significantly reduce center console workload.

[Toumi et al., 2015], the authors have developed a collaborative system based on Hybrid-IDS and mobile agents in Cloud computing, to define a dynamic context which enables the detection of new and known attacks in this environment.

[Wankhade and Chandrasekaran, 2016] propose a method by SVM – ACO this method is characterized by a higher detection rate, faster operating time, the system is very efficient, KDD cup 99 dataset and a poor performance of SVM and high calculation.

[Eesa et al., 2017]7 propose a detection method by Decision Tree based on mobile agents this method is characterized by good performance when using 5 features, the U2R attack offers better performance with 41 features and it uses a set of data KDD cup 99

[Teng et al., 2017] propose a method by Decision tree and SVM this method is characterized by good accuracy and detection efficiency, KDD cup 99 dataset and poor performance of SVM and high calculation.

[Li et al., 2018] propose a detection method based also on mobile agents this method is characterized by No single point of failure, limited scalability, good performance, solve the bottleneck problem and it uses KDD cup 99 dataset.

[Achbarou et al., 2018]propose a new flexible, distributed and adaptive model, based on intelligent agent technology and DIDS, called MAS-DIDS. This model consists of a group of reactive, autonomous and cooperating agents that interact with each other to reduce the workload of an IDS, making the system more efficient and secure. The experimental results show that the MAS-DIDS can increase the intrusion detection rate and decrease the false positive rate compared to the use of a centralized IDS, meaning that our proposal is realistic and valuable for detecting both known and unknown attacks in a complex and dynamic environment such as cloud computing.

[Idhammad et al., 2018] present a distributed Machine Learning based intrusion detection system for Cloud environments. The proposed system is designed to be inserted in the Cloud side by side with the edge network components of the Cloud provider. This allows to intercept incoming network traffic to the edge network routers of the physical layer. A time-based sliding window algorithm is used to preprocess the captured network traffic on each Cloud router and pass it to an anomaly detection module using Naive Bayes classifier. A set of commodity server nodes based on Hadoop and MapReduce are available for each anomaly detection module to use when the network congestion increases. For each time window, the anomaly network traffic data on each router side are synchronized to a central storage server. Next, an ensemble learning classifiers based on the Random Forest is used to perform a final multi-class classification step in order to detect the type of each attack.

[Dahiya and Srivastava, 2018] propose a framework in which a feature reduction algorithm is used for reducing the less important features and then applied

the supervised data mining techniques on UNSW-NB15network dataset for fast, efficient and accurate detection of intrusion in the Netflow records using Spark. In this paper, we have used two feature reduction algorithms, namely, Canonical Correlation Analysis (CCA) and Linear Discriminant Analysis (LDA) and seven well known classification algorithms.
[Gao et al., 2019] offers a distributed DDoS network intrusion detection system based on big data technology. The proposed detection system consists of two main components: a real-time network traffic collection module and a network traffic detection module. This system gives good results in accuracy and false positive rate (FPR)

[Khan et al., 2019]propose a scalable and hybrid IDS, which is based on Spark ML and the convolutional-LSTM (Conv-LSTM) network. This IDS is a two-stage ID system: the first stage employs the anomaly detection module, which is based on Spark ML. The second stage acts as a misuse detection module, which is based on the Conv-LSTM network, such that both global and local latent threat signatures can be addressed. Evaluations of several baseline models in the ISCX-UNB dataset show that our hybrid IDS can identify network misuses accurately in 97.29% of cases and outperforms state-of-the-art approaches during 10-fold cross-validation tests.

[Zhong et al., 2020] propose a Big Data based Hierarchical Deep Learning System (BDHDLS). BDHDLS utilizes behavioral features and content features to understand both network traffic characteristics and information stored in the payload. Each deep learning model in the BDHDLS concentrates its efforts to learn the unique data distribution in one cluster. This strategy can increase the detection rate of intrusive attacks as compared to the previous single learning model approaches. Based on parallel training strategy and big data techniques, the model construction time of BDHDLS is reduced substantially when multiple machines are deployed.

[Manan et al., 2019] propose a Distributed intrusion detection scheme for next generation networks and presents a study of SIP protocol and discovers the critical security vulnerabilities in the course of registration phase. We focused on DDoS attacks on IMS server using SIP particularly with REGISTER message and proposed a scheme based on multi agent systems for intrusion detection which takes the advantage of the distributed paradigm to implement an efficient distributed system, as well as the integration of existing techniques, i.e., the well-known IDS SNORT.

[Riyad et al., 2019] propose a method based on mobile agents this method is characterized by high and better scalability, no single point of failure, high performance, efficient system and real traffic;

Another important work by Laqtib et al 2019 [Laqtib et al., 2019], considered hierarchical collaborative IDS using deep learning techniques under MANET, in which each node has an IDS agent running. The authors presented well-known deep learning models, namely: CNN, Inception-CNN, Bi-LSTM and GRU. Next, they made a systematic comparison of CNN and RNN on the DL-based IDS. Experimental results indicated good performance for all 4 models. But on TPR, the base CNN and the launch CNN failed, the Bi-LSTM model also performed worse on accuracy rates than other models.

[Mezghani and Ktata, 2020] propose a distributed intelligent agent based intrusion detection system using deep learning algorithms DIAFIDS-DLA. The main idea is to distribute the tasks on intelligent and autonomous agents which, with their interactions, allow the detection process to unfold. This intrusion detection system is composed of four main agents The detection process begins with the preprocessing agent to present standardized and adapted data for learning. Then the data is loaded into the networks through the CNN and RNN classifier agents for learning to generate prediction models that will be useful for the analytical agent for the final detection. The final prediction values are 0.99% for a loss rate of 0.098%.

4 OPEN ISSUES IN CURRENT AGENT PLATFORMS AND THE SOLUTION

In open multi-agent systems, agents are mobile and may leave or enter the system. This dynamicity results in two closely related agent communication problems, namely, efficient message passing and service agent discovery.

In open agent systems [Jang et al., 2004], new agents may be created and agents may move from one computer node to another. With the growth of computational power and network bandwidth, large-scale open agent systems are a promising technology to support coordinated computing. For example, agent mobility can facilitate efficient collaboration with agents on a particular node. A number of multi-agent systems, such as EMAF [Jang et al., 2004], JADE [Bellifemine et al., 1999], InfoSleuth [Bayardo Jr et al., 1997], and OAA [Cohen et al., 1994], support open agent systems. However, before the vision of scalable open agent systems can be realized, two closely related problems must be addressed:

Message Passing Problem: In mobile agent systems, efficiently sending messages to an agent is not simple because they move continuously from one agent platform to another. For example, the original agent platform on which an agent is created should manage the location information about the agent.
However, doing so not only increases the message passing overhead, but it

slows down the agent's migration: before migrating, the agent's current host platform must inform the the original platform of the move and may wait for an acknowledgement before enabling the agent.

Service Agent Discovery Problem: In an open agent system, the mail addresses or names of all agents are not globally known. Thus an agent may not have the addresses of other agents with whom it needs to communicate. To address this difficulty, middle agent services, such as brokering and matchmaking services [Decker et al., 1997], need to be supported. However, current middle agent systems suffer from two problems: lack of expressiveness–not all search queries can be expressed using the middle agent supported primitives; and incomplete information–a middle agent does not possess the necessary information to answer a user query

We address the message passing problem for agents in part by providing a richer name structure: the names of agents include information about their current location. When an agent moves, the location information in its name is updated by the platform that currently hosts the agent. When the new name is transmitted, the location information is used by other platforms to find the current location of that agent if it is the receiver of a message. We address the service agent discovery problem in large-scale open agent systems by allowing client agents to send search objects to be executed in the middle agent address space. By allowing agents to send their own search algorithms, this mitigates both the lack of expressiveness and incomplete information.

This kind of proprietary support is not adequate for the next generation of MAS, which are likely to be open and massive,[Palanca et al., 2020] 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,[Palanca et al., 2020] 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 wellknown 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 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 thisway, 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 [Palanca et al., 2020], 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 [Palanca et al., 2020], 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 [Sanchis et al., 2014] 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 [Palanca et al., 2020], 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.
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.

To sum up, we need an Efficient Agent Communication in Multi-agent Systems, this feature we can support it by the use of Gajim[5] , so efficient message passing is facilitated .To facilitate service agent discovery, middle agents support application agent-oriented matchmaking and brokering services.
Gajim supports third-party plugins. Examples include:
Gajim-OMEMO, adding support for OMEMO[6]

5 Gajim for SPADE

There are many, many XMPP clients for you to choose from. To get you started. For me i choose Gajim
first of all you need to create an account
As with email, you need an account with a service provider. Use a public provider from the curated list of XMPP Providers or be your own provider by hosting a server yourself.
XMPP Providers takes various aspects into consideration to recommend providers. There are several other (uncurated) lists of providers[7]
In this work i use jabber[8]
Then you log in your account.
After all of this you create your first agent absolutely with the power of SPADE. It's time for us to build our first SPADE agent. We'll assume that we have a registered user in an XMPP server with a jid and a password. The jid contains the agent's name (before the @) and the DNS or IP of the XMPP server (after the @). But remember! You should have your own jid and password in an XMPP server running in your own computer or in the Internet. In this example we will assume that our jid is your-jid@your-xmpp-server and the password is your-password.
A basic SPADE agent is really a python script that imports the spade module.

[5] `https://dev.gajim.org/`
[6] `https://dev.gajim.org/gajim/gajim-plugins/-/wikis/OmemoGajimPlugin`
[7] `https://xmpp.org/getting-started/`
[8] `https://list.jabber.at/`

Agent communications

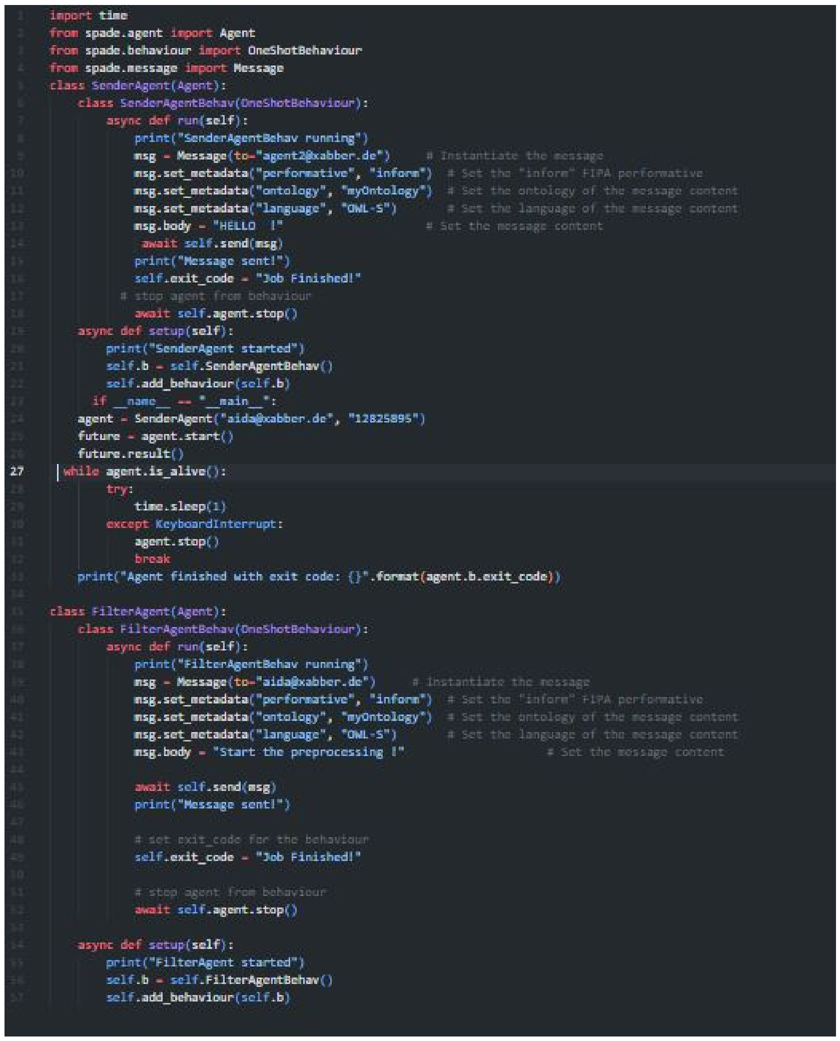
Using templates

Templates is the method used by SPADE to dispatch received messages to the behaviour that is waiting for that message. When adding a behaviour you can set a template for that behaviour, which allows the agent to deliver a message received by the agent to that registered behaviour. A Template instance has the same attributes of a Message and all the attributes defined in the template must be equal in the message for this to match.
The attributes that can be set in a template are:

to :the jid string of the receiver of the message.
sender :the jid string of the sender of the message.
body :the body of the message.
thread :the thread id of the conversation.
metadata :a (key, value) dictionary of strings to define metadata of the message. This is useful, for example, to include FIPA attributes like ontology, performative, language, etc.

Sending and Receiving Messages

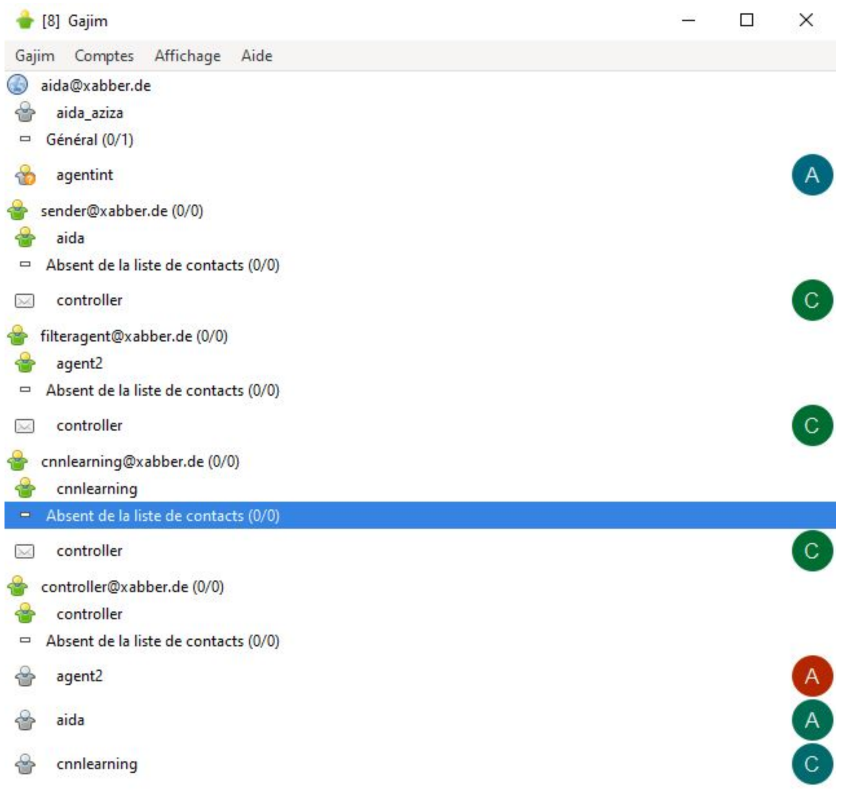
As you know, messages are the basis of every MAS. They are the centre of the whole "computing as interaction" paradigm in which MAS are based. So it is very important to understand which facilities are present in SPADE to work with agent messages.
First and foremost, threre is a Message class. This class is spade.message.Message and you can instantiate it to create new messages to work with. The class provides a method to introduce metadata into messages, this is useful for using the fields present in standard FIPA-ACL Messages. When a message is ready to be sent, it can be passed on to the send() method of the behaviour, which will trigger the internal communication process to actually send it to its destination. Note that the send function is an async coroutine, so it needs to be called with an await statement.
Here is a explaining example 1:



```python
import time
from spade.agent import Agent
from spade.behaviour import OneShotBehaviour
from spade.message import Message
class SenderAgent(Agent):
    class SenderAgentBehav(OneShotBehaviour):
        async def run(self):
            print("SenderAgentBehav running")
            msg = Message(to="agent2@xabber.de")     # Instantiate the message
            msg.set_metadata("performative", "inform")  # Set the "inform" FIPA performative
            msg.set_metadata("ontology", "myOntology")  # Set the ontology of the message content
            msg.set_metadata("language", "OWL-S")       # Set the language of the message content
            msg.body = "HELLO  !"                       # Set the message content
             await self.send(msg)
            print("Message sent!")
            self.exit_code = "Job Finished!"
          # stop agent from behaviour
            await self.agent.stop()
    async def setup(self):
        print("SenderAgent started")
        self.b = self.SenderAgentBehav()
        self.add_behaviour(self.b)
      if __name__ == "__main__":
    agent = SenderAgent("aida@xabber.de", "12825895")
    future = agent.start()
    future.result()
  while agent.is_alive():
        try:
            time.sleep(1)
        except KeyboardInterrupt:
            agent.stop()
            break
    print("Agent finished with exit code: {}".format(agent.b.exit_code))

class FilterAgent(Agent):
    class FilterAgentBehav(OneShotBehaviour):
        async def run(self):
            print("FilterAgentBehav running")
            msg = Message(to="aida@xabber.de")     # Instantiate the message
            msg.set_metadata("performative", "inform")  # Set the "inform" FIPA performative
            msg.set_metadata("ontology", "myOntology")  # Set the ontology of the message content
            msg.set_metadata("language", "OWL-S")       # Set the language of the message content
            msg.body = "Start the preprocessing !"                 # Set the message content

            await self.send(msg)
            print("Message sent!")

            # set exit_code for the behaviour
            self.exit_code = "Job Finished!"

            # stop agent from behaviour
            await self.agent.stop()

    async def setup(self):
        print("FilterAgent started")
        self.b = self.FilterAgentBehav()
        self.add_behaviour(self.b)
```

Fig. 1: Sending and Receiving Messages example

When you run this code you find the flexibility and the interaction between the agents in gajim especially the rapidity of receiving and sending messages 2 .


[8] Gajim

Gajim Comptes Affichage Aide

aida@xabber.de
aida_aziza
Général (0/1)
agentint A
sender@xabber.de (0/0)
aida
Absent de la liste de contacts (0/0)
controller C
filteragent@xabber.de (0/0)
agent2
Absent de la liste de contacts (0/0)
controller C
cnnlearning@xabber.de (0/0)
cnnlearning
Absent de la liste de contacts (0/0)
controller C
controller@xabber.de (0/0)
controller
Absent de la liste de contacts (0/0)
agent2 A
aida A
cnnlearning C

Fig. 2: Communication between agents in Gajim

6 Conclusion

In open multi-agent systems, agents are mobile and may leave or enter the system. This dynamicity results in two closely related agent communication problems, namely, efficient message passing and service agent discovery.
we find that gajim it can be one of the best solution of instant messaging client for the XMPP protocol we used to create a multi agent system which uses the GTK-footnote`https://www.openhub.net/p/gtk/analyses/latest/languages_summary` toolkit.

References

Achbarou et al., 2018. Achbarou, O., El Kiram, M. A., Bourkoukou, O., and Elbouanani, S. (2018). A new distributed intrusion detection system based on multi-agent system for cloud environment. *International Journal of Communication Networks and Information Security*, 10(3):526.

Baig, 2012. Baig, Z. A. (2012). Multi-agent systems for protecting critical infrastructures: A survey. *Journal of Network and Computer Applications*, 35(3):1151–1161.

Baldoni et al., 2018. Baldoni, M., Baroglio, C., Capuzzimati, F., and Micalizio, R. (2018). Commitment-based agent interaction in jacamo+. *Fundamenta Informaticae*, 159(1-2):1–33.

Bayardo Jr et al., 1997. Bayardo Jr, R. J., Bohrer, W., Brice, R., Cichocki, A., Fowler, J., Helal, A., Kashyap, V., Ksiezyk, T., Martin, G., Nodine, M., et al. (1997). Infosleuth: Agent-based semantic integration of information in open and dynamic environments. *Acm Sigmod Record*, 26(2):195–206.

Bellifemine et al., 1999. Bellifemine, F., Poggi, A., and Rimassa, G. (1999). Jade–a fipa-compliant agent framework. In *Proceedings of PAAM*, volume 99, page 33. London.

Bergenti, 2014. Bergenti, F. (2014). An introduction to the jadel programming language. In *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*, pages 974–978. IEEE.

Bergenti et al., 2020. Bergenti, F., Caire, G., Monica, S., and Poggi, A. (2020). The first twenty years of agent-based software development with jade. *Autonomous Agents and Multi-Agent Systems*, 34(2):1–19.

Bigus et al., 2002. Bigus, J. P., Schlosnagle, D. A., Pilgrim, J. R., Mills III, W. N., and Diao, Y. (2002). Able: A toolkit for building multiagent autonomic systems. *IBM Systems Journal*, 41(3):350–371.

Bordini and Hübner, 2005. Bordini, R. H. and Hübner, J. F. (2005). Bdi agent programming in agentspeak using jason. In *International workshop on computational logic in multi-agent systems*, pages 143–164. Springer.

Caire et al., 2008. Caire, G., Gotta, D., and Banzi, M. (2008). Wade: a software platform to develop mission critical applications exploiting agents and workflows. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*, pages 29–36.

Cavalcante et al., 2012. Cavalcante, R. C., Bittencourt, I. I., da Silva, A. P., Silva, M., Costa, E., and Santos, R. (2012). A survey of security in multi-agent systems. *Expert Systems with Applications*, 39(5):4835–4846.

Cohen et al., 1994. Cohen, P. R., Cheyer, A., Wang, M., and Baeg, S. C. (1994). An open agent architecture. In *AAAI Spring Symposium*, volume 1.

Dahiya and Srivastava, 2018. Dahiya, P. and Srivastava, D. K. (2018). Network intrusion detection in big dataset using spark. *Procedia computer science*, 132:253–262.

Decker et al., 1997. Decker, K., Sycara, K., and Williamson, M. (1997). Middle-agents for the internet. In *IJCAI (1)*, pages 578–583.

Eesa et al., 2017. Eesa, A. S., Abdulazeez, A. M., and Orman, Z. (2017). A dids based on the combination of cuttlefish algorithm and decision tree. *Science Journal of University of Zakho*, 5(4):313–318.

Galland et al., 2010. Galland, S., Gaud, N., Rodriguez, S., and Hilaire, V. (2010). Janus: Another yet general-purpose multiagent platform. In *Seventh AOSE Technical Forum, Paris*.

Gao et al., 2019. Gao, Y., Wu, H., Song, B., Jin, Y., Luo, X., and Zeng, X. (2019). A distributed network intrusion detection system for distributed denial of service attacks in vehicular ad hoc network. *IEEE Access*, 7:154560–154571.

Gutknecht and Ferber, 2000. Gutknecht, O. and Ferber, J. (2000). The m ad k it agent platform architecture. In *Workshop on infrastructure for scalable multi-agent systems at the international conference on autonomous agents*, pages 48–55. Springer.

Hannoun et al., 2000. Hannoun, M., Boissier, O., Sichman, J. S., and Sayettat, C. (2000). Moise: An organizational model for multi-agent systems. In *Advances in Artificial Intelligence*, pages 156–165. Springer.

Idhammad et al., 2018. Idhammad, M., Afdel, K., and Belouch, M. (2018). Distributed intrusion detection system for cloud environments based on data mining techniques. *Procedia Computer Science*, 127:35–41.

Jain and Jain, 1997. Jain, L. C. and Jain, R. K. (1997). *Hybrid intelligent engineering systems*, volume 11. World Scientific.

Jang et al., 2004. Jang, M.-W., Ahmed, A., and Agha, G. (2004). Efficient agent communication in multi-agent systems. In *International Workshop on Software Engineering for Large-Scale Multi-agent Systems*, pages 236–253. Springer.

Jennings et al., 1998. Jennings, N. R., Sycara, K., and Wooldridge, M. (1998). A roadmap of agent research and development. *Autonomous agents and multi-agent systems*, 1(1):7–38.

Khan et al., 2019. Khan, M. A., Karim, M. R., and Kim, Y. (2019). A scalable and hybrid intrusion detection system based on the convolutional-lstm network. *Symmetry*, 11(4):583.

Laqtib et al., 2019. Laqtib, S., Yassini, K. E., and Hasnaoui, M. L. (2019). A deep learning methods for intrusion detection systems based machine learning in manet. In *Proceedings of the 4th International Conference on Smart City Applications*, pages 1–8.

Li et al., 2018. Li, Y., Du, M., and Xu, J. (2018). A new distributed intrusion detection method based on immune mobile agent. In *2018 Sixth International Conference on Advanced Cloud and Big Data (CBD)*, pages 215–219. IEEE.

Manan et al., 2019. Manan, J., Ahmed, A., Ullah, I., Merghem-Boulahia, L., and Gaïti, D. (2019). Distributed intrusion detection scheme for next generation networks. *Journal of Network and Computer Applications*, 147:102422.

Mezghani and Ktata, 2020. Mezghani, S. and Ktata, F. B. (2020). A distributed intelligent agent based intrusion detection system using deep learning algorithms. In *Proceedings of the 2nd International Conference on Digital Tools & Uses Congress*, pages 1–4.

Palanca et al., 2020. Palanca, J., Terrasa, A., Julian, V., and Carrascosa, C. (2020). Spade 3: Supporting the new generation of multi-agent systems. *IEEE Access*, 8:182537–182549.

Poslad et al., 2000. Poslad, S., Buckle, P., and Hadingham, R. (2000). The fipa-os agent platform: Open source for open standards. In *proceedings of the 5th international conference and exhibition on the practical application of intelligent agents and multi-agents*, volume 355, page 0.

Ricci et al., 2006. Ricci, A., Viroli, M., and Omicini, A. (2006). Cartago: A framework for prototyping artifact-based environments in mas. In *International Workshop on Environments for Multi-Agent Systems*, pages 67–86. Springer.

Riyad et al., 2019. Riyad, A., Ahmed, M. I., and Khan, R. R. (2019). An adaptive distributed intrusion detection system architecture using multi agents. *International Journal of Electrical and Computer Engineering*, 9(6):4951.

Sanchis et al., 2014. Sanchis, A., Julián, V., Corchado, J. M., Billhardt, H., and Carrascosa, C. (2014). Using natural interfaces for human-agent immersion. In *International Conference on Practical Applications of Agents and Multi-Agent Systems*, pages 358–367. Springer.

Teng et al., 2017. Teng, S., Wu, N., Zhu, H., Teng, L., and Zhang, W. (2017). Svm-dt-based adaptive and collaborative intrusion detection. *IEEE/CAA Journal of Automatica Sinica*, 5(1):108–118.

Teodorescu et al., 1998. Teodorescu, H.-N. L., Kandel, A., and Jain, L. C. (1998). *Fuzzy and neuro-fuzzy systems in medicine*, volume 2. CRC Press.

Toumi et al., 2015. Toumi, H., Talea, A., Marzak, B., Eddaoui, A., and Talea, M. (2015). Cooperative trust framework for cloud computing based on mobile agents. *International Journal of Communication Networks and Information Security*, 7(2):106.

Wankhade and Chandrasekaran, 2016. Wankhade, A. and Chandrasekaran, K. (2016). Distributed-intrusion detection system using combination of ant colony optimization (aco) and support vector machine (svm). In *2016 International Conference on Micro-Electronics and Telecommunication Engineering (ICMETE)*, pages 646–651. IEEE.

Zhai Shuang-Can and Weiming, 2014. Zhai Shuang-Can, H. C.-j. and Weiming, Z. (2014). Multi-agent distributed intrusion detection system model based on bp neural network. IEEE.

Zhong et al., 2020. Zhong, W., Yu, N., and Ai, C. (2020). Applying big data based deep learning system to intrusion detection. *Big Data Mining and Analytics*, 3(3):181–195.