UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

Departament de Sistemes Informàtics i Computació

Universitat Politècnica de València

Plataforma per a la interoperabilitat de sistemes multi-agent

Treball Final de Màster

Màster en Intel·ligència Artificial, Reconeixement de Formes i Imatge Digital

Autor: Sancho i Martínez, Joan

Tutors: Julián Inglada, Vicente Javier

Costa, Ângelo

2016 - 2017

Agraïments

Als meus tutors del projecte Vicente Javier Julián Inglada i Ângelo Costa, per guiar-me i ajudar-me en aquest projecte.

A Javier Palanca Cámara, per ajudar-me a comprendre millor la plataforma de SPADE.

A la meua família, per tindre molta paciència amb mi en la realització d'aquest projecte.

Resum

Actualment existeixen moltes plataformes per a dissenyar sobre elles sistemes d'agents intel·ligents, encara que els mateixos agents només interactuen entre ells de forma aïllada a altres sistemes externs. Amb aquest projecte es vol establir un canal de comunicació que es puga aplicar a qualsevol sistema multi-agent per a poder comunicar-se amb altres de diferents.

L'objectiu del projecte consisteix en desenvolupar un mòdul de comunicació en la plataforma de sistemes multi-agent SPADE, per a permetre la interacció amb altres sistemes multi-agent diferents. El mòdul es compon entre un agent intel·ligent i un servei web implementat amb Flask. El protocol de comunicació que s'utilitza es basa en l'arquitectura REST, que permet la interoperabilitat entre sistemes distribuïts en una xarxa web.

Per a aconseguir aquest objectiu principal, es realitzen diferents anàlisis per a concloure el fet d'utilitzar la plataforma multi-agent i el framework web mencionats anteriorment i es fan diverses proves amb diferents sistemes distribuïts per a comprovar el funcionament del mòdul implementat.

Paraules clau: Sistemes multi-agent, Plataformes d'agents, SPADE, Python, REST, Flask.

Resumen

Actualmente existen muchas plataformas para diseñar sobre ellas sistemas de agentes inteligentes, aunque los mismos agentes sólo interactúan entre ellos de forma aislada a otros sistemas externos. Con este proyecto se quiere establecer un canal de comunicación que se pueda aplicar a cualquier sistema multi-agente para poder comunicarse con otras distintas.

El objetivo del proyecto consiste en desarrollar un módulo de comunicación en la plataforma de sistemas multi-agente SPADE, para permitir la interacción con otros sistemas multi-agente distintos. El módulo se compone entre un agente inteligente y un servicio web implementado en Flask. El protocolo de comunicación que se utiliza se basa en la arquitectura REST, que permite la interoperabilidad entre sistemas distribuidos en una red web.

Para conseguir este objetivo principal, se realizan distintos análisis para concluir el hecho de utilizar la plataforma multi-agente y el framework web mencionados anteriormente y se hacen diversas pruebas con distintos sistemas distribuidos para comprobar el funcionamiento del módulo implementado.

Palabras clave: Sistemas multi-agente, Plataformas de agentes, SPADE, Python, REST, Flask.

Abstract

Currently there are many platforms for designing agent systems on them, although the same agents only interact with each other in isolation to other external systems. With this project we want to make a communication channel that can be applied to any multi-agent system to be able to communicate with different ones.

The objective of the project consist to develop a communication module in the SPADE multi-agent system, to allow interactions with other multi-agent systems. The module is composed by an intelligent agent and a web service implemented with Flask. The communication protocol used is based on the REST architecture, that allows the interoperability between distributed systems in a web network.

In order to achieve this main objective, different analyses are made to conclude the use of the multi-agent platform and the web framework mentioned above and different tests are done with different distributed systems to verify the performance of the implemented module.

Keywords: Multi-Agent Systems, Agent platforms, SPADE, Python, REST, Flask.

Taula de continguts

Taula de figures

1 Introducció

1.1 Motivació

La informàtica ha evolucionat tant que tot el que ens rodeja en el dia a dia està programat per a realitzar les funcions que desitgem. Últimament amb aquest avanç, estan apareixent també en el nostre entorn la tendència a utilitzar la intel·ligència artificial, de manera que una o varies màquines es comporten com persones, amb la possibilitat de decidir quines accions realitzar segons el seu entorn.

En les empreses actuals s'ha arribat a utilitzar aquest camp de la informàtica per a simular entorns reals, on s'utilitzen diferents entitats amb coneixements i funcions pròpies per a que puguen interactuar entre ells i trobar una possibles solucions a problemes que no s'han pogut resoldre amb tecnologies anteriors. Aquests entorns on es comuniquen diverses entitats són els sistemes de plataformes multi-agent, on les entitats amb intel·ligència artificial s'anomenen agents.

Actualment és normal trobar plataformes per a desenvolupar i executar sistemes amb agents intel·ligents. Però moltes d'aquestes són tancades respecte a la comunicació externa a la plataforma i existeixen moltes maneres de crear col·laboracions entre agents de diferents plataformes. La implementació d'aquest canal de comunicació en cada una de les plataformes multi-agents pot permetre un gran avanç en un aspecte oblidat en el món de la intel·ligència artificial.

Tenint en compte que el canal de comunicació a crear pot aprofitar els sistemes de comunicació sense fil, a part de permetre la interoperabilitat entre diferents plataformes multi-agent, també permetria la interacció entre un sistema multi-agent i altres serveis o dispositius amb el mateix protocol de comunicació, els quals tendeixen a aparéixer en molts d'aquells que qualsevol persona utilitza o té accés al seu ús sovint, incloent les que fan referència a la Internet de les coses.

1.2 Objectius

L'objectiu principal del projecte consisteix en afegir a una plataforma multi-agent un sistema de comunicació que permeta a la plataforma comunicar-se amb diferents sistemes de plataformes multi-agent diferents. El canal de comunicació cal que seguisca les restriccions del protocol REST, per a que es puga utilitzar també per a interactuar amb diversos serveis web que no requereixen ser sistemes multi-agent. Per a aconseguir aquest objectiu, es realitzaran els següents subobjectius:

- Analitzar les diferents plataformes multi-agent, incloent les dissenyades pel Grup de Tecnologia Informàtica – Intel·ligència Artificial del Departament de Sistemes Informàtics i Computació, per a després seleccionar-ne una amb la que es realitzarà el projecte.

- Analitzar els diversos *Frameworks* web per a comparar-los i escollir-ne un amb el que s'implementarà el servei web REST. L'anàlisi dels diversos *Frameworks* dependrà de la plataforma multi-agent escollida anteriorment.

- Implementar el mòdul REST dins de la plataforma multi-agent, el qual consistirà en un agent de la plataforma amb funcionalitats d'un servei REST. L'agent a implementar ha de poder comunicar-se amb els agents del mateix sistema i els sistemes multi-agent externs.

- Partint del desenvolupament del mòdul REST, crear un exemple bàsic on interactuen diferents sistemes multi-agents, de manera que es puguen trobar possibles errors de funcionament i poder arreglar-los.

- Realitzar un exemple més complexe de sistema distribuït on es demostre la utilització del mòdul REST implementat. L'exemple consistirà en que un sistema tracta d'obtindre informació d'una altra, que al mateix temps ha d'interactuar amb altres sistemes distribuïts. Junt a la implementació dels diferents sistemes de l'exemple, es realitzaran proves per a comprovar el seu funcionament i comparar diferents casos d'execució.

- Realitzar un segon exemple que simule un entorn real, on es tracta de resoldre un problema real. En aquest cas l'exemple consisteix en la creació d'un sistema distribuït que es pot aplicar en el camp de la medicina, on els sistemes multi-agent s'encarreguen de la coordinació entre diferents hospitals per a decidir quin pacient necessita amb més urgència el trasplant d'un òrgan disponible.

1.3 Estructura

La memòria es divideix en els següents punts: L'estat de l'art, el desenvolupament del mòdul, els sistemes implementats, les conclusions, la bibliografia i un glossari.

En l'estat de l'art s'explica en què consisteix una plataforma de sistemes multi-agent, descrivint les plataformes que s'han analitzat i raonant el perquè s'ha elegit la plataforma SPADE. També se descriu l'arquitectura de programari REST i la llista de *frameworks* web, que s'han tingut en compte per al final seleccionar aquella que s'ha utilitzat per a realitzar el canal de comunicació, que es tracta de Flask.

La secció de desenvolupament del mòdul explica amb més profunditat el funcionament i el que permet utilitzar la plataforma de SPADE per a realitzar sistemes multi-agent. Després conta el funcionament del *framework* web Flask per a realitzar APIs web i finalment es descriu la forma en que s'ha implementat el mòdul REST, que permet realitzar interaccions amb altres sistemes, junt a un exemple bàsic amb dos sistemes multi-agent, on s'explica pas a pas com es realitza una sol·licitud simple a un altre sistema.

En l'apartat dels sistemes implementats es mostren els diferents sistemes distribuïts implementats, amb la seua explicació de com estan dissenyades i les proves realitzades per a comprovar el seu funcionament. Mentres que un dels sistemes desenvolupats és menys complexe, l'altre es basa en la resolució d'un problema real.

En les conclusions s'explica els objectius que s'han pogut aconseguir amb aquest projecte junt a l'explicació de les diferents propostes que es poden realitzar en un futur a partir d'aquest treball.

Finalment es troben els apartats de bibliografia i glossari, on respectivament es mostren les fonts d'informació que s'han consultat per a fer aquest projecte i la llista de paraules que s'han utilitzat en la memòria que poden ser un poc complicades d'entendre per al lector, junt amb la seua definició.

2 Estat de l'art

2.1 Plataforma multi-agent

Una plataforma multi-agent consisteix en un sistema informàtic que permet generar un entorn virtual amb múltiples agents intel·ligents, que poden interactuar entre ells. Els agents són components oberts i cooperatius amb intel·ligència artificial, on cadascun d'aquests té una funció dins de la plataforma. Aquests sistemes permeten resoldre problemes que són difícils o impossible de resoldre per part d'un sol component amb intel·ligència artificial, de manera que és necessària la interacció entre diversos agents per a trobar una possible solució.

Contenen un conjunt d'eines que faciliten el desenvolupament dels agents i estalvien al programador la seua implementació, com per exemple del sistema de comunicació entre agents o del monitoratge dels estats de cada agent. Respecte al sistema de comunicació, és possible que els agents respecten algun protocol estàndard utilitzat també per moltes altres plataformes, com pot ser l'estàndard FIPA.

En general, una plataforma multi-agent ha de proveir un entorn de vida segur per als agents i proveir serveis importants, que solen ser protocols d'interacció, ontologies i sistemes de transport de missatges, que poden estar implementats amb altres agents intel·ligents[1]. També ha de proveir una eina de desenvolupament que permeta als programadors la possibilitat d'afegir manualment codi personalitzat als agents en algun llenguatge de programació, amb l'objectiu de millorar-los en les seues funcions[2].

Existeixen diverses plataformes, on cada una d'elles pot provindre d'una versió anterior d'una altra o conté característiques idèntiques o prou diferents:

Magentix2. Es tracta d'una plataforma per a sistemes multi-agents oberts, creada per la GTI-IA o Grup de Tecnologia Informàtica - Intel·ligència Artificial de la Universitat Politècnica de València[3], com una millora del projecte Magentix, respecte a les seues funcionalitats i el proveir nous serveis i eines que permeten una administració òptima dels sistemes. L'objectiu principal de la plataforma és aplicar la tecnologia dels agents intel·ligent al món real dels negocis, indústria, logística, e-comerç, salut, etc. Un altre objectiu és desenvolupar tecnologies que es puguen aprofitar en l'alt dinamisme de la topologia dels sistemes i amb interaccions flexibles, on són conseqüència de la natura autonòmica i distribuïda dels components. Seguint aquest sentit, aquesta plataforma ha sigut estesa per a suportar interaccions flexibles entre protocols i conversacions, i entre organitzacions d'agents.

Proveeix suport a tres nivells, on el primer nivell dona suport a les organitzacions, tecnologies i tècniques relacionades en les societats d'agents. El segon nivell es centra en les interaccions, tecnologies i tècniques relacionades en la comunicació entre agents. El tercer es situa en el suport a nivell d'agent, tecnologies i tècniques relacionades en els agents individuals, com el propi aprenentatge i raonament.

Magentix2 utilitza el protocol AMQP o *Advanced Message Queing Protocol*, com a fonament per a la comunicació entre agents, que facilita la interoperabilitat entre entitats heterogènies, on en esta plataforma els agents interactuen entre ells mitjançat missatges FIPA-ACL. La plataforma incorpora el servei de publicació *Trace Manager*, que s'encarrega de coordinar els processos de seguiment d'esdeveniments, permetent als agents publicar o anular les publicacions que realitzen, subscrivint o donant-se de baixa, per a poder rastrejar informació o consultar-ne en temps d'execució.

Incorpora un nou tipus d'agents, que es diu agent conversacional o *Cagent*. Aquest tipus d'agent permet la creació automàtica de converses simultànies basades en protocols d'interacció, que poden estar predefinits o poden canviar de forma dinàmica en temps d'execució. També incorpora agents argumentatius, que implementen un *framework* d'argumentació basat en casos per a generar arguments, per a seleccionar el millor argument a plantejar tenint en compte el context social, i per a avaluar els arguments proposats en el mateix diàleg per altres agents.

També esta integrat en Magentix2 el *framework* THOMAS o *Methods, Techniques and Tools for Open Multi-Agent Systems*, amb el propòsit de proporcionar un suport complet per a les organitzacions virtuals. Els agents tenen accés a la infraestructura, oferta per THOMAS, a través d'un conjunt de serveis que inclouen dos components principals, que són el facilitador de serveis i el servei d'administració d'organitzacions. La funcionalitat del facilitador de serveis és com la d'un servei de pàgines grogues i com la d'un descriptor de serveis, que s'encarrega de proporcionar un servei de pàgines verdes. El servei d'administració d'organitzacions permet la creació i gestió de qualsevol organització, dels rols que segueixen els agents i de les normes que estableixen el seu comportament.

Magentix2 incorpora a part un mòdul de seguretat que proveeix característiques respecte a seguretat, privacitat i interoperabilitat que no ofereixen altres plataformes d'agents. Aquest mòdul es basa en estàndards oberts i tecnologies de codi obert, de manera que permet el desenvolupant de sistemes multi-agents segurs i oberts al mateix temps.

La plataforma ofereix suport natiu per a poder executar agents de Jason, a part dels agents implementats en Java, de manera que aquests agents es poden beneficiar de la comunicació i els serveis que ofereix Magentix2. També té desenvolupada una

interfície en HTTP que permet als usuaris supervisar i interactuar amb els agents que estan en execució.

JADE o **Java Agent Development Framework**. És una plataforma de codi lliure, creada i desenvolupada per Telecom Italia[4], per a desenvolupar aplicacions basades en agents intel·ligents que compleixen les especificacions FIPA per a sistemes multi-agents interoperables. L'objectiu de la plataforma és simplificar el desenvolupament al mateix temps que garanteix el compliment dels seus estàndards mitjançat un conjunt de serveis de sistemes i agents. JADE pot ser considerat com un agent *middleware* que implementa una plataforma multi-agents i un *framework* de desenvolupament. S'ocupa de tots aquells aspectes que no són propis de la part interna de l'agent i que són independents de les aplicacions, com el transport de missatges, la codificació i l'anàlisi, o el cicle de vida dels agents.

L'arquitectura de JADE es compon de contenidors d'agents, que poden distribuir-se sobre una xarxa. Un contenidor és un procés Java proporcionat per l'entorn d'execució junt als serveis necessaris per allotjar i executar agents. Hi ha un contenidor principal, anomenat *main container*, que actua com a punt d'inici de l'execució de la plataforma, de manera que la resta de contenidors han d'unir-se a aquest per a que siguen registrats en la plataforma.

Els agents s'implementen com un fil de procés per agent, encara que aquests necessiten executar tasques en paral·lel. Aprofitant la implementació multiprocés que ofereix el llenguatge Java, JADE utilitza també la programació de comportaments cooperatius. Durant el temps d'execució també s'inclouen alguns comportaments per a les tasques més comunes en la programació d'agents, com els protocols d'interacció FIPA.

JADE proporciona una interfície gràfica per a l'administració remota, el seguiment i el control de l'estat de cada agent. Permet crear i iniciar l'execució d'un agent en un *host* remot, amb la condició de que ha d'haver un contenidor d'agent que s'estiga executant. També permet controlar altres plataformes d'agents remotes compatibles amb FIPA.

La plataforma ofereix tres tipus diferents d'agents bàsics per a la supervisió d'agents i missatges. Són:

- *DummyAgent*, que permet inspeccionar els missatges intercanviats entre agents, a part d'incloure una interfície per a editar, rebre, guardar i enviar missatges ACL entre agents.

- *SnifferAgent*, que pot seguir tots els missatges enviats en tota la plataforma. Permet mostrar mitjançat una interfície tots els missatges realitzats, de manera que els guarda per a un posterior anàlisi.

- *IntrospectorAgent*, que permet la supervisió del cicle de vida i l'intercanvi de missatges de cada agent.

En relació a la negociació i coordinació entre agents, JADE simplifica el seu desenvolupament en les aplicacions, on els recursos i la lògica del control es distribueixen en l'entorn del sistema. Els agents són proactius, que significa que poden controlar el seu propi fil d'execució, de manera que es poden programar fàcilment per a iniciar l'execució sense cap intervenció per part de l'usuari, basant-se només en objectius i estats. Aquesta característica fa que els entorns generats per JADE siguen adequats per a la realització d'aplicacions màquina a màquina, com per exemple la automatització d'una planta industrial o la gestió d'una xarxa de comunicacions.

Com que JADE compleix amb les especificacions FIPA, aquesta plataforma permet la interoperabilitat entre agents de diferents plataformes. Totes les aplicacions on faça falta la comunicació entre organitzacions diferents poden beneficiar-se d'aquesta interoperabilitat, incloent les de màquina a màquina. També ofereix un conjunt homogeni d'APIs que són independents de la xarxa subjacent i la versió de Java, oferint les mateixes APIs per a J2EE, J2SE i J2ME. Aquesta característica permet als desenvolupadors d'aplicacions la reutilització del codi per a diversos dispositius diferents.

Jack. És una plataforma que es descriu com un entorn per al desenvolupament, execució i integració de sistemes multi-agents basats en el llenguatge de programació Java. Ha sigut creada per *Autonomous Decision-Making Software* i es tracta d'una plataforma dissenyada a partir d'altres anteriors creades per la pròpia empresa, que són *Procedural Reasoning System* i *Distributed Multi-Agent Reasoning System[5]*.

Inclou el llenguatge de programació orientat a agents Jack i proveeix extensions, orientades als agents en el llenguatge Java, proporcionant un entorn per poder suportar un paradigma de programació orientat a agents. Aquestes extensions que fan possible aquest paradigma són la definició de noves classes base, interfícies i mètodes. També hi ha entre les extensions una sintaxis nova per a que Java suporte les classes orientades a agents i una semàntica nova per a suportar l'execució dels sistemes multi-agent. Aquesta plataforma esta tan centrada en aquest paradigma que comporta una gran versatilitat respecte a l'arquitectura dels agents, que es poden implementar des d'ordres bàsiques amb Java a comportaments complets i complexos.

L'entorn de desenvolupament proporciona la capacitat de crear agents proactius, socials i distribuïts, de manera que s'adapten dinàmicament a entorns canviants sense la necessitat de la interacció humana. Jack es basa en una plataforma lleugera que permet executar molts agents simultàniament, incloent computadors portàtils com PDAs o *Smatphones*.

L'estructura d'un agent es basa en els conceptes del paradigma BDI. El primer concepte és el conjunt de creences que té l'agent sobre l'entorn. El segon és el conjunt d'esdeveniments que rep de l'entorn, els quals l'agent pot respondre. El tercer és el conjunt d'objectius que vol aconseguir i l'ultim concepte és el conjunt de plans que permeten descriure com arribar als objectius partint de les creences de l'agent i els esdeveniments que van apareixent a l'entorn. Els agents de Jack no estan lligats a cap llenguatge de comunicació entre agents, de manera que poden utilitzar un protocol simbòlic d'alt nivell, com per exemple el de FIPA-ACL.

Una de les seues extensions més importants és la de *Jack Teams*, que fa referència al paradigma BDI i facilita el modelatge de les estructures socials i el comportament coordinat. El comportament d'un grup d'agents, i en particular la coordinació de l'activitat dels agents, es defineix des de l'entitat del grup, de manera que cada grup d'agents existeix com una entitat amb creences diferents a les dels altres agents que componen la plataforma. Aquesta capacitat d'equip proporciona una base flexible, on una gran varietat d'algoritmes de treball en equip poden ser dissenyades, desenvolupades i provades. Jack permet la programació de recerca de solucions orientades a equips d'agents, amb construccions adequades per a expressar relacions socials entre agents, com la expressió de l'activitat coordinada.

Jack permet aplicar sistemes multi-agents en entorns sintètics. Els entorns sintètics simulen la física dels aspectes del món, com el terreny, clima, vehicles i persones. Per al cas del comportament humà, Jack utilitza una nova arquitectura cognitiva, *Cojack*, que permet la creació de models de la conducta humana, que preveuen com els éssers humans es comporten en una situació determinada, segons la teoria en la que es basa aquesta arquitectura.

SPADE. És una plataforma d'agents intel·ligents basada en la tecnologia XMPP/Jabber i que suporta l'estàndard de comunicació FIPA. La plataforma ha sigut desenvolupada en el llenguatge de programació Python i distribuïda amb una llicència de *software* lliure per la GTI-IA de la Universitat Politècnica de València.

A part de tindre una plataforma per a agents, SPADE conté una llibreria en forma de mòdul de Python per a poder crear agents amb tota una col·lecció de classes, funcions i eines. En un futur, aquesta plataforma permetrà la utilització d'altres tipus d'agents

implementats en altres llenguatges de programació, sempre que s'hi comuniquen mitjançat el protocol XMPP amb missatges de comunicació en el llenguatge FIPA-ACL.

El protocol de missatgeria XMPP conté un conjunt de característiques que faciliten la comunicació entre agents. La primera és l'ús de la comunicació asíncrona, de manera que un emissor emmagatzema el seu missatge fins que el receptor torni a estar connectat. La segona característica és la seguretat, que s'aplica en la capa de transport XMPP que permet l'intercanvi de missatges entre dos extrems, on l'autentificació i l'encriptació del canal de comunicació compleixen amb els protocols TLS i SASL. L'última consisteix en la flexibilitat que permet XMPP, perquè es pot estendre més enllà del seu ús en la missatgeria instantània.

Els agents en SPADE estan compostos per un mecanisme de connexió a la plataforma, un sistema de tramitació de missatges i un conjunt de mètodes de comportament. Les comunicacions son manipulades per mitjans del protocol Jabber. Aquest protocol té un mecanisme per a registrar i autentificar usuaris en un servidor de Jabber, que en aquest cas els clients del servidor són agents intel·ligents. El sistema de tramitació de missatges que conté cada agent en SPADE actua de manera que quan un agent rep un missatge, l'emmagatzema en una cua, depenent del tipus de missatge, i quan un agent vol enviar-ne un, el tramitador el posa en el servidor Jabber fins que el puga rebre el receptor.

El comportament d'un agent consisteix en una tasca o funció que s'executa a partir d'una plantilla de missatges, que controla entre un conjunt de plantilles quin dels mètodes de comportament s'ha d'utilitzar depenent dels tipus de missatges que va rebent l'agent. Entre els tipus de comportaments que ofereix SPADE estan el d'executar una tasca una vegada o contínuament. Partint d'aquests comportaments, la plataforma permet implementar també als agents com màquines d'estat finit, on un agent segueix una traça d'accions, representats com estats, fins a arribar a un estat final.

SPADE també permet la creació d'agents partint de l'arquitectura de creences, desitjos i intencions. El sistema BDI està desenvolupada amb computació orientada a serveis junt a la compilació dinàmica de serveis, també coneguda com computació orientada a objectius distribuïts. Cada agent BDI té una base de dades, on emmagatzemen les creences que van adquirint. La forma en la que es guarda aquesta informació pot ser elegida entre un conjunt d'implementacions de PROLOG, que són SWI.Prolog, Flora2, Eclipse o SPARQL. Per defecte SPADE utilitza SpadeKB per a evitar l'ús de mòduls externs. SpadeKB suporta la lògica de primer ordre, permet afegir, eliminar o buscar creences, i és capaç de emmagatzemar dades en forma de tuples, els quals contenen un nom junt a un valor de diferents tipus[6].

Jason. Es tracta d'una plataforma de sistemes multi-agent desenvolupada en Java amb codi obert pels desenvolupadors Jomi Fred Hübner i Rafael Heitor Bordini. La utilització d'aquest llenguatge de programació permet la personalització de molts aspectes d'un agent intel·ligent.

La plataforma utilitza com a extensió del programa el llenguatge de programació orientat a agents AgentSpeak per a programar l'entorn de cada agent, a part de que permet implementar operacions semàntiques i el seu ús com a eina de desenvolupament de sistemes multi-agent. Jason s'utilitza com un *plugin* per a entorns integrats de desenvolupament, com Eclipse o NetBeans, i per a altres plataformes de sistemes multi-agent com si es tractés d'un *middleware* de sistemes distribuïts basats en agents, com per exemple JADE.

En el camp dels llenguatges de programació orientats a agents intel·ligents, AgentSpeak es tracta d'un dels llenguatges abstractes amb més influencia basat en l'arquitectura BDI, que és una de les millors aproximacions respecte a la implementació d'agents cognitius. El tipus d'agents programats amb AgentSpeak són moltes vegades denominats com a sistemes de planificació reactius. Una de les característiques més importants d'aquest llenguatge és el seu fonament teòric, que és una implementació de la semàntica operacional, formalment aplicat a aquest llenguatge i a moltes de les extensions de Jason.

A més d'interpretar el llenguatge AgentSpeak, Jason també conté altres característiques com el maneig de plans que no han tingut èxit durant la seua execució, acte de parla basat en la comunicació entre agents i anotació de creences sobre les fonts d'informació. També permet l'ús d'anotacions d'objectius declaratius, variables d'alt ordre, ordres d'estil imperatiu en el contingut d'un pla i varies altres extensions de llenguatge. Jason té suport de sistemes multi-agent i d'agents que raonen sobre ells que utilitzen el model de Moise+, que és un model d'organització de sistemes multi-agent basat en nocions com rols, grups i missions[7].

Respecte a l'elecció de la plataforma de sistemes multi-agents per a implementar el mòdul de Restful, s'ha tingut en compte que ha de ser una plataforma desenvolupada dins del grup GTI–IA de la Universitat Politècnica de València, ja que permet la consulta i resolució de dubtes directament per part del seus creadors i la possible millora de la plataforma al afegir una nova funció, de manera que la llista de plataformes a poder utilitzar en aquest projecte es redueix a dos, Magentix2 i SPADE. Entre aquests dos s'ha escollit SPADE perquè està implementada en Python, un llenguatge de programació amb una comunitat de programador que està en creixement, i el seu sistema permet determinar l'estat actual d'un agent en temps real, mitjançat la notificació de presència que utilitza el protocol de comunicació XMPP.

També permet la gestió d'organitzacions d'agents i té una corba d'aprenentatge en el seu ús més baixa que la de Magentix2, degut a que s'ha utilitzat anteriorment per a realitzar altres projectes.

Una vegada escollida la plataforma multi-agent, es realitza una explicació de l'arquitectura REST i de l'anàlisi dels diferents *Frameworks* web que es poden utilitzar amb SPADE, que es mostren en els següents apartats.

2.2 REST

Representational State Transfer o el servei web Restful és una arquitectura de programari que proveïx la interoperabilitat entre sistemes informàtics distribuïts a través d'Internet. El terme REST es va introduir l'any 2000 per Roy Thomas Fielding, que el va utilitzar per a dissenyar el protocol HTTP 1.1 i l'identificador uniforme de recursos o URI. Amb aquest terme se volia demostrar com una aplicació web ha d'estar ben dissenyada, com si es tractara d'una màquina d'estats virtual, on un usuari pot progressar a través de l'aplicació mitjançat la selecció d'enllaços i operacions com transicions entre estats, de manera que en cada estat es transfereix a l'usuari informació per al seu ús[8].

REST permet als sistemes realitzar peticions per a accedir i manipular recursos web mitjançat operacions uniformes, predefinides i sense cap relació amb altres peticions, seguint un protocol sense estats, que s'explicarà a continuació. Un servei d'aquest tipus és un model centrat en les dades, on els recursos venen identificats per URIs i conté la informació emprant el format XML o JSON entre altres. Aquests recursos poden ser manipulats partint d'accions predefinides en les capçaleres HTTP. Fent ús d'operacions independents entre ells, un sistema REST pot aspirar a tindre un major rendiment, una major fiabilitat i escalabilitat, mitjançat la reutilització de components que poden ser gestionats i actualitzats sense afectar a tot el sistema, inclús si està en execució. El tipus d'operacions permeses en REST inclou aquelles predefinides per part de HTTP, on les més utilitzades són:

GET, per a accedir o obtindre informació d'un o múltiples recursos.

POST, per poder publicar o crear un nou recurs.

PUT, per a actualitzar o reemplaçar tota la informació d'un recurs existent amb altra informació.

PATCH, per a actualitzar o modificar part de la informació continguda en un recurs existent.

DELETE, per poder eliminar un recurs existent.

En un servei REST s'han de complir 6 restriccions, de manera que permeten al servei aconseguir rendiment, escalabilitat, simplicitat, modificabilitat, portabilitat i fiabilitat:

Interfície uniforme. Defineix la interfície que ha d'haver entre clients i servidors. Simplifica i desacobla l'arquitectura del sistema, el qual permet a cada part desenvolupar-se de forma independent. Els 4 principis d'una interfície uniforme són:

- Identificació de recursos, on els recursos individualment són identificats mitjançat URIs. Els recursos en sí mateix estan separats de la representació que s'envia al client per part del servidor.

- Manipulació dels recursos mitjançat les seues representacions. Quan un client té la representació d'un recurs, incloent qualsevol metadada adjunta, aquest té prou informació per a poder modificar o eliminar el recurs del servidor, sempre que el client tinga permisos per poder fer-ho.

- Missatges auto-descriptius, on cada missatge conté suficient informació per a descriure com s'ha de processar ell mateix.

- *Hypermedia as the Engine of Application State* o HATEOAS. Els clients lliuren l'estat del recurs mitjançat el seu contingut, els paràmetres de consulta, les seues capçaleres i URI de petició. Els servidors lliuren l'estat del recurs mitjançat el seu contingut, les seues capçaleres i codi de resposta. Açò tècnicament fa referència a l'hipermèdia o l'hiperenllaç amb hipertext. Partint de la descripció anterior, és necessari que hi haja enllaços en el contingut o en les capçaleres dels recursos enviats, per a proveir una URI que permeta recuperar el recurs mateix o d'altres relacionats.

Sense estats. La comunicació entre client i servidor no necessita que el servidor guardi informació del client entre diverses peticions consecutives. És necessari l'ús d'un únic estat per a gestionar la sol·licitud que es manté, com si fora part de la URI, dels paràmetres de consulta, del seu contingut o de les seues capçaleres. La URI només identifica el recurs i el seu contingut té l'estat d'aquest. Després de que el servidor processe l'estat o alguna de les seues parts que importen, es retorna al client mitjançat capçaleres i continguts de resposta.

Normalment es programa amb un contenidor, que actua com una sessió i manté l'estat a través de múltiples sol·licituds HTTP. Però en REST el client ha d'incloure tota la informació per a realitzar una petició al servidor, reenviant el mateix estat les vegades

necessàries si s'han d'abastar diverses sol·licituds. Açò permet una major escalabilitat, ja que el servidor no ha de mantindre, actualitzar o comunicar l'estat d'una sessió.

Cacheable. Les respostes del servidor poden guardar-se o no en una memòria cau, tant de manera implícita, explícita o negociada, per a preveure en els clients la reutilització d'informació inapropiada en respostes per a altres sol·licituds. Amb açò es vol minimitzar les interaccions entre client i servidor, fent que el client accedeixi a la representació del recurs, guardada en una memòria cau, i millorar l'escalabilitat i rendiment del sistema.

Separació Client-Servidor. Ja s'havia mencionat que una interfície uniforme permet separar el client del servidor. Esta separació permet que el client no es preocupe per la informació emmagatzemada, que pot emmagatzemar-se en diferents servidors, de manera que millora la portabilitat del codi font del client. El servidor tampoc s'ha de preocupar de la interfície o l'estat del client, de manera que el servidor pot ser més escalable. El servidor i el client també poden ser reemplaçats i desenvolupat de manera independent, sempre que la interfície que els separa no s'altere.

Sistema de capes. Un client de forma ordinària està connectat directament a un servidor final o a un intermediari situat entre els dos. L'ús de servidors intermediaris permet una millor escalabilitat repartint la càrrega dels recursos i utilitzant una memòria cau compartida. Un sistema de capes també permet millorar les polítiques de seguretat del mateix.

Codi a petició. El servidor temporalment pot ser capaç de millorar o personalitzar la funcionalitat d'un client, transferint-li codi que ell puga executar. Exemples de codi poden ser components compilats en Java o *scripts* en JavaScript. Aquesta restricció és opcional en l'arquitectura REST.

Complint les anteriors restriccions, un sistema distribuït amb l'arquitectura REST podrà obtindre les propietats emergents desitjables. En el cas de que no es complisca qualsevol de les restriccions obligatòries, el sistema no pot denominar-se estrictament com que un servei Restful[9][10].

2.3 Frameworks web

Per a poder crear un servei REST, un programador pot desenvolupar-lo completament a ma o pot utilitzar un *framework* web, que proveïx una col·lecció de paquets de codi o mòduls que permeten la creació d'una aplicació o servei web sense la necessitat d'implementar protocols, *sockets* o gestors de processos a baix nivell. Generalment els *frameworks* proveïxen suport per a diverses activitats com la interpretació de peticions, la generació de respostes o l'emmagatzematge de dades de manera persistent.

Per a aplicacions web amb una implementació complexa, que requereixen de diversos tipus d'abstraccions normalment situats un per damunt de l'altre com si es tractés d'una pila, existeixen *frameworks* que intenten proveir una solució completa, coneguts com *frameworks* web de pila completa, que intenten proporcionar components a cada capa de la pila. Aquest tipus de *framework* permet a una aplicació web la utilització de combinacions de bases de dades HTTP, mecanismes d'emmagatzematge com els d'una base de dades, motor de plantilles, un transportador de peticions, un mòdul d'autentificació d'usuaris i un conjunt d'eines AJAX, que consisteixen en tecnologies web que permeten a un usuari crear aplicacions web asíncrones.

També existeixen *frameworks* web que no són de pila completa, també anomenats *microframewoks* web, que utilitzen una càrrega de recursos baixa durant la seua execució i proveïxen la base d'un servei o aplicació web, tant si s'executa en els seus propis processos independents sobre un servidor HTTP de Apache o en un altre entorn distint. En els *microframeworks* el programador pot introduir els motors de plantilles que vullga i altres components per a executar damunt de la pila, encara que molts d'aquests ja proveïxen eixe tipus de tecnologies[11].

Encara que n'hi han *frameworks* web que permeten directament la implementació de serveis amb l'arquitectura REST, la majoria estan dissenyats per a poder adaptar les característiques d'una API web, per a que aquesta complisca amb les restriccions necessàries mencionades en l'apartat anterior.

Com que en aquest projecte s'ha decidit utilitzar la plataforma muti-agent SPADE, és necessari un *framework* web que permeta l'ús del llenguatge de programació Python per a la implementació del servei de comunicació externa de la plataforma. Entre els més coneguts, s'han trobat els següents:

Flask. És un *microframework* web basat en les eines WSGI de Werkzeug i el motor de plantilles Jinja2, creat en el 2010 per Armin Ronacher amb llicencia BSD. Actualment existeixen aplicacions web conegudes que utilitzen Flask a part de la pàgina de la comunitat web del propi *framework*, com per exemple Pinterest o LinkedIn.

El *framework* no requereix d'eines i llibreries particulars externes, no té capa d'abstracció de base de dades, ni cap altre component on llibreries existents de tercers proporcionen funcions idèntiques. Però també suporta extensions que poden afegir característiques a una aplicació com si estigueren implementades en Flask. Aquestes extensions serveixes normalment per a validacions, control de càrrega, afegir tecnologies d'autentificació i inserir eines relacionades en *frameworks* pareguts.

Entre les seues característiques, conté depuradors de serveis, servidors de desenvolupament, suport integrat d'unitats de proves i suport de *cookies.* També

suporta el maneig de peticions REST i es basa en l'estàndard d'Unicode. Suporta completament les últimes versions de Python 2, mentres que en Python 3 existeixen extensions que encara no ho suporten[12].

El codi font de Flask està disponible des d'un repositori de GitHub. Respecte a exemples implementats amb aquesta eina, es poden trobar fàcilment moltes pàgines web, inclús diferents documentacions, on expliquen com implementar tant d'APIs web bàsiques com de REST amb aquest *framework* i resolen dubtes del seu funcionament, a part de que hi ha una comunitat oficial per a obtindre ajuda respecte a dubtes.

Bottle. Aquest altre *microframework* també està desenvolupat amb les especificacions WSGI, inventat l'any 2009 per Marcel Hellkamp baix la llicència MIT. Està dissenyat per a ser ràpid i lleuger, per a poder crear aplicacions web de manera fàcil, i es distribueix com un mòdul sense cap altra dependència que la llibreria estàndard de Python, de manera que pot utilitzar-se tant en les últimes versions de Python 2 i 3.

Permet l'ús de relacions entre URLs dinàmiques i mètodes, que s'activen quan un usuari realitza una consulta o petició, conté un motor per a la implementació de plantilles ràpid que en suporta de Mako, Jinja2 i Cheetah, i utilitza un accés convenient a dades, arxius, *cookies*, capçaleres i altres metadades relacionades amb HTTP. També conté implementat una llibreria de comunicació asíncrona i un servidor de desenvolupament en HTTP que suporta Paste, Fapws3, Bjoern, Gae, CherryPy o qualsevol altre servidor amb l'estàndard WSGI[13].

En Internet existeixen molts tutorials per a aprendre a utilitzar-lo com a servei web i en la pàgina oficial de Bottle es pot consultar tota la documentació necessària per a comprendre el seu funcionament, tenint en compte que hi ha una comunitat en Google per a resoldre dubtes i un repositori en GitHub amb el codi font del *framework* web.

Falcon. És també un *microframework* web ràpid i lleuger, dissenyat per Kurt Griffiths i distribuït per l'empresa Rackspare amb la llicencia de Apache 2.0 l'any 2013. S'utilitza en la creació de petits serveis, aplicacions en segon pla i *frameworks* d'alt nivell. Existeixen varies organitzacions que ho utilitzen com Cronitor, EMC, Hurricane Electric, OpenStack, Opera Software, Wargaming i Rackspace.

Permet desenvolupar aplicacions i serveis web amb les últimes version de Python 2 mentres que en Python 3 només permet el seu ús en les versions 3.3, 3.4 i 3.5, encara que també es pot utilitzar amb Cpython, PyPy i Jython, de manera que milloren el seu rendiment. Falcon té dependències amb Six i Python-mimeparse. Six és una llibreria que proveïx utilitats per a resoldre diferències entre les distintes versions de Python i Python-mimeparse és un mòdul que permet l'ús de funcions per a gestionar *mime-types*.

Aquest *microframework* web utilitza rutes basades en plantilles URI RFC, assignacions d'URIs a recursos basat en REST i objectes de petició i resposta intuïtius. També té suport de l'estàndard d'Unicode, un sistema de respostes d'errors en format HTTP, un entorn per a la realització de proves i suport per a l'ús de llibreries de comunicació asíncrona[14].

Falcon està expòs també en un repositori de GitHub i existeixen exemples sobre la seua utilització en la creació d'APIs, dispersos en varies pàgines web i en la seua documentació, que és accessible des de la pàgina oficial.

Django. És un *framework* web de pila completa que segueix l'arquitectura *model-view-controller* o MVC, que consisteix en un assignador de relacions d'objectes que intervé entre models de dades, una base de dades relacional, un sistema per a processar peticions HTTP amb un sistema web de plantilles, i un remitent d'URLs basat en expressions regulars. Django va ser creat per la fundació *software* de Django l'any 2005 baix la llicència BSD de 3 clàusules. Hi han pàgines web que utilitzen aquest *framework* com Instagram, Mozilla, The Washington Times, Disqus, Bitbucket i Nextdoor.

L'objectiu primari de Django és la facilitació en la creació de pàgines webs amb bases de dades complexes. Està dissenyat per a facilitar el desenvolupament als programadors, conté sistemes de seguretat que ajuden en la implementació per a evitar error comuns, és escalable i permet la utilització d'un gran conjunt d'extensions, com l'autentificació d'usuaris, l'administració de continguts i canals RSS entre moltes.

Dins del *framework* s'inclou un servidor web lleuger i independent per a realitzar proves, un sistema de serialització i validació que pot traduir formes en HTML i valors per a emmagatzemar en una base de dades, i un *framework* cau que pot utilitzar diversos mètodes emmagatzematge web. També té un sistema per a estendre les capacitats del motor de plantilles, un sistema d'internacionalització per a traduir components propis de Django en una varietat de llenguatges i un sistema de serialització que pot produir i llegir representacions d'instàncies XML o JSON referents a models de Django[15].

Actualment es pot utilitzar només en l'ultima versió de Python 2 i les últimes versions de Python 3, existeix un repositori també en GitHub, té una documentació extensa tant en la seua pàgina web oficial com en altres pàgines o llibres i hi ha una gran varietat d'exemples per a implementar APIs web, incloent les que tenen arquitectura REST, tant d'oficials com no en Internet.

CherryPy. És un *microframework* web orientat a objectes HTTP que permet als programadors implementar aplicacions i serveis web com si es tractés d'un programa

de Python orientat a objectes. Es va dissenyar en el 2001 per The CherryPy Team amb llicencia BSD. Està dissenyat per a un desenvolupament ràpid d'aplicacions web mitjançat l'adaptació de protocols HTTP i pot servir com un servidor web que pot utilitzar-se partint d'un entorn compatible amb els estàndards WSGI. Partint de CherryPy s'executen pàgines web com Netflix, CherryMusic, Aculab Cloud o Listic, i altres productes com TurboGears, Indigo, SABnzbd o read4me.

Implementa un compilador de HTTP, un servidor web *thread-pooled*, un entorn de proves de funcionament d'aplicacions, un conjunt d'eines per a codificació, cau de web, continguts estàtics, creació de sessions i sistemes d'autorització d'usuaris. CherryPy permet el control simultani de diversos servidors HTTP i conté un adaptador natiu de mod_python, que és un modul de servidor web de Apache.

Suporta les últimes versions de Python 2 i qualsevol de Python 3, incloent varies implementacions del propi llenguatge com Cpython, IronPython, Jython i PyPy. El *microframework* no té dependències amb altres llibreries, encara que es poden instal·lar algunes que permeten utilitzar funcions extra, com una llibreria per a processar informació en JSON, utilització de memòria cau en sessions, o rutes per al maneig de URLs amb mètodes[16]. Té un repositori amb el codi font en GitHub, un grup de programadors i canal de missatgeria per a resoldre dubtes i debatre sobre CherryPy, i exemples de diverses implementacions de serveis en diferents pàgines web no oficials, a part del tutorial de la web oficial.

Pyramid. Es tracta d'un *microframework* web de codi obert basat en WSGI, creat l'any 2008 per l'organització Pylons Project baix la llicència BSD. Està inspirat per els *frameworks* web Zope, Pylons i Django, i s'utilitza en diverses organitzacions i companyies com AdRoll, BraveWords, ITCase, Noppo i Yelp entre moltes.

Està basat en l'arquitectura *model-view-controller* i integrat amb bases de dades SQL mitjançat l'eina *software* SQLAlchemy i amb la base de dades d'objectes Zope. Pyramid permet als programador la definició de rutes utilitzant expressions regulars que assignen als objectes. També permet recorreguts d'objectes jeràrquics, on cada part de una URL és un objecte que conté un altre, com si es tractés d'un sistema de carpetes d'arxius.

Permet l'ús de decoradors de funcions per a especificar les rutes i el tipus de petició entre altres paràmetres, la utilització de predicats per a limitar les correspondències d'una finestra cridada depenent del nom o el tipus, i l'ús de renderitzadors per poder retornar continguts en diferents tipus de serialització, com JSON. També permet l'especificació de continguts dins d'un paquet extern per a obtindre diferents tipus de renderització i la utilització d'*events* i subscriptors. Un *event* és un objecte que s'executa en un moment determinat en temps d'execució d'una aplicació i un

subscriptor dins d'un *event* permet executar un codi concret, com enviar un missatge o carregar una imatge. A part existeixen moltes extensions que milloren o aporten més funcionalitats al *framework* web[17].

El codi font de Pyramid també està en GitHub i es pot executar amb suport complet en les últimes versions de Python 2 i 3. En la pròpia pàgina web oficial existeix una gran varietat d'exemples, a part d'una extensa documentació amb les funcions de l'API del *microframework*, encara que no hi han moltes referencies a REST. Hi han diversos grups de comunitats de programadors que permeten debatre funcionalitats de Pyramid, com resoldre dubtes o casos concrets de la seua utilització en un servei web.

TurboGears. És un *framework* web dissenyat mitjançat l'arquitectura MVC per Kevin Dangoor l'any 2005 amb llicència MIT. Permet crear serveis web tant de pila completa com de incompleta, tal com fan els *microframeworks* web. Les organitzacions o webs que l'utilitzen són SourceForge, la comunitat de Fedora, Kamisons, PyF i Moksha entre altres.

El *framework* web està compost per eines WSGI menudes que poden treballar en conjunt i generar bons resultats, que són Kid, Genshi, Mako, SQLObject i SQLAlchemy, que permeten l'ús d'un model de dades, una base de dades relacional, el motor de plantilles, i un controlador que transforma l'entrada de dades en ordres per al sistema web. Té suport simultani de múltiples bases de dades, particionat de dades en horitzontal, suport de varies eines de JavaScript i suport de diversos formats d'intercanvi de dades[18].

Treballa amb les últimes versions de Python 2 i amb les versions de Python 3 a partir de la 3.3, existeix un repositori en GitHub amb el codi font del *framework* web, documentació i eines de desenvolupament. També té una comunitat de programadors que permet realitzar consultes, crear projectes i extensions de TurboGears. Hi han exemples d'implementacions d'APIs web bàsiques per internet, incloent algunes implementacions d'APIs REST.

web2py. És un altre *framework* web de pila completa de codi obert creat per Massimo Di Pierro en el 2007 baix la llicència LGPLv3. web2py està inspirat en Ruby on Rails i Django, de manera que es centra en un desenvolupament ràpid i una arquitectura MVC. S'utilitza en moltes pàgines web com TechJobs, Movuca, Memeforge, Ourway i LinkFindr.

Pot executar-se amb Apache, Lighttpd, Cherokee i amb molts altres servidors per la via de CGI, FastCGI, WSGI, mod_proxy o mod_python. Pot estar acoblat amb *middlewares* i aplicacions WSGI de tercers. Utilitza mètodes de seguretat per a

preveure tipus de vulnerabilitats molt comuns en els serveis web i compleix amb les bones pràctiques referents a la enginyeria del *software*, que fan del codi dels serveis més llegible, escalable i fàcil de mantindre, tal com fa el disseny *model-view-controller,* la validació de la secció del servidor o el maneig segur d'arxius enviats. Utilitza múltiples protocols com HTML/XML, RSS/ATOM, PDF, JSON, AJAX i REST entre altres.

Inclou un servidor web, una base de dades relacional, un entorn de desenvolupament web, un sistema de gestió d'interfícies web, múltiples mètodes d'autentificació, controls basats en rols, una capa abstracta de base de dades i una memòria cau basada en RAM i ROM per a millorar l'escalabilitat d'un sistema[19].

S'executa amb qualsevol de les últimes versions de Python 2, en PyPy o en Jython. El codi es pot trobar en el seu repositori de GitHub i existeixen diversos grups de suport en diferents idiomes. Existeixen diversos documents extensos per part de la pàgina web oficial com per part de les comunitats de programadors que es centren en web2py, al igual que en exemples d'implementacions de serveis o aplicacions web.

Tornado. Aquest és un altre *microframework* web inventat l'any 2009 per l'organització FriendFeed amb la llicència Apache 2.0. Es caracteritza per la seua llibreria de creació de xarxes asíncrones. També utilitza un sistema I/O en xarxa sense bloqueig que permet a Tornado la utilització de moltes connexions obertes, sent ideal per a aplicacions que requereixen una connexió a llarg termini amb distints usuaris, i una llibreria de corutines que permet escriure el codi asíncron d'una manera més senzilla en comptes de la forma de desencadenament de crides.

El *framework* web ofereix, junt a un servidor HTTP, una alternativa de sistema de pila completa a WSGI, on és possible utilitzar Tornado en un contenidor WSGI o utilitzar un servidor de Tornado com un contenidor d'un *framework* web amb estructura WSGI, encara que qualsevol dels casos tenen limitacions. En el cas d'utilitzar Tornado amb el seu propi servidor HTTP s'aconsegueixen superar aquestes limitacions[20].

Suporta l'última versió de Python 2 i qualsevol de Python 3 a partir de la versió 3.3, encara que és recomanable l'ús de Python 2 per a un millor suport del la tecnologia integrada de SSL. Existeix un canal de suport oficials i per a discussions en casos referents a l'ús de Tornado, i un repositori en GitHub amb el codi font del *framework* amb la documentació, exemples de serveis i informació respecte a llibreries externes que afegeixen funcionalitats a Tornado.

Per a la selecció del *framework* web a utilitzar en la implementació del mòdul Restful en la plataforma de sistemes multi-agents, s'han analitzat diferents característiques de cadascun d'ells. S'ha comparat les últimes versions estables de cadascun d'ells junt al

temps que han estat existint, per a saber si actualment segueixen desenvolupant-se i si es tracten de projectes de *software* madurs, encara que existeixen casos de projectes amb pocs anys d'existència que es basen en altres amb molts anys de desenvolupament i que estan tenint una comunitat de programadors en creixement.

Un altra característica a tindre en compte és la compatibilitat amb diferents versions de Python, on permet vore si només estan disponibles per a la versió de Python 2 o esta implementat o adaptant-se també a Python 3, ja que suposa una millora en el rendiment dels programes comparat amb l'anterior versió. També s'ha tingut en compte si són *frameworks* web de pila completa o no, de manera que el seu ús pot demanar o no un major consum de recursos.

Altres característiques importants són l'existència en Internet d'exemples d'implementacions de distints serveis webs de cadascun d'aquests *frameworks*, concretament aquells relacionats amb l'arquitectura i protocols de REST, on també permet comprovar la corba d'aprenentatge en la utilització de cadascun d'ells, de manera que permet conéixer si la seua programació és intuïtiva i requereix poc de codi per al seu funcionament. Com a última característica comparada és la llicència d'ús, per saber si és accessible el codi font del *framework* web i si existeixen limitacions en la seua utilització.

A continuació es mostra una taula on es comparen les característiques de cadascun dels *frameworks* descrits anteriorment(Fig. 1).

Framework web	Última versió estable	Anys en activitat	Versions de Python	Tipus de framework web	Exemples de codi REST	Aprenentatge d'ús	Llicència
Flask	0.12.2 (2017)	7 anys	2.6+, 3.3+	Microframework	Molts	Ràpid	BSD
Bottle	0.12.13 (2017)	8 anys	2.5+, 3.x	Microframework	Molts	Ràpid	MIT
Falcon	1.2.0 (2017)	4 anys	2.6+, 3.3-3.5	Microframework	Prou	Ràpid	Apache 2.0
Django	1.11.1 (2017)	12 anys	2.7, 3.4+	Framework de pila completa	Prou	Lent	BSD de 3 clàusules
CherryPy	10.2.2 (2017)	16 anys	2.6+, 3.x	Microframework	Pocs	Mitjà	BSD
Pyramid	1.8.3 (2017)	9 anys	2.7, 3.4+	Microframework	Prou	Mitjà	BSD
TurboGears	2.3.10 (2016)	12 anys	2.6+, 3.3+	Microframework - Framework de pila completa	Pocs	Lent	MIT
web2py	2.14.6 (2016)	10 anys	2.6+	Framework de pila completa	Pocs	Lent	LGPLv3
Tornado	4.5.1 (2017)	8 anys	2.7, 3.3+	Microframework	Prou	Mitjà	Apache 2.0

Fig. 1. Comparació de característiques dels *Frameworks* web

En un principi, s'han descartat aquells que consisteixen en *frameworks* web de pila completa, degut a que poden requerir de molts recursos i no és recomanable per a implementar un servei web d'aquest tipus amb un agent de SPADE, que requereix de poca càrrega. Després s'han descartat els que no tenen una corba d'aprenentatge fàcil, no molt intuïtius a l'hora de programar i que no tenen una comunitat en Internet que publiquen molts exemples d'implementacions d'APIs Restful, de manera que al final només queden dins de la selecció els *microframeworks* web Flask i Bottle.

Aquests dos *frameworks* web tenen unes característiques molt similars, inclús en la implementació d'APIs web, encara que tenen diferents llicències, que en el seu ús són similars. No suporten les mateixes versions de Python però si que se poden utilitzar en les últimes tant de Python 2 i 3, i només es duen un any de maduresa referent al seu *software*. També si aprofundim en les diferencies d'aquests *frameworks*, Flask conté un gestor d'arxius web per a la creació i modificació de pàgines web, permet la

utilització de moltes extensions[1] i es mostra en molts documents rellevants referents a l'aprenentatge en la implementació de serveis web en Python mitjançat *microframeworks*[2]. Per l'altre costat, Bottle està implementat només amb la llibreria estàndard de Python dins d'un únic arxiu de codi i demostra ser molt ràpid, tal com mostren alguns anàlisis de rendiment de diferents *frameworks* web en versions estables anteriors[3].

Partint de les característiques mencionades anteriorment, finalment s'ha escollit Flask. Encara que Bottle té millor rendiment durant la seua execució, s'ha preferit més la facilitat de trobar informació sobre implementacions de codi i informació respecte a la resolució de dubtes o problemes similars als d'altres programadors. També s'ha tingut en compte que Flask pot servir en un futur com a *microframework* web per a posteriors projectes, on és possible l'ús d'HTML. Amb aquesta selecció s'implementarà el servei REST que s'incorporarà en la plataforma de sistemes multi-agents SPADE. Açò s'explicarà en el següent apartat, on es mostra el funcionament del sistema SPADE, del mòdul REST i de quina manera s'ha programat aquesta inserció a un agent.

1 Quora. *Flask vs. Bottle, what are their advantages and disadvantages?* https://www.quora.com/Flask-vs-Bottle-what-are-their-advantages-and-disadvantages

2 Google. *Google search: python framework rest*. https://www.google.es/search?q=python+microframework+rest&ie=UTF-8&sa=Search&channel=fe&client=browser-ubuntu&hl=en&gws_rd=cr,ssl&ei=TdM6Wf2TB8ixatnlhfgP#channel=fe&hl=en&tbm=bks&q=python+framework+rest

3 Kirill Klenov. *Python Web Framework Benchmarks.* http://klen.github.io/py-frameworks-bench/#results

3 Desenvolupament

3.1 SPADE

Ja s'ha mencionat anteriorment que SPADE és una plataforma d'agents intel·ligents implementada en Python basada en el protocol XMPP i que suporta l'estàndard FIPA. També s'havia dit com està compost un agent en la plataforma i quins tipus da comportaments utilitza per a poder realitzar accions.

Per a que es puga executar un sistema de SPADE, primer cal crear la plataforma i després afegir-hi els agents mitjançat un sistema de registrament XMPP, que manipula internament les comunicacions i els registres de la plataforma multi-agents. Cada agent, quan es registra en la plataforma, ho fa amb un identificador, conegut com Jabber ID o JID, i una contrasenya. El JID es mostra com una direcció de correu composta per un nom d'usuari i un domini de servidor, que per al cas dels agents es tracta del nom que s'utilitza per a identificar-lo i el servidor des d'on s'està executant. Normalment la direcció XMPP d'un agent és el seu propi canal de comunicació, on s'afegeix el prefixe 'xmpp://'. Si el registre es realitza amb èxit, l'agent mantendrà un canal de comunicació XMPP obert amb la plataforma.

Un agent intel·ligent en SPADE es compon per un mètode *_setup()* i un conjunt de mètodes de funcionalitat, que poden ser opcionals. El primer mètode es tracta de la primera part que s'executa en un agent quan s'activa després de registra-se en una plataforma. El *_setup()* s'utilitza normalment per a inicialitzar paràmetres determinats, registrar serveis en el facilitador de serveis o agents en el sistema d'administració d'agents de la plataforma, o afegir mètodes de comportament al propi agent. Els mètodes de comportament consisteixen en les funcionalitats o les tasques que pot realitzar un agent dins de la plataforma, que es poden definir com varies subclasses d'un agent, instanciant-les des del mètode *_setup()* per a que es puguen executar. També es poden definir comportaments com una classe fora del constructor d'un agents, que s'afegeix a l'agent quan aquest es registra en la plataforma i comence a executar-se, de manera que s'evita la repetició de codi en distints agents que tenen comportaments idèntics.

A continuació es mostra un diagrama de seqüència(Fig. 2), on un *script* registra i executa un agent dins d'una plataforma de SPADE. Una vegada activat, l'agent executa automàticament el seu mètode *_setup()*. Després aquest agent seguirà actiu en la plataforma fins que el *script* decidisca parar la seua execució.

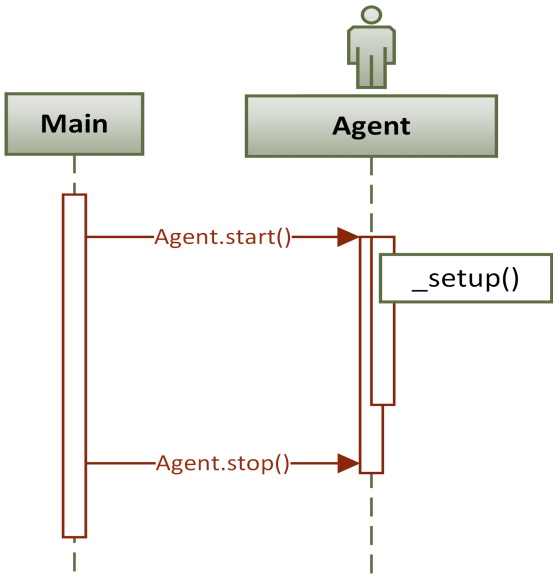

Fig. 2. Execució d'un agent bàsic

El comportament d'un agent per defecte conté dos mètodes, que són *onStart()* i *_process()*, on el primer s'executa quan comença a executar-se el comportament i el segon el segueix en l'execució. Les diferències que n'hi han entre aquests mètodes són que *_process()* cal definir-lo en un mètode de comportament obligatòriament mentres que *onStart()* és opcional i que el mètode *onStart()* dins d'un comportament s'executa una vegada mentres que *_process()* s'executa de manera repetitiva fins que es deté l'agent. Esta combinació de mètodes dins d'un comportament permet la inicialització de variables que després s'aniran modificant contínuament fins que l'agent de amb aquest comportament deixi de funcionar o es desconnecta de la plataforma multi-agent.

Els comportaments que poden utilitzar els agents intel·ligents són del tipus *Cyclic*, *Periodic*, *One-Shot*, *Time-Out*, *Event* i *Finite-State-Machine*. El comportament *Cyclic*, tal com ho diu el seu nom, s'utilitza per a realitzar tasques repetitives de manera indefinida. Es tracta del mètode de comportament més simple, format pels dos mètodes mencionats anteriorment. Partint d'aquest comportament es basen la resta. El comportament *Periodic* funciona igual que el *Cyclic*, amb la diferencia de que es pot determinar un temps d'espera entre diferents execucions del mètode *_onTick()*, que seria l'equivalent al de *_process()* en el tipus *Cyclic*. La duració de la pausa entre dos iteracions es determina en el moment que es crea l'instancia del comportament.

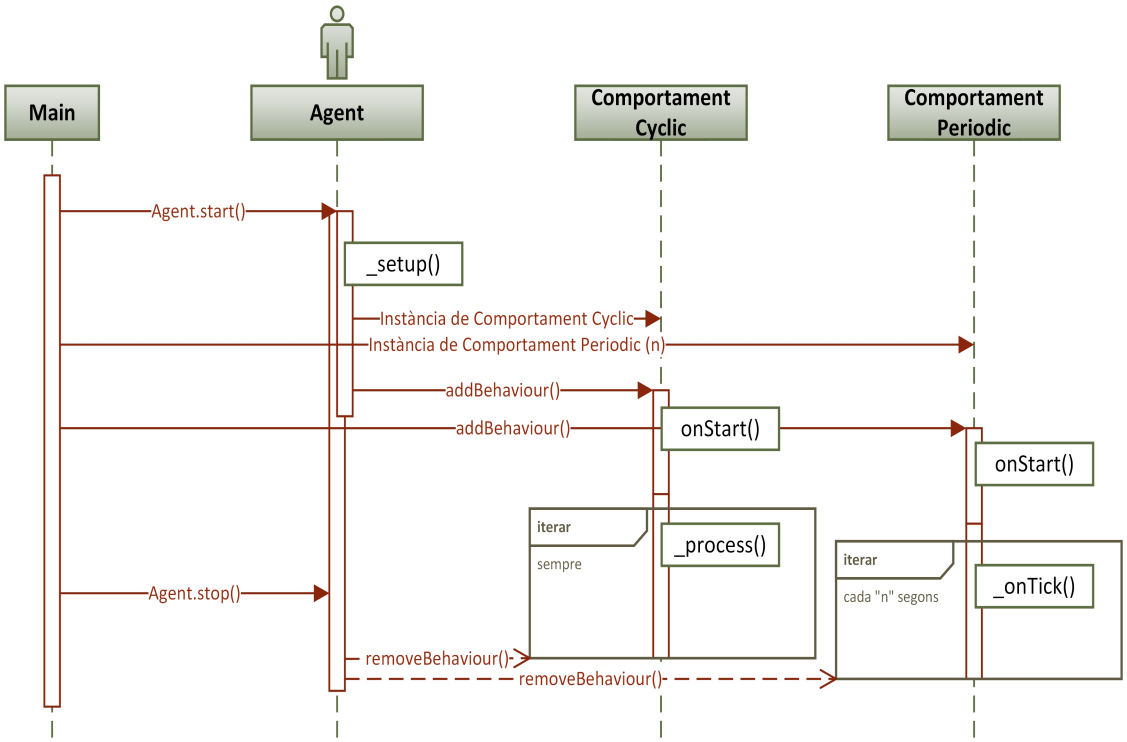

Fig. 3. Execució dels entorns *Cyclic* i *Periodic*

En el diagrama de seqüència anterior(Fig. 3) es mostra un exemple d'execució dels comportaments *Cyclic* i *Periodic,* on de forma paral·lela, el *script* i l'agent creen diferents instàncies dels comportaments i els afegeixen a l'agent registrat. Una vegada registrats, les instàncies de comportament executen també en paral·lel entre ells, el mètode *onStart()*, seguit pel de *_process(),* per al cas del tipus *Cyclic,* i el mètode *_onTick()* per al tipus *Periodic*. Aquests dos últims mètodes s'executaran contínuament fins que es decidisca parar a l'agent, que s'encarregarà de remoure les seues instàncies de comportament abans d'aturar-se.

Amb el cas de *One-Shot*, el comportament només s'executa una vegada i conté un tercer mètode opcional, *onEnd()*, que s'utilitza quan la instància del comportament vaja a finalitzar, de manera que pot ser útil per a realitzar tasques esporàdiques. Respecte al tipus de comportament *Time-Out*, es pot descriure com una variació del tipus *One-Shot* amb la funcionalitat de parar la seua execució durant un cert temps, declarat en el moment d'instanciar el comportament a l'agent. En aquest tipus, el mètode *_process()* està substituït pel de *timeOut()*, que s'executa després d'haver passat el temps d'espera, seguit pel de *onEnd()*.

El següent diagrama(Fig. 4) mostra el funcionament dels comportaments *One-Shot* i *Time-Out*, que en la forma de crear-los i afegir-los a un agent és idèntic al dels mètodes de comportament mencionats anteriorment. La diferència es troba en que es poden compondre de tres mètodes que es van executant un darrere d'altre directament, excepte en el cas del tipus *Time-Out*, que espera abans d'executar el mètode *timeOut()* uns segons, determinats en el moment d'afegir la instància a l'agent

registrat. També està la diferència de que l'agent no necessita remoure aquestes instàncies quan es deté, ja que s'eliminen de la llista de comportaments actius de l'agent quan finalitzen la seua execució.

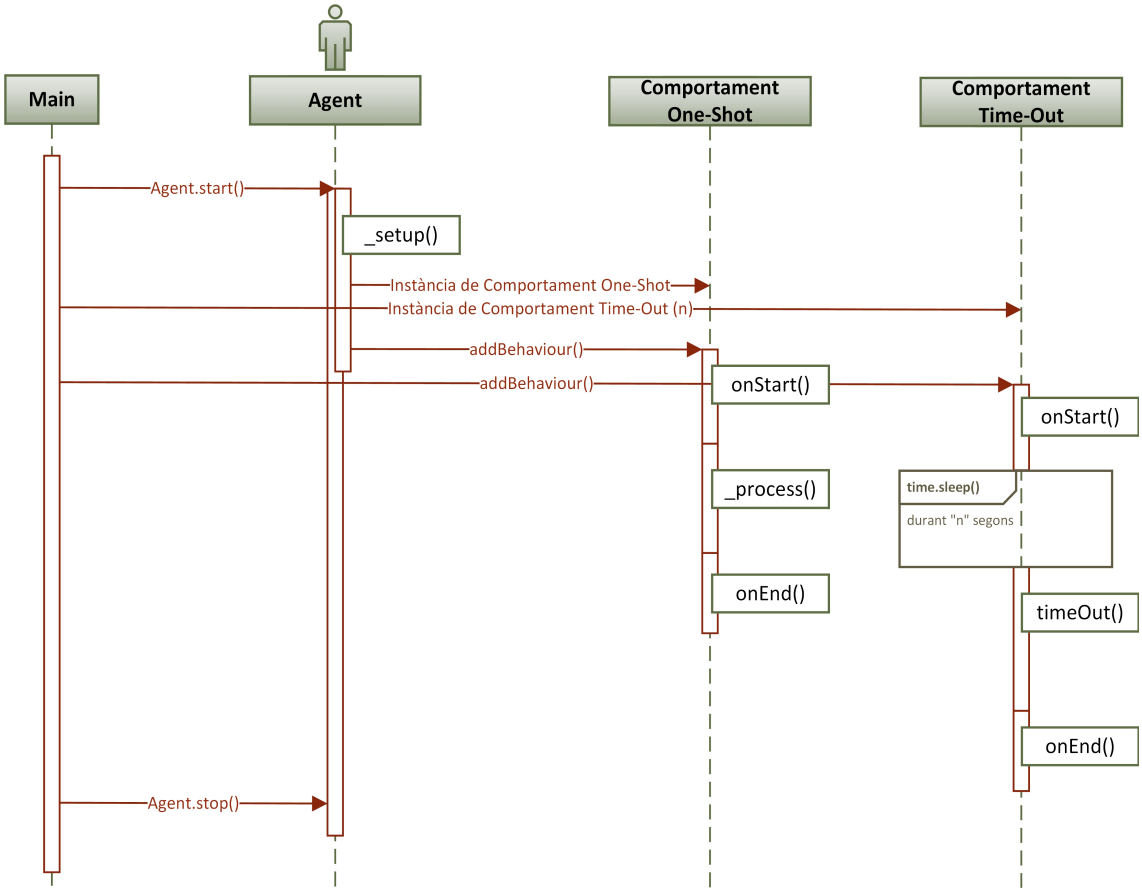

Fig. 4. Execució dels entorns *One-Shot* i *Time-Out*

El tipus de comportament *Event* és també paregut al de *One-Shot* respecte al fet de que només s'executa una volta, però es caracteritza pel cas de que no comença a executar-se després d'instanciar-lo i afegir-lo a l'agent, sinó que s'executa quan l'agent al que s'instancia rep un missatge en concret. Per a poder instanciar un comportament del tipus *Event*, cal afegir també una plantilla amb les característiques necessitaries, que consisteixen en els paràmetres que ha de tindre el missatge que el desencadena.

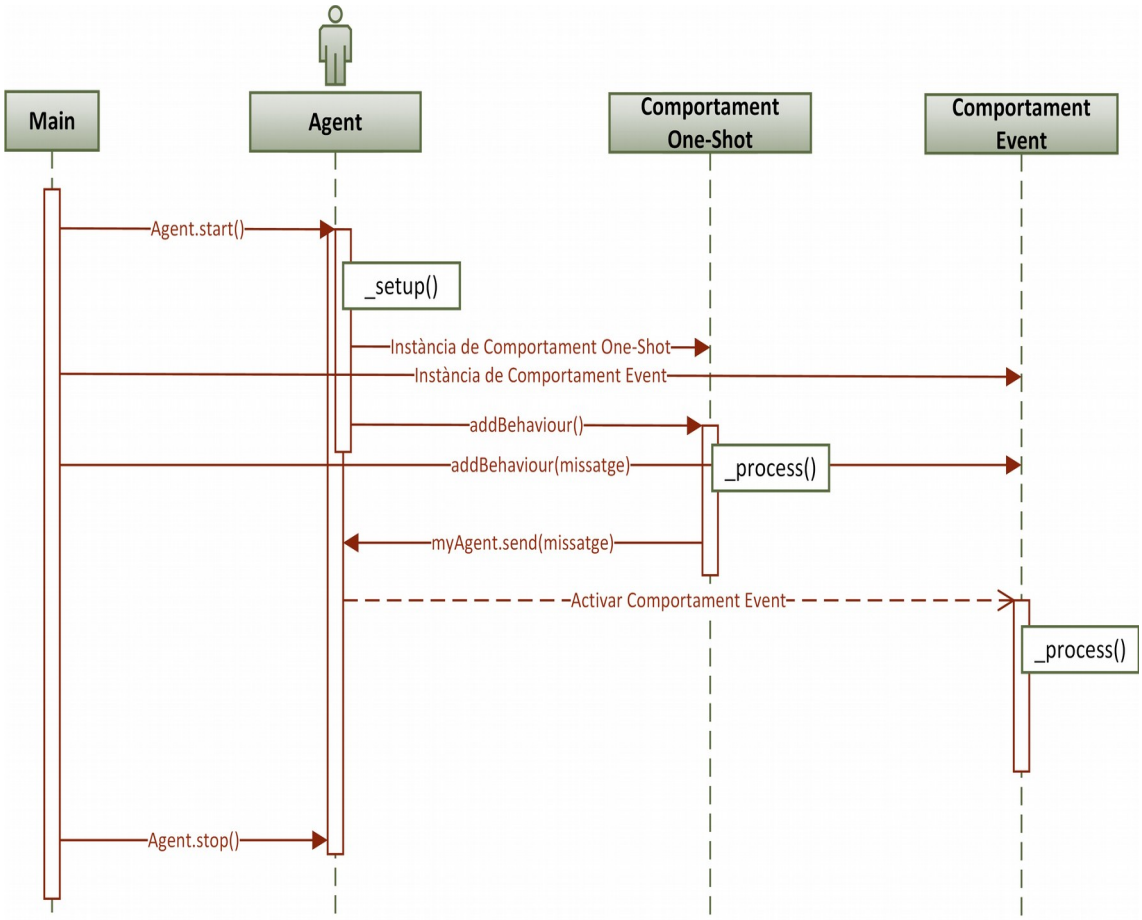

Fig. 5. Execució de l'entorn *Event* amb un entorn *One-Shot*

En el diagrama de seqüència d'abans(Fig. 5) es visualitza com funciona el comportament *Event*, on l'agent registrat en la plataforma crea i afegeix la instància del tipus *One-Shot* i el *script* afegeix el de tipus *Event* a l'agent. Mentres que el mètode *One-Shot* es va executant, el comportament *Event* no s'activa fins que l'agent al que està afegit reba un missatge en concret, determinat anteriorment durant la creació de la instancia. Una vegada que la instància *One-Shot* envia a l'agent el missatge, l'agent activa l'altre comportament per a que s'execute. Per a aquest exemple s'han obviat els mètodes *onStart()* i *onEnd()* per als diferents mètodes de comportament, ja que les seues implementacions són opcionals.

El comportament del tipus *Finite-State-Machine* es pot definir com un conjunt de comportaments que representen els distints estats d'una màquina d'estat finit, on el comportament d'un agent va canviant conforme es va transitant entre els diferents estats. Tal com està format una màquina d'estat finit, aquest tipus es caracteritza per un comportament inicial, un comportament final, un conjunt d'intermedis i un grup de transicions unidireccionals entre comportaments.

El mètode del tipus *Finite-State-Machine* es declara amb un conjunt de comportaments com els mencionats anteriorment i un conjunt de mètodes que ja venen definits en la classe *FSMBehaviour()*, que fa referència al mètode que crea màquines d'estat finit.

Els mètodes que s'han d'utilitzar per a crear una màquina d'estats són *registerState()* i *registerTransition()*, on respectivament serveixen per a registrar els diferents estats amb distints identificadors junt al nom de cada comportament que el formen, i per a registrar les diferents transicions entre dos estats, representats amb els identificadors tant d'estat com de transició. També estan els mètodes *registerFirstState()* i *registerLastState()* per a registrar el primer i l'últim comportament dins de la màquina d'estats.

En cada comportament intern s'utilitza la variable *_exitcode* per a definir la transició que es pot utilitzar per a canviar d'estat, representat amb un identificador que ha d'estar registrat anteriorment en la creació de la màquina d'estat finit.

Anteriorment s'havia mencionat que cada agent intel·ligent de SPADE conté un sistema de tramitació de missatges per a poder comunicar-se dins de la plataforma mitjançat el protocol Jabber. Per a que un agent es comunique amb un altre, primer es crea l'identificador de l'agent a qui es vol enviar un missatge, mitjançat un objecte de la classe AID, que permet identificar agents registrats en la plataforma. Aquest objecte permet representar l'agent receptor mitjançat el nom de l'agent i el conjunt de direccions que utilitza per comunicar-se, que normalment el nom és el propi JID amb el que s'ha registrat en la plataforma i el conjunt de direccions normalment està format només amb la direcció XMPP del canal de comunicació amb la plataforma.

Per a definir un missatge, existeix en SPADE la classe *ACLMessage* per a crear instàncies de missatges que compleixen l'estàndard FIPA i que permet generar-los amb diferents propietats fàcils d'extreure com el tipus de missatge, l'ontologia, el llenguatge del contingut, el receptor i el contingut del missatge. Cal tindre en compte que la propietat del receptor del missatge s'ha d'especificar amb un objecte de la classe AID, tal com s'ha explicat anteriorment. Una vegada creat el missatge, l'agent emissor pot enviar-lo utilitzant el mètode *send()*, definit internament en qualsevol agent, al que es pot accedir des d'un comportament instanciat a l'agent.

Per l'altre costat, per a que un agent reba un missatge cal que tinga un comportament on s'utilitze el mètode *_recieve()*, on es pot definir si es vol bloquejar l'execució de l'agent receptor de manera indefinida o durant un temps abans de rebre el missatge. Quan un agent rep un missatge d'aquesta manera, obté un objecte de la classe *ACLMessage*, amb les característiques que l'emissor haja declarat. A part, també és necessari crear una plantilla de missatge que s'ha d'afegir a l'entorn encarregat de rebre missatges quan es cree l'instancia. Aquesta plantilla es pot modificar per a afegir restriccions respecte al tipus de missatges que es vol que reba l'agent receptor, impedint que ho reben la resta d'agents registrats en la mateixa plataforma, que també tenen comportaments de receptor.

En el següent diagrama(Fig. 6) es mostra com dos agents es registren en la plataforma de SPADE i s'afegeixen a cadascun d'ells una instància de comportament distint del tipus *One-Shot*, on un s'implementa per a enviar un missatge i l'altre per a rebre'l. En el cas de l'agent receptor, l'execució del seu comportament es deté fins que reba el missatge o passe un temps, definit en el mètode *_receive()*, dins de *_process()*.

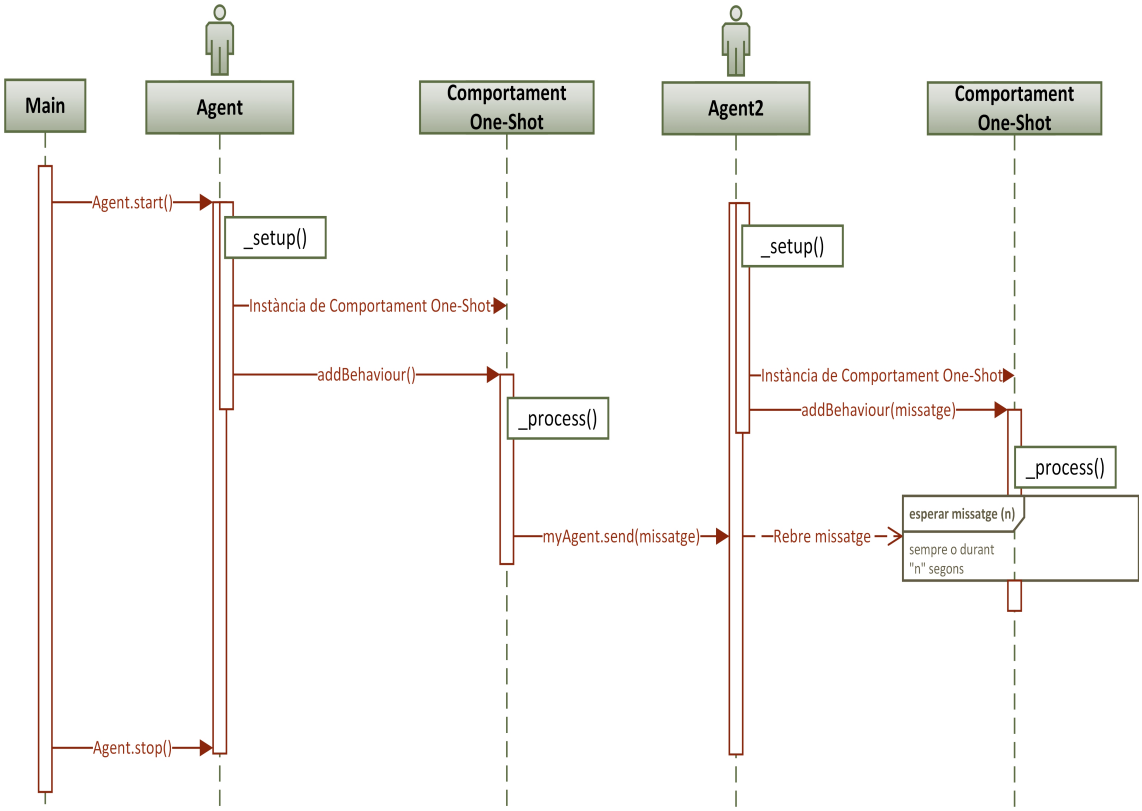

Fig. 6. Exemple d'emissor i receptor de missatges

El sistema d'administració d'agents o AMS és un servei implementat amb un agent de plataforma, que es crea directament al mateix temps que s'inicia la plataforma. Permet administrar, identificar i cercar altres agents en la plataforma, complint l'estàndard FIPA. També registra agents abans de que comencen a actuar i des-registra els agents una vegada s'hagen detingut. A part ofereix serveis als agents respecte a l'obtenció d'informació de la plataforma on es situen, la cerca d'altres agents i la modificació de la informació pròpia de cadascun d'ells en el AMS.

Per a la cerca d'informació de la plataforma, només cal que un agent utilitze el seu mètode *getPlaformInfo()* per a obtindre-ho en el llenguatge semàntic FIPA-SL. En el cas de buscar un altre agent, cal crear abans una plantilla amb la descripció de l'agent a buscar mitjançat una instància *AmsAgentDescription* de la classe *AMS*, on es poden afegir les restriccions de l'identificador d'agent, el seu propietari i un valor d'estat. Una vegada preparada la plantilla, un agent utilitza el seu mètode *searchAgent()* afegint la plantilla com a argument, de manera que el resultat que s'obté és una llista dels agents que complixen amb les restriccions, també expressada en el llenguatge FIPA-SL.

Finalment per a modificar la informació desada en el sistema d'administració d'agents cal seguir les mateixes instruccions que abans, amb la diferència de que en comptes de cercar un agent, s'utilitza el mètode *modifyAgent()* afegint-li la plantilla amb les propietats que es volen modificar.

La plataforma de SPADE també té un facilitador de serveis, implementat en un altre agent de plataforma, que s'utilitza com una guia on els agents poden publicar els serveis que ofereixen. Per a que un agent puga publicar-ne, primer necessita crear una instància *ServiceDescription* de la classe *DF* per a cadascun dels serveis que es vullguen publicar. En una d'aquestes instàncies es pot definir el seu nom, protocols, ontologies, llenguatges, propietari i una descripció. Després de tindre preparats les instàncies necessitaries, es crea una instància *DfAgentDescription*, també de la classe *DF*, on s'han d'afegir els serveis que es volen publicar. Finalment es publica els serveis d'un agent utilitzant un mètode propi dels agents, que és *registerService()* amb l'instancia de la col·lecció de serveis com a argument. Per a la modificació, inserció o eliminació de diferents serveis, l'agent ha de tornar a declarar de nou tota la llista de serveis amb els seus canvis realitzats, de manera que un agent només pot tindre registrats una sola col·lecció de serveis en la plataforma.

La cerca de serveis dels agents d'una plataforma és similar a la cerca d'agents mitjançat el sistema d'administració d'agents. Primer es crea una instància de *ServiceDescription* amb les restriccions que es vullguen complir. Després s'afegix aquesta instància a una altra del tipus *DfAgentDescription* i finalment s'utilitza el mètode *searchService()*, propi de l'agent, amb la instància mencionada anteriorment com a argument. El resultat que s'obté és una llista del conjunt de serveis publicats pels agents de la plataforma que compleixen amb les restriccions declarades en la cerca.

Anteriorment s'havia mencionat que SPADE permet la implementació d'agents amb una arquitectura BDI, que en un principi no s'utilitzarà per a complir amb els objectius d'aquest projecte. Aquesta està formada per un conjunt d'elements indispensables en un sistema BDI com ja s'ha explicat abans, que són les creences, els objectius, els serveis i els plans. Una creença és una peça d'informació de la base de coneixements d'un agent, l'objectiu és la informació o creença que vol assolir un agent, el servei és una intenció o acció que permet generar noves creences dins d'un agent a partir d'anteriors que es troben en la mateixa base de coneixement, i el pla és una seqüència de serveis que permeten arribar a un objectiu partint d'un conjunt de creences inicials.

En el model BDI de SPADE, les creences es componen mitjançat tuples amb la informació i el seu tipus, que s'emmagatzemen en la base de coneixements d'un agent amb el mètode *addBelieve()*. Els serveis o intencions es registren en el facilitador de

serveis junt als seus paràmetres, que són la informació d'entrada, la informació d'eixida, i les condicions de creences prèvies i posteriors. Quan es registra el perfil d'un servei, aquest ha de contindre un mètode, definit anteriorment, que permet afegir noves creences a un agent quan l'utilitze.

El pla també es defineix de manera semblant a un servei, amb els seus canals d'entrada i eixida d'informació, les seues condicions de creences i la seqüència de serveis a utilitzar. En el cas dels objectius, consisteixen en instàncies de la classe *Goal,* on s'ha d'indicar el conjunt de creences que es volen cercar. També requereix la implementació del funcionament que s'executarà quan s'aconseguisca l'objectiu, mitjançat un mètode que s'ha d'incloure com un paràmetre del mètode *setGoalCompletedCB()*. Una vegada definit l'objectiu, s'afegeix aquest a un agent amb la funció *addGoal()*.

Partint del funcionament explicat de la plataforma SPADE, es tractarà de crear l'agent REST amb diferents comportaments i un servei web, amb el qual es podran realitzar comunicacions amb sistemes externs. El funcionament d'aquest servei web i com s'implementa l'agent REST s'explicaran en els següents subapartats.

3.2 Incorporació del mòdul

En esta secció s'explica com s'ha incorporat en la plataforma de SPADE el mòdul amb serveis web amb arquitectura Restful, explicant també com s'implementa aquest servei, l'agent que l'utilitzarà i com interactuarà amb altres agents, tant de la mateixa plataforma com d'una altra externa. La incorporació d'un mòdul amb funcionalitats de l'arquitectura Restful, permetrà al sistema multi-agent de SPADE poder comunicar-se amb altres plataformes, com un protocol de comunicació alternatiu als MTPs de XMPP, P2P, HTTP i SIMBA, que ja estan implementats en la plataforma.

El *framework* web Flask permet crear un servei web de REST com si es tractés d'un servidor, que activa les seues funcions depenent de les peticions que va rebent en una adreça IP i port, i respon amb un objecte als clients, que pot contindre informació de diferents tipus. Per al seu funcionament, només cal declarar una variable, on s'emmagatzema el servei web, al que s'aniran afegint diversos mètodes amb la sol·licitud corresponent que necessita realitzar un client per a activar-lo. Una vegada afegit tots el mètodes necessaris, es crida al mètode propi *run()* amb els paràmetres necessaris per a utilitzar una adreça IP i un port en concret com a adreça, on es poden realitzar peticions.

En la declaració dels mètodes que pot executar el servidor web, Flask permet afegir un decorador, on es pot assignar la ruta necessària per a activar-lo, indicant la URI i el

tipus de peticions com a restriccions de la seua execució, que poden ser per exemple els tipus GET, POST o PUT. La URI que fa referència a un mètode pot ser estàtica o dinàmica, de manera que el mètode només pot admetre una URI o moltes amb un cos semblant. En el cas de les URIs dinàmiques, se pot afegir variables dins de l'enllaç, que poden estar especificades si han de ser d'un tipus en concret, com poden ser variables numèriques o cadenes de caràcters. Les variables que s'especifiquen d'aquesta forma després han d'estar declarades com a arguments en la funció que fan referència, de manera que el mètode retorne informació segons el contingut de les variables. A part, aquestes funcions estan obligades a tornar un objecte abans de tancar la connexió amb un client que haja fet la petició, de manera que no es poden realitzar comunicacions asíncrones entre client i servidor. La informació que retornen aquests mètodes normalment són cadenes de caràcters, codis HTTP de l'estat del resultat de la petició o pàgines web en HTML si es tractés d'un servidor de pàgines web.

També es poden afegir altres tipus de decoradors a aquestes funcions, com poden ser requisits d'accés, de manera que només tenen accés alguns usuaris registrats al servidor, o capturadors d'errors. Respecte als capturadors d'error, es pot implementar un mètode a part que s'execute cada vegada que es detecte un error dins de la comunicació entre client i servidor, com per exemple l'error de variables no vàlides en la URI realitzada pel client, de problemes interns dins del servidor o d'accessos denegats al client.

Tenint en compte que Flask funciona per a rebre peticions i respondre-les, el modul es reduiria a un servei web que permet comunicar-se amb altres plataformes multi-agent com si es tractés d'un servidor. Per a que el modul de REST també puga realitzar peticions a altres plataformes, s'ha utilitzat la llibreria *Requests*[21]. Aquesta llibreria de codi obert per a Python permet realitzar peticions o consultes a qualsevol direcció web, com si l'aplicació que el gastés es tractara d'un client que vol navegar per diverses webs. Permet realitzar qualsevol tipus de petició, com per exemple els tipus GET, PUT, POST, OPTIONS i DELETE, i retorna un objecte amb diferents propietats, com poden ser el contingut de l'objecte, el codi HTML de la resposta a la petició realitzada o la capçalera de l'objecte retornat pel servidor. En cadascuna d'aquestes possibles propietats se suposa que Flask s'ha d'encarregar de retornar-les a cada client, depenent de com s'haja implementat l'API del servei web.

Per a afegir el mòdul de Restful a la plataforma, s'ha pensat que entre els diferents agents de la plataforma hi haja un que s'encarregue de les comunicacions externes a la plataforma, seguint la idea de crear-lo com un agent amb funcions úniques, com són l'administració d'agents i el facilitador de serveis. L'agent amb aquest mòdul es comunicaria amb els agents de la mateixa plataforma mitjançat el protocol XMPP,

mentres que se comunicaria amb plataformes externes seguint el protocol de REST. En el següent esquema es mostra la representació de com es comuniquen internament i externament els agents en un sistema multi-agent de SPADE partint de l'agent que es vol implementar(Fig. 7):

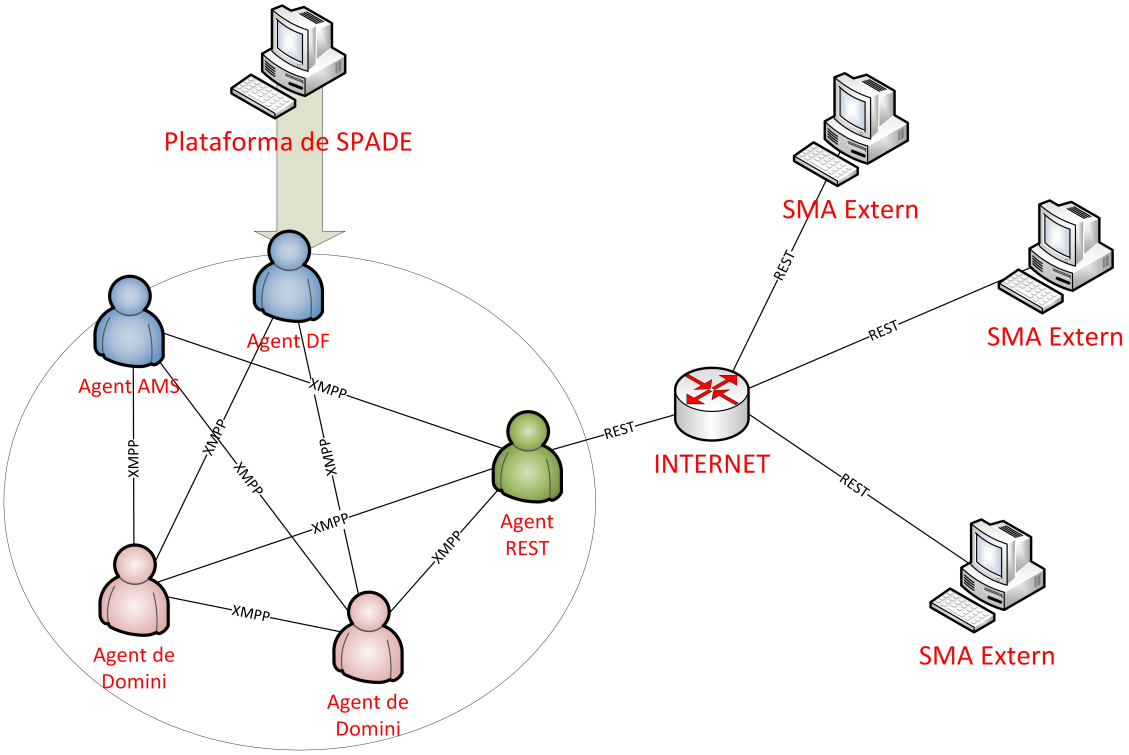

Fig. 7. Representació de les comunicacions en la plataforma SPADE

Amb la inserció d'aquest mòdul mitjançat un agent a la plataforma, la representació per capes de SPADE quedaria de la següent manera, representant també on es realitza la comunicació amb un altre sistema(Fig. 8):

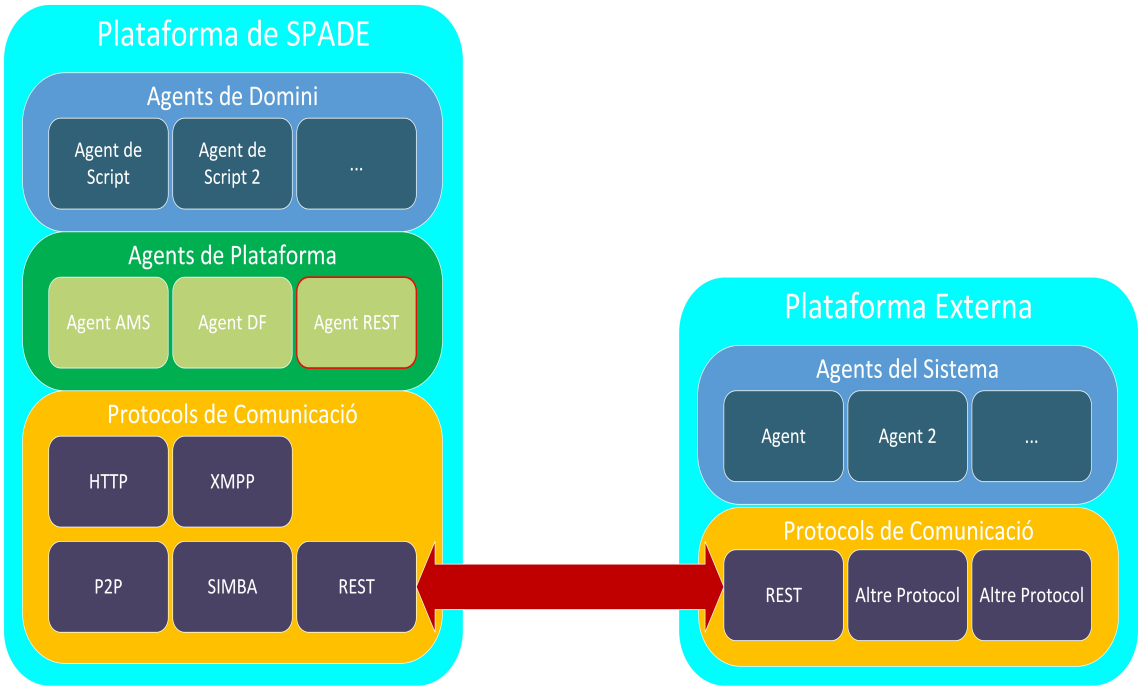

Fig. 8. Representació en capes de la comunicació entre una plataforma de SPADE i una altra externa

El diagrama de seqüència que es mostra a continuació(Fig. 9) explica com es comunicaria SPADE amb una plataforma externa mitjançat l'agent que s'encarrega del protocol de REST, on es mostra primer el cas en que la plataforma SPADE fa d'emissor i la externa de receptor, i un segon cas on es canvien aquests càrrecs:

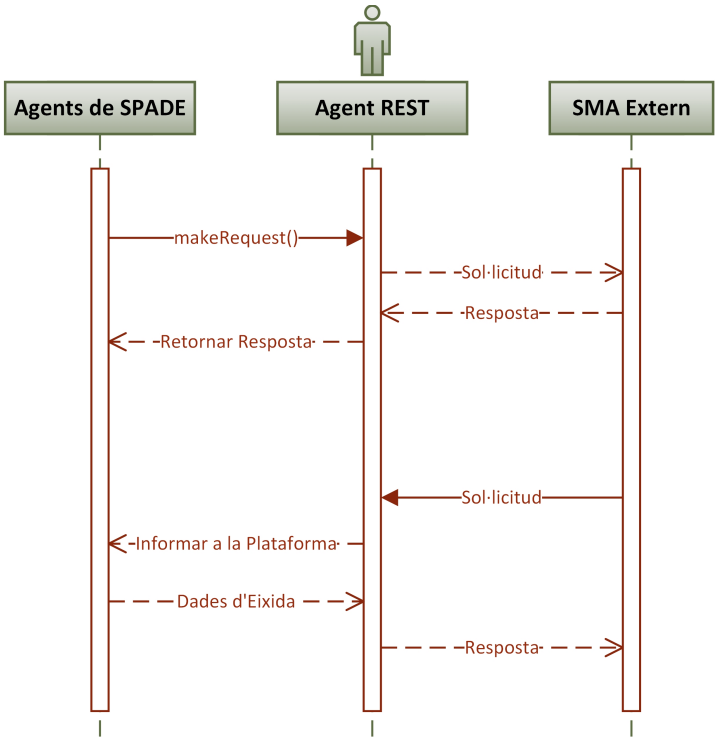

Fig. 9. Funcionament de l'agent REST

Amb la informació de com funciona Flask i on es situaria el mòdul web de REST, ja es té una idea de com s'ha d'implementar l'agent REST dins d'un sistema multi-agent de

SPADE. La forma en la que s'implementa aquest agent s'explica en el següent subapartat.

3.3 Implementació

Anteriorment s'ha explicat com funciona el servei web de REST amb Flask, de manera que rep sol·licituds i retorna les dades necessàries. En aquest punt s'explicarà com rep l'agent intel·ligent a implementar aquestes peticions i tracta de recopilar la informació necessària mitjançat la cooperació dels altres agents que formen el sistema en el que es troba, per a després donar-la al servei web, que ho envia al client.

Per començar, s'ha de tindre en compte que el servei web creat amb Flask i l'agent al que es vol afegir aquest servei realitzen les seues tasques en fils d'execució diferents, de manera que es necessari implementar un sistema de comunicació entre aquests dos per a que puguen sincronitzar-se quan l'agent obtinga la informació de les peticions que va rebent el servei web, i quan el servei sàpiga si ja està preparada la informació que ha recopilat l'agent per a retornar-la al client. S'ha de tindre en compte també que no es pot afegir aquest servei web dins de l'agent com si es tractés d'un comportament, degut a que existeixen problemes amb la llista d'arguments dels mètodes que s'implementen amb variables obtingudes a partir d'URIs, si al mateix temps es tracten com a mètodes d'una classe.

Per a realitzar aquesta comunicació entre agent i servei, s'ha dissenyat dins de l'agent una llista de peticions(Fig. 10), on es van afegint les sol·licituds que rep el servei web i on l'agent tracta de completar-les afegint les dades necessàries. La llista de peticions consisteix en un diccionari, on la clau de cada sol·licitud és un identificador numèric únic i el valor és una llista de paràmetres. Aquesta llista de paràmetres està formada com a mínim per un valor d'estat de la petició, el tipus de petició i un camp buit, on es guarda la informació que cal retornar a un client. El valor d'estat permet comprovar si una petició encara no ha sigut analitzada per l'agent, està comprovada per l'agent però encara està en procés d'obtindre la informació a retornar, o ja té la informació necessària per a enviar-la al seu client corresponent. El paràmetre referent al tipus de petició permet a l'agent gestionar quina informació cal obtindre, depenent de si es tracta per exemple d'una petició GET, PUT o POST. A la llista de paràmetres es poden afegir més camps opcionals en el casos on la sol·licitud que s'obté conté arguments addicionals, que permet especificar les dades que requereix la tasca que ha de realitzar la plataforma d'agents, sempre que l'agent que es comunica amb el servei Restful sàpiga com gestionar aquests paràmetres.

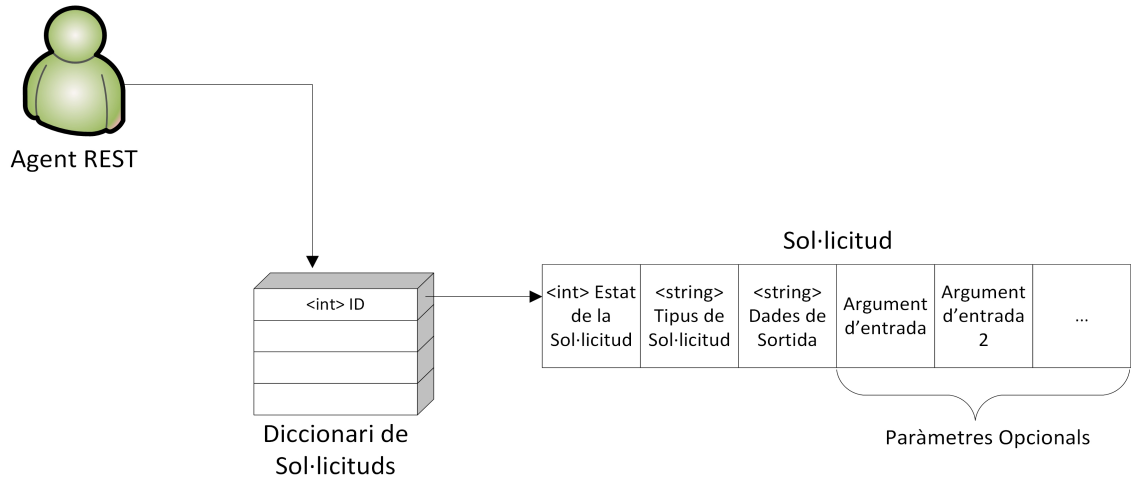

Fig. 10. Representació de la petició dins de l'Agent REST

Quan el servei web rep una petició, crea un diccionari amb els camps de propietats necessaris, l'afegeix a la llista de sol·licituds de l'agent REST i espera fins que comprove que el valor d'estat de la petició demostre que ja s'ha obtingut la informació, per a després accedir al contingut del diccionari i enviar-ho al client.

L'agent REST es compon per diferents comportaments, que són el mètode receptor de peticions del servei web, el comportament receptor d'informació de la plataforma i el mètode de comportament que permet realitzar peticions a plataformes externes. El comportament receptor de sol·licituds del servei s'encarrega que realitzar les comunicacions amb els diferents agents de la plataforma on es troba depenent del tipus de sol·licituds que rep. Aquest comportament, que és del tipus *Cyclic*, comença buscant en la seua llista de peticions, de manera que si troba un diccionari que no ha sigut comprovat per ell, modifica el seu valor d'estat, determinant que la sol·licitud està en procés de buscar una resposta a retornar, i comprova el seu tipus de petició. Depenent del tipus de sol·licitud, l'agent REST crearà un missatge o altre, tenint en compte també de si la sol·licitud conté camps de propietats extra, i l'enviarà a l'agent de la plataforma que li corresponga.

El mètode de comportament per a obtindre informació per part de la plataforma, que també és del tipus *Cyclic*, permet rebre missatge respecte a les respostes de les peticions, generades per la resta d'agents intel·ligents, de manera que obté la petició incompleta de la seua llista i l'afegeix la informació obtinguda, canviant també el valor d'estat del diccionari per a determinar que la sol·licitud ja està preparada per a retornar al client. Per últim, el comportament que realitza peticions externes funciona a partir de missatges que va rebent per part dels altres agents de la mateixa plataforma SPADE, on partint del contingut del missatge obtingut s'especifica la petició que es vol realitzar, mitjançat la llibreria de Python *Requests*. Una vegada obtingut la informació per part de la plataforma externa, s'emmagatzema en un missatge amb les propietats necessàries i es retorna a l'agent que ho ha demanat.

En la interacció entre el servei Restful i l'agent REST també s'utilitza un sistema de control de concurrència, per a evitar condicions de carrera entre els dos. El sistema de control implementat permet detindre l'execució del mètode que activa Flask quan rep una sol·licitud, de manera que el bloqueja fins que l'agent REST determine el valor d'estat de la sol·licitud creada en la llista com a completada. La raó d'aquest bloqueig està en que existeix una condició de carrera en el cas de que Flask intente retornar informació al client abans de que s'haja recopilat i rebut per part de l'agent REST. En el següent diagrama de seqüència es pot vore com s'executen i interactuen l'agent REST i el servei Restful implementat en Flask, on també es representa la interacció entre l'agent REST amb els altres agents de la plataforma i entre el servei web amb un sistema multi-agent extern(Fig. 11):

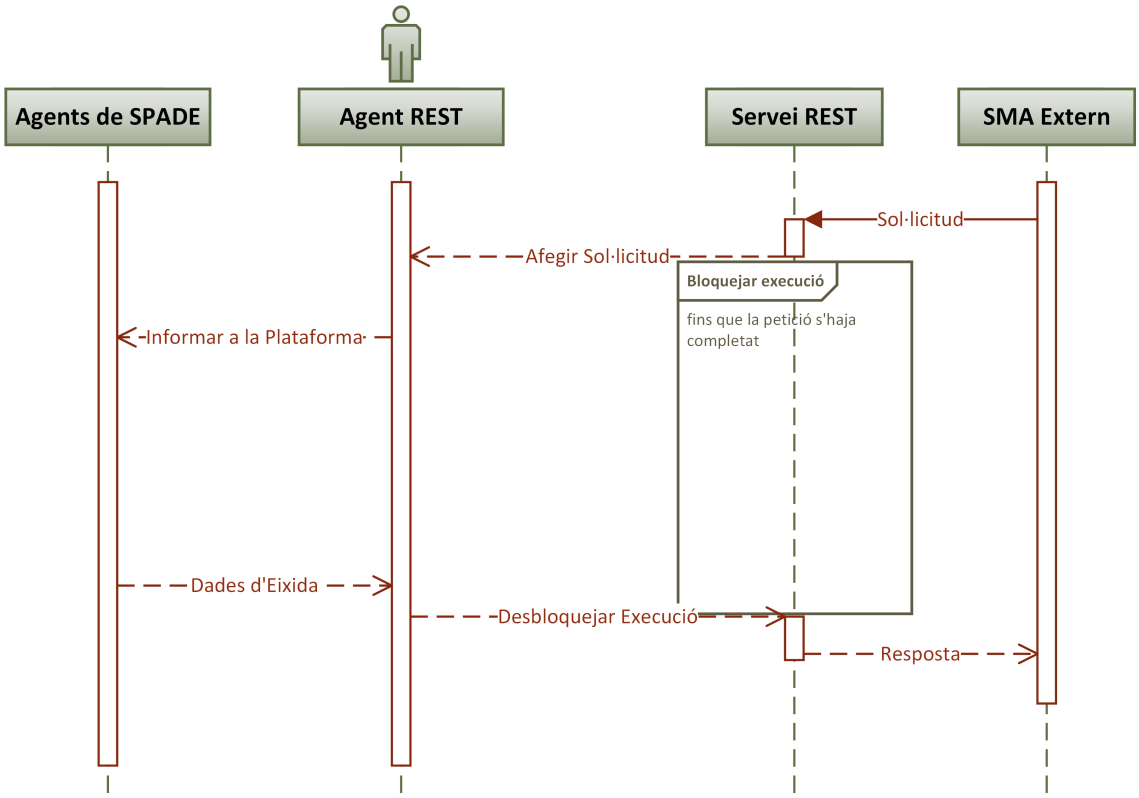

Fig. 11. Diagrama de seqüència de la comunicació entre l'agent REST i el servei web

Encara que aquest protocol d'interacció permeta gestionar les peticions, comunicar-se amb la resta de la plataforma i retornar al client la informació desitjada, existeixen casos en que la plataforma SPADE es bloqueja indefinidament en mitat de l'obtenció d'informació, de manera que el servei Restful es queda també bloquejat indefinidament esperant una resposta per part de la plataforma. Per a resoldre aquest problema, s'ha afegit al sistema de control de concurrència una condició d'espera, de manera que si no s'ha realitzat la petició en un temps definit, s'elimina la petició de la llista de diccionaris, es reinicia l'execució dels comportaments dels agents afectats d'aquesta manera i s'envia al client un missatge d'error, amb el codi d'estat HTTP corresponent al tipus de problema.

Un segon cas a tindre en compte en aquest protocol és el de l'acceptació per part de Flask de peticions dinàmiques amb variables dins de la URI o paràmetres necessaris que la plataforma de SPADE o el servei Restful són incapaços d'entendre per errors de sintaxi, de manera que poden afectar al funcionament de la plataforma. Aquest problema es resol dins del mateix mètode del servei web que s'activa al rebre una sol·licitud abans d'afegir-la a la llista de sol·licituds de l'agent REST, de manera que es comprova si totes les variables i paràmetres importants són del tipus adequat i representats de manera correcta. Si alguna d'aquestes dades no està ben representada, es respon directament al client amb l'error detectat.

Al igual que aquests problemes poden aparéixer en la plataforma de SPADE, també podrien ocórrer en altres de externes i que al mateix temps no ho tenen resolt, de manera que també existeix el cas de que la nostra plataforma es quede bloquejada o actue de manera incorrecta quan s'intente realitzar una petició a una plataforma externa i no retorna una resposta. Per a aquest cas, la llibreria *Requests* permet definir un temps d'espera de la resposta, de forma que si s'excedeix aquest temps d'espera, es talla la connexió amb la plataforma externa i es retorna a l'agent que demanava la petició un missatge d'error en la comunicació. També existeix el cas en el que s'intente realitzar una petició a una direcció no vàlida, on se capturaria l'excepció de la mateixa forma que en el cas anterior.

Respecte a la utilització del comportament de realitzar sol·licituds a sistemes o serveis externs, com que pot ser demanat per qualsevol agent del domini del sistema, s'ha pensat en que el comportament l'afegeix a l'agent REST aquell agent que vullga utilitzar-lo. Amb aquesta idea, qualsevol agent pot demanar fer una petició a l'agent REST sense realitzar una cua d'espera de totes les sol·licituds que s'han de realitzar encara, de manera que cadascuna d'aquestes es realitzen en paral·lel. La paral·lelització d'aquest comportament permet reduir el temps necessari per a realitzar una petició externa, a part de que no bloqueja el funcionament de diversos agents del sistema.

Com a últim detall que s'ha tingut en compte respecte a la implementació de l'agent REST, és la utilització de codis d'estat HTTP en les respostes de cada sol·licitud, tant per part del servei Restful com per part de l'agent Rest al realitzar peticions web, en el cas de que es genere qualsevol error no relacionat amb el servei extern. La utilització de codis d'estat permet especificar el tipus de resposta que s'ha generat seguint un estàndard aprofitat per moltíssimes aplicacions web, de manera que qualsevol pot entendre com s'ha realitzat la petició. El conjunt de codis es divideix en cinc grups:

Respostes informatives. Aquests tipus de codis demostren que el servidor ha rebut la petició i que el client pot continuar enviant informació. Es distribueixen entre la primera centena de números.

Peticions correctes. Estos codis indiquen que la sol·licitud realitzada s'ha rebut correctament, entés i acceptada. Es situen en el segon centenar de la llista de codis.

Redireccions. Aquesta classe de codi avisa de que el client ha de realitzar una acció addicional per a completar la petició. La acció addicional ha de ser realitzada per un agent d'usuari. Els codis d'aquest tipus ocupen la tercera centena.

Errors del client. És el grup de codis que demostren que la petició realitzada conté sintaxis incorrecta o que no es pot processar en el servidor. Generalment fan referència a que el client no ha realitzat una sol·licitud de manera correcta. Es distribueixen en la quarta centena de números.

Errors del servidor. Aquests codis signifiquen que el servidor no ha pogut completar la sol·licitud del client, tenint en compte que la petició realitzada pel client és aparentment valida. Els codis es situen en el cinqué centenar de la llista.

Partint de les funcionalitats implementades en l'agent REST, només faran falta codis d'estat que demostren peticions correctes, sol·licituds errònies i resultats no desitjables per part del servidor. En la taula que hi ha a continuació es mostren els codis HTTP utilitzats, junt a la seua descripció i si el que realitza la seua notificació és el servei web al rebre les sol·licituds o l'agent Rest al realitzar una petició:

Codi d'estat	Component que ho notifica	Descripció
200	Servei web	La petició s'ha realitzat correctament.
201	Servei web	La petició referent a la inserció d'un nou recurs s'ha efectuat de manera correcta.
400	Servei web	La petició té errors de sintaxi.
404	Agent REST	La petició no existeix.
406	Servei web	La petició no s'admet en el servidor.
408	Agent REST	El client ha superat el seu temps d'espera de la resposta.
500	Servei web	El servidor ha trobat un error intern al realitzar la petició.

Fig. 12. Codis d'estat HTTP utilitzats

Respecte a les restriccions que té el servei web, en un principi no es pot decidir si incompleix o no alguna per determinar si és del tipus REST, perquè escasseja de funcionalitat com a API web i depén d'un sistema implementat sobre la plataforma SPADE. Partint del sistemes distribuïts dissenyats, que s'explicaran més endavant, es podrà comprovar si les APIs implementades en cadascun d'ells complixen amb els requisits d'una arquitectura REST. En el següent subapartat s'explica com funcionaria un sistema distribuït simple amb el mòdul REST que s'ha aconseguit implementar.

3.4 Exemple bàsic

A continuació s'explica com s'implementa amb codi un exemple de comunicació entre dos sistemes multi-agent de SPADE, realitzant un seguiments de les funcions que es van utilitzant durant l'execució de l'exemple, on una realitza una sol·licitud d'obtindre informació i l'altra tractarà de retornar una cadena de caràcters. Cadascun dels sistemes es componen de dos agents, que són l'agent REST i de domini, que realitza la petició o genera la informació a retornar.

Per part del sistema que realitza la petició només cal importar les llibreries de SPADE i *Requests*, a més de crear una variable global on s'emmagatzema l'adreça on s'executen els agents del sistema(Fig. 13).

```
import spade
import requests

spadeHost = ""
```

Fig. 13. Llibreries i variables globals del sistema emissor

La seua execució començaria amb la creació dels agents del sistema, afegint a l'agent REST els comportaments necessaris amb les seues plantilles de missatges amb els que permeten executar-los(Fig. 14).

```
spadeHost = "127.0.0.1"

a = MyAgent("agent@"+spadeHost, "secret")
rest = restAgent("rest@"+spadeHost, "secret")

aclt = spade.Behaviour.ACLTemplate()
aclt.setPerformative("request")
t = spade.Behaviour.MessageTemplate(aclt)

rest.addBehaviour(makeRequest(), t)

a.start()
rest.start()
```

Fig. 14. Inicialització del sistema emissor

En el cas del sistema que realitza la recepció de la sol·licitud, inicialment es declaren les llibreries de SPADE, de l'obtenció del temps actual i les necessàries de Flask per a crear el servei web i retornar missatges d'error en el cas de trobar errors en l'obtenció de la informació a enviar a l'emissor. Respecte a les variables globals, faran falta la que emmagatzema l'adreça del sistema multi-agent, el temps màxim que pot tardar el sistema a retornar peticions i la variable on s'emmagatzema el servei Restful(Fig. 15).

```
import time
import spade
from flask import Flask, abort

spadeHost = ""
timeout = 5
app = Flask(__name__)
```

Fig. 15. Llibreries i variables globals del sistema receptor

L'execució d'aquesta part de l'exemple seria igual que el demostrat anteriorment, amb la diferencia que s'instancien a l'agent REST els comportaments de comunicació amb el sistema per a resoldre les peticion rebudes i el de rebre per part del sistema la informació necessària per a determinar una sol·licitud com finalitzada i enviar-la al servei Restful. També s'inicialitza el servei web per a que comence a rebre peticions externes(Fig. 16).

```
spadeHost = "127.0.0.1"
a = MyAgent2("agent2@"+spadeHost, "secret")
rest = restAgent2("rest2@"+spadeHost, "secret")

aclt = spade.Behaviour.ACLTemplate()
aclt.setPerformative("complete")
t = spade.Behaviour.MessageTemplate(aclt)

rest.addBehaviour(petitionCompleted(), t)
rest.addBehaviour(RestBehaviour, None)

a.start()
rest.start()

app.run(host = "192.168.1.19", port = "5000")
```

Fig. 16. Inicialització del sistema receptor

Amb els dos sistemes inicialitzats, el sistema emissor comença executant el seu agent que vol realitzar una petició a l'altra plataforma. Per a realitzar-ho, envia un missatge a l'agent REST per a que ho faça(Fig. 17).

```
class MyBehav(spade.Behaviour.OneShotBehaviour):
        def _process(self):
            msg = spade.ACLMessage.ACLMessage()
            msg.setPerformative("request")
            msg.addReceiver(spade.AID.aid("rest@"+spadeHost,
["xmpp://rest@"+spadeHost]))
            self.myAgent.send(msg)
```

Fig. 17. Comportament per a demanar realitzar una petició externa

Partint d'aquest missatge, l'agent REST activa el seu comportament de realitzar sol·licituds(Fig. 18). Aquest comportament intentarà realitzar una petició utilitzant la llibreria *Requests* amb la direcció web on es troba el sistema multi-agent extern i declarant un temps límit d'espera. Si el sistema extern respon, s'emmagatzema el contingut de la resposta en una variable declarada al principi del comportament. En el cas de que es produïsca algun problema en la petició, es guarda un avis d'error en la variable mencionada abans. Una vegada realitzada la sol·licitud, es retorna a l'agent que ha demanat fer una petició un missatge amb el contingut obtingut.

```
class makeRequest(spade.Behaviour.Behaviour):
    def _process(self):
        msg = self._receive(block=True)
        result = ""
        try:
            r = requests.get('http://192.168.1.19:5000/',
timeout = 5)
            result = r.text
        except requests.exceptions.Timeout:
            result = "ERROR 408: timeout exception"
        except requests.exceptions.RequestException:
            result = "ERROR 404: request refused"
        msg2 = spade.ACLMessage.ACLMessage()
        msg2.setPerformative("agent")
        msg2.addReceiver(spade.AID.aid("agent@"+spadeHost,
["xmpp://agent@"+spadeHost]))
        msg2.setContent(result)
        self.myAgent.send(msg2)
```

Fig. 18. Comportament per a realitzar sol·licituds externes

Quan es realitza la petició, el sistema receptor activa aquella funció implementada en el servei web de Flask que accepta la URI realitzada per l'emissor(Fig. 19). El mètode que s'activa afegeix una nova petició a la llista pròpia de sol·licituds de l'agent REST del sistema i comença a esperar a que el sistema obtinga la informació necessària per a retornar-la a l'emissor. Si s'aconsegueix obtindre aquesta informació abans de que s'acabe el temps d'espera, el servei Restful retorna la informació i elimina la sol·licitud completada de la llista de peticions de l'agent REST. En el cas de que es supere el

temps límit d'espera, el servei reinicia l'execució de l'agent REST, descarta la sol·licitud del sistema i retorna a l'emissor l'error que s'ha produït.

```
@app.route('/', methods=['GET'])
def requestMethod():
    idp = rest.id
    rest.id += 1
    rest.petitions[idp] = [0, "GET", ""]
    endtime = time.time() + timeout
    remaining = endtime - time.time()
    while rest.petitions[idp][0] != 2 and remaining > 0.0:
        time.sleep(1)
        remaining = endtime - time.time()
    if rest.petitions[idp][0] == 2:
        response = rest.petitions[idp][2]
        del rest.petitions[idp]
        return response
    else:
        stopRestBehaviour()
        del rest.petitions[idp]
        startRestBehaviour()
        abort(500)
```

Fig. 19. Mètode per a rebre peticions amb Flask

Per a reiniciar l'execució de l'agent REST, es remouen aquells comportaments que s'han bloquejat per a després tornar-los a afegir a l'agent. En aquest cas només caldrà tindre en compte el comportament que revisa les peticions per a després comunicar-se amb la resta d'agents(Fig. 20).

```
RestBehaviour = restBehav()

def startRestBehaviour():
    RestBehaviour = restBehav()
    rest.addBehaviour(RestBehaviour, None)
    print "Rest Behaviour started"

def stopRestBehaviour():
    rest.removeBehaviour(RestBehaviour)
    print "Rest Behaviour stopped"
```

Fig. 20. Mètodes per a afegir i esborrar un comportament de l'agent REST

Una vegada que s'haja afegit una nova petició a la llista pròpia de l'agent REST, el comportament seu s'activa(Fig. 21) per a analitzar la sol·licitud i enviar-la amb el seu

identificador de petició a l'agent adequat per a que obtinga les dades necessàries. En aquest cas només es pot enviar a un únic agent.

```
class restBehav(spade.Behaviour.Behaviour):
    def _process(self):
        if len(self.myAgent.petitions)==0:
            time.sleep(1)
        else:
            for k in self.myAgent.petitions.keys():
                if self.myAgent.petitions[k][0]==0:
                    self.myAgent.petitions[k][0] = 1
                    if self.myAgent.petitions[k][1]=="GET":
                        msg = spade.ACLMessage.ACLMessage()
                        msg.setPerformative("agent2")
                        msg.addReceiver(spade.AID.aid(
"agent2@"+spadeHost, ["xmpp://agent2@"+spadeHost]))
                        msg.setContent(str(k))
                        self.myAgent.send(msg)
```

Fig. 21. Comportament on l'agent REST revisa les peticions i envia missatges al sistema SPADE

Quan l'altre agent del sistema reba el missatge per part de l'agent REST, intentarà obtindre la informació necessària i ho retornarà amb l'identificador que havia rebut, activant el comportament de recepció de missatges de l'agent REST per part del sistema de SPADE(Fig. 22). En aquest cas, l'agent del sistema retorna l'identificador de la sol·licitud amb una cadena de caràcters.

```
class MyBehav(spade.Behaviour.OneShotBehaviour):
        def _process(self):
            msg = self._receive(block=True)
            content = msg.getContent()
            msg2 = spade.ACLMessage.ACLMessage()
            msg2.setPerformative("complete")
            msg2.addReceiver(spade.AID.aid("rest2@"+spadeHost,
["xmpp://rest2@"+spadeHost]))
            msg2.setContent(content + "-HELLO WORLD")
            self.myAgent.send(msg2)
```

Fig. 22. Comportament on l'agent del sistema genera la informació a retornar

L'agent REST, al rebre el missatge en el seu comportament, completa la petició de la seua llista amb la informació rebuda, on selecciona la petició de la llista mitjançat l'identificador rebut i modifica el seu esta per a determinar que s'ha completat. Una vegada completada la sol·licitud, el mètode de Flask que esta encara bloquejat es desbloqueja per a retornar al sistema emissor la informació obtinguda(Fig. 23). En aquest cas consisteix en una cadena de caràcters.

```
class petitionCompleted(spade.Behaviour.Behaviour):
    def _process(self):
        msg = self._receive(block=True)
        content = msg.getContent().split("-")
        pid = int(content[0])
        self.myAgent.petitions[pid][2] = content[1]
        self.myAgent.petitions[pid][0] = 2
```

Fig. 23. Comportament de l'agent REST per a completar les sol·licituds

Una vegada rebuda la resposta a la petició del sistema emissor, l'agent REST retorna la resposta a l'agent que havia demanat realitzar la sol·licitud, tal com s'havia explicat anteriorment, de manera que l'agent del sistema que ho havia sol·licitat mostra pel terminal la resposta obtinguda del sistema extern(Fig. 24).

```
class MyBehav2(spade.Behaviour.OneShotBehaviour):
        def _process(self):
            msg = self._receive(True)
            if msg:
                print msg.getContent()
            else:
                print "No message received"
```

Fig. 24. Comportament per a mostrar pel terminal la informació obtinguda

L'execució d'aquest exemple es mostraria de la següent manera en els diferents terminals, on s'executen cada sistema multi-agent(Fig. 25)(Fig. 26):

```
agent2@127.0.0.1: starting
rest2@127.0.0.1: starting
* Running on http://192.168.1.19:5000/ (Press CTRL+C to quit)
192.168.1.19 - - [13/Jul/2017 11:27:20] "GET / HTTP/1.1" 200 -
```

Fig. 25. Eixida per terminal del sistema receptor

```
rest@127.0.0.1: starting
agent@127.0.0.1: starting
HELLO WORLD
```

Fig. 26. Eixida per terminal del sistema emissor

Amb la simplicitat en la que està implementada l'API web s'aconsegueix complir amb les diferents restriccions que cal tindre en compte, per a que el servei implementat siga del tipus REST. Partint de l'explicació realitzada anteriorment, de com s'implementa l'agent REST dins d'un sistema multi-agent de SPADE, s'explicarà en el següent apartat la seua aplicació en diferents sistemes distribuïts més complexes.

4 Avaluació

4.1 Sistema distribuït sintètic

4.1.1 Descripció del problema

El primer exemple de sistema distribuït on s'aplica el modul REST implementat consisteix en un conjunt de sistemes multi-agent, que permeten obtindre la temperatura mitjana d'un país a partir de les dades obtingudes de les temperatures en diferents zones. Els diversos sistemes es distribueixen en diferents capes jeràrquiques(Fig. 27), on cadascuna conté un tipus dels següents sistemes:

Sistema de ciutat. Representa a una ciutat i es compon per un agent que representa la ciutat i diversos agents que s'encarreguen d'obtindre la temperatura de diverses zones, tant per dins o pels voltants dels límits de la ciutat. L'objectiu del sistema és obtindre les temperatures de cada zona i enviar aquests resultats al representant de la ciutat, per a que ho envie als sistemes que estan per sobre d'ell. Aquest tipus de sistema es situaria en la capa inferior de la distribució jeràrquica.

Sistema d'autonomia. Aquest sistema es situa damunt dels sistemes que representen les ciutats, de manera que permet obtindre les temperatures de les ciutats que formen l'autonomia. Es compon per un agent representant de l'autonomia i un conjunt d'agents que representen a les diferents ciutats que formen l'autonomia. El sistema utilitza els representants de cada ciutat per a que es comuniquen amb el sistema de la seua ciutat i obtindre la mitjana de temperatures, de manera que aquesta informació es dona al representant de l'autonomia, que s'encarrega d'enviar-ho al sistema de la capa superior. El tipus de sistema d'autonomia es situa en la distribució jeràrquica per damunt dels sistemes de ciutat i sota el sistema de país.

Sistema de país. Consisteix en el sistema que es situa per damunt de qualsevol altre sistema i que només pot existir-ne un dins del sistema distribuït. Esta format per l'agent representant del país i un conjunt d'agents que representen a les diferents autonomies. El funcionament d'aquest sistema és paregut al dels sistemes d'autonomia, amb la diferència de que actua en un nivell superior i que permet rebre ordres per part d'un usuari mitjançat una interfície. Partint de les ordres que se li dona a aquest sistema mitjançat la seua interfície, el representant del país informa de la petició a realitzar a cadascun dels agents representants d'autonomies, per a obtindre la mitjana de temperatures de cada part del país i obtindre la mitjana total.

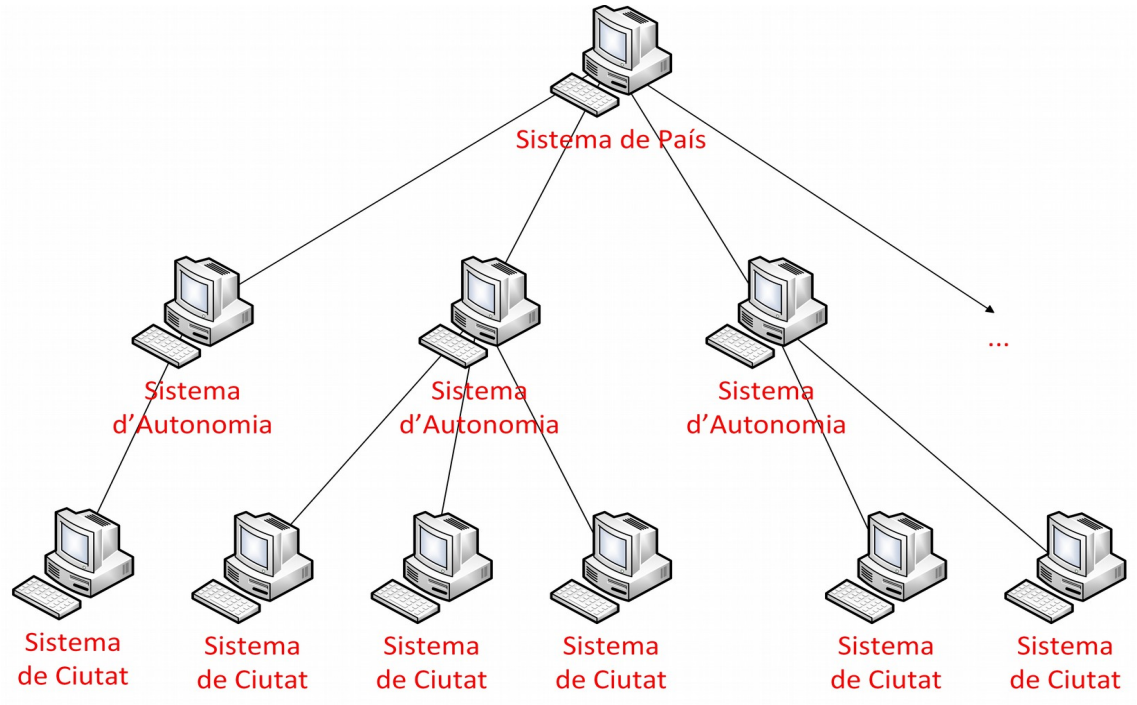

Fig. 27. Representació jeràrquica del sistema distribuït

4.1.2 Implementació

Partint de l'explicació del sistema distribuït plantejat, en cadascun d'aquests tipus de sistemes mencionats s'utilitza un agent REST per a rebre les sol·licituds dels sistemes que es situen damunt d'ells i les realitzar a aquells que es situen en la capa inferior. Com que el sistema distribuït plantejat només consisteix en obtindre la mitjana de temperatures en un país, les peticions a realitzar entre les diferents capes del sistema són les d'obtindre la temperatura mitja del sistema inferior i obtindre informació dels serveis del sistema i peticions que se li poden fer mitjançat Restful. Aquestes dos últimes s'implementen en qualsevol sistema multi-agent per a facilitar informació a serveis externs sobre el sistema amb el que es volen comunicar.

Dins de la sol·licitud de la mitjana de temperatures, es divideix en tres peticions amb resultats diferents, que serien l'obtenció de la temperatura en graus Celsius, Kelvin o Fahrenheit, sent el primer tipus el que es retorna per defecte. En la següent taula(Fig. 28) es mostren les diferents sol·licituds que es poden realitzar en els sistemes de tipus autonomia i país, junt al seu tipus i la seua descripció:

Petició	Tipus	Descripció
/	GET	Demana els serveis del sistema extern.
/	OPTIONS	Demana la llista de les sol·licituds que admet el sistema extern.
/average	GET	Demana la mitjana de temperatures en graus Celsius.
/average/kelvin	GET	Demana la mitjana de temperatures en graus Kelvin.
/average/fahrenheit	GET	Demana la mitjana de temperatures en graus Fahrenheit.

Fig. 28. Llista de possibles peticions a poder realitzar

Les sol·licituds que accepten cadascun del agents REST dels sistemes d'autonomia i ciutat són les mateixes que s'han explicat abans, que poden retornar la mitjana de temperatures en diferents unitat, la informació respecte als serveis de la plataforma i la llista de URIs que admeten.

Els diferents sistemes d'aquesta distribució estan implementats de manera que es poden declarar diverses propietats abans de que s'executen, com són el nom al que fa referència al sistema creat, la seua adreça d'ubicació en la xarxa de sistemes distribuïts, el conjunt d'agents locals o representants dels sistemes inferiors que el formen i les seues adreces per a realitzar les comunicacions externes. Les implementacions de les diferents parts d'aquest sistema distribuït estan disponibles en un repositori propi de GitHub[4].

En el sistema de ciutat, l'agent REST està modificat per a que puga rebre les peticions mencionades anteriorment, de manera que si es tracten d'obtenció d'informació dels serveis o de les sol·licituds permeses del sistema, el mateix agent recopila aquesta informació i la retorna. En el cas d'obtindre la mitjana de temperatures de la ciutat, crea una petició amb els seus arguments, si en té, per a que la llegisca el mateix agent i envie l'ordre a realitzar al representant de la ciutat. Una vegada que obtinga la mitjana de temperatures, la retorna mitjançat el servei Restful.

L'agent que representa la ciutat s'implementa per a que demane als agents locals de la ciutat la temperatura que obtenen de la seua zona en el tipus que correspon, quan es reba el missatge de sol·licitud de la temperatura per part de l'agent REST. Aquesta obtenció d'informació es realitza mitjançat un comportament del tipus *Time-Out*, que

4 GitHub. *SPADE-with-REST/temp_server/*. https://github.com/joasanm/SPADE-with-REST/tree/master/temp_server

espera un temps a que els agents locals puguen obtindre la temperatura i enviar-la a aquest comportament. Una vegada haja finalitzat el temps d'espera del comportament *Time-Out*, l'agent representant de la ciutat no accepta més temperatures, de manera que obté la mitjana de les que ha pogut rebre i ho envia a l'agent REST.

Respecte a la implementació de l'agent local, només s'encarrega d'obtindre la temperatura de la zona que li ha tocat i la converteix, si es necessari, a graus Kelvin o Fahrenheit seguint les formules de conversió oficials[5], per a després enviar-la a l'agent representant de la ciutat. L'organització del sistema de ciutat és el que es mostra en la següent imatge(Fig. 29):

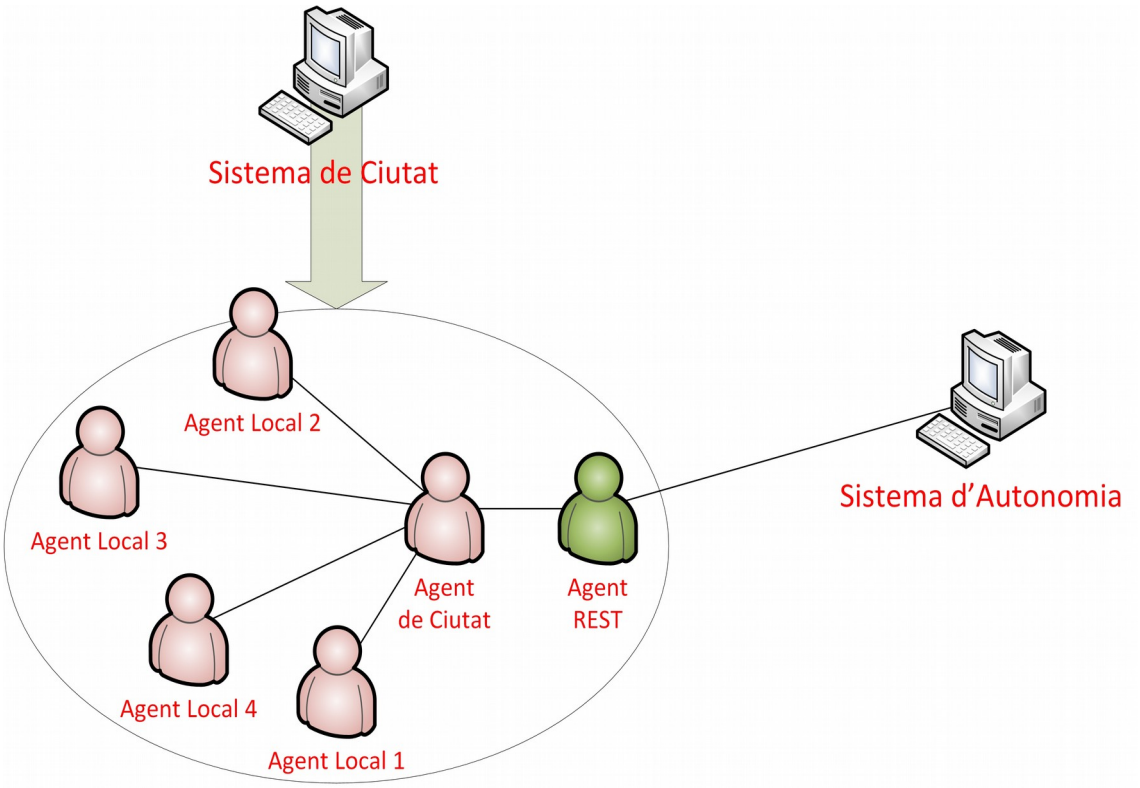

Fig. 29. Organització del sistema de ciutat

El sistema d'autonomia s'executa també partint de les peticions que rep l'agent REST, de manera que respon directament si es tracta de l'obtenció dels serveis o sol·licituds acceptables pel sistema. En el cas d'altres possibles peticions, l'agent REST realitza les accions necessàries per a comunicar l'ordre a realitzar al representant de l'autonomia.

El representant del sistema d'autonomia, quan rep la petició de l'agent REST, demana als diferents representants de les ciutats l'obtenció de la mitjana de temperatures en el tipus de graus que s'hagen sol·licitat per part de l'agent REST, de manera que es segueix el mateix mètode de recepció d'informació dels diferents agents com en el del

5 Measure Converter. *Temperature conversion table and converter*. http://www.allmeasures.com/temperature.html

sistema de ciutat, quan es necessita rebre la temperatura dels diferents agents locals. Una vegada obtinguda les diferents temperatures dels missatges amb dades vàlides, l'agent representant de l'autonomia realitza la mitjana de les temperatures i ho envia a l'agent REST per a retornar-ho a qui ho ha demanat.

L'agent representant de la ciutat dins del sistema d'autonomia s'encarrega de comunicar-se amb l'agent REST per a realitzar una sol·licitud al seu sistema de ciutat, amb l'objectiu d'obtindre la mitjana de temperatures de la ciutat i en la conversió que li hagen demanat. El resultat retornat per el sistema extern s'envia a l'agent representant de l'autonomia. Cal tindre en compte que el comportament de l'agent REST per a realitzar peticions es crea com una instància de comportament diferent per a cada representant de ciutat, tal com s'havia mencionat en apartats anteriors, per a paral·lelitzar tasques a realitzar dins del sistema. L'organització d'aquest sistema és el que es mostra a continuació(Fig. 30):

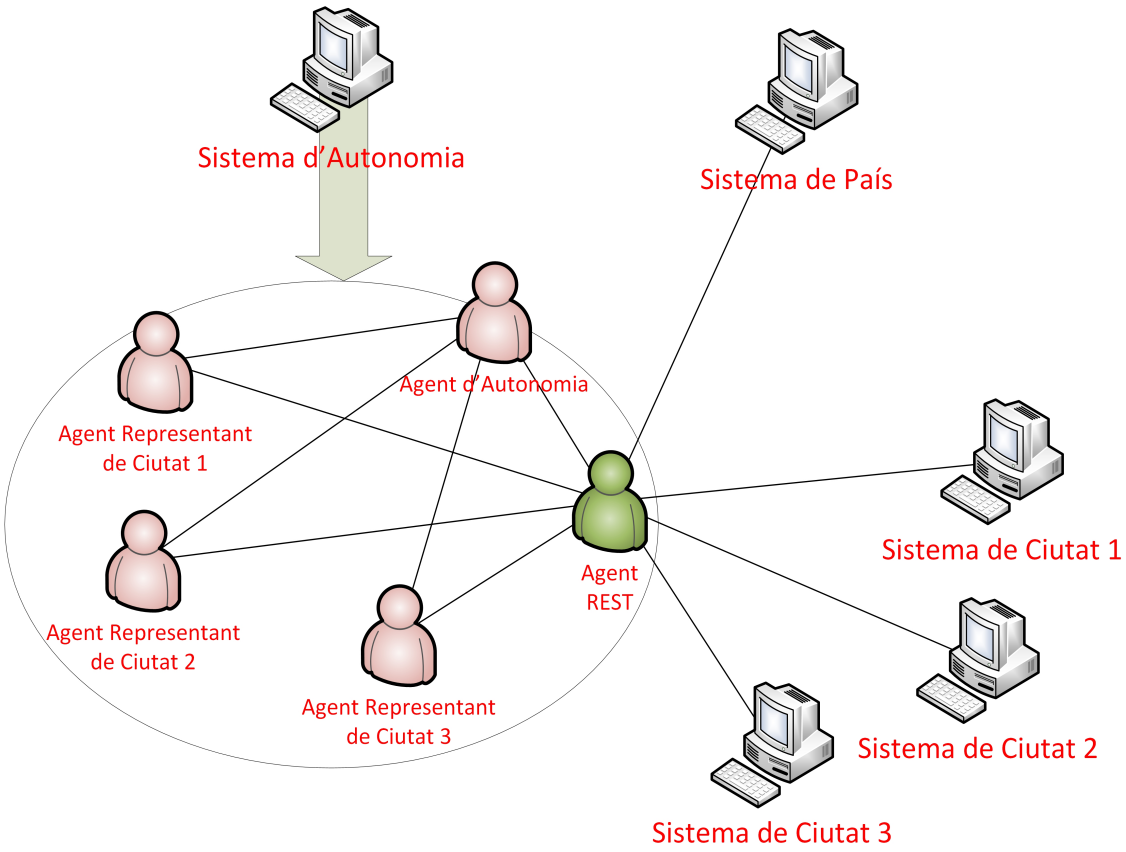

Fig. 30. Representació del sistema d'autonomia

Finalment, el sistema de país s'implementa per a que comence amb l'agent representant de país rebent per part de l'usuari la petició que es vol realitzar. Si la sol·licitud que demana l'usuari no existeix en el sistema distribuït o té errors de sintaxi, el representant del país li respon informant que no existeix la petició que vol realitzar. En el cas contrari, l'agent s'encarrega d'enviar als representants de les diferents autonomies la sol·licitud que han de realitzar, de manera que envien el seu resultat a

un comportament del representant del sistema del tipus *Time-Out*, de la mateixa forma que s'ha implementat en els altres sistemes. Quan haja passat el temps límit, l'agent representant de país realitza la mitjana i la mostra a l'usuari.

Cadascun dels representants d'autonomia que formen el sistema de país s'encarreguen, quan li ho sol·liciten, de crear una instància del comportament de realitzar peticions de l'agent REST, per a comunicar-se amb els seus respectius sistemes d'autonomia i obtindre les temperatures demanades. Una vegada realitzada la petició, l'agent REST envia el resultat al representant d'autonomia que ho haja demanat, per a retornar-lo a l'agent representant del país.

En el sistema de país, l'agent REST està modificat només per a realitzar peticions, que vullguen realitzar els diferents representants d'autonomia, tal com s'ha explicat anteriorment. A continuació es mostra l'organització del sistema de país(Fig. 31).

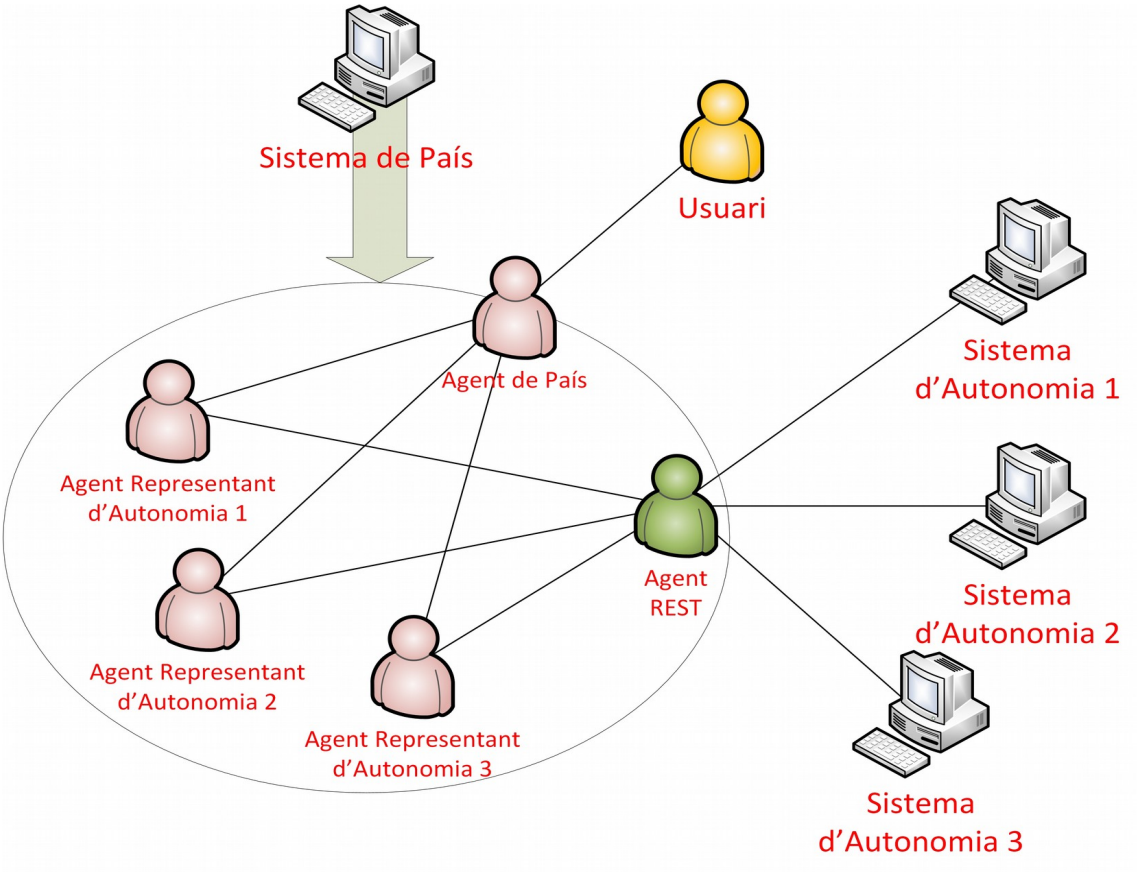

Fig. 31. Organització del sistema de país

Les APIs de REST implementades en cada agent REST dels sistemes permeten separar el servidor del client mitjançat una interfície uniforme, de manera que no existeixen dependències entre dos sistemes i es poden modificar de manera independent. La part que rep les sol·licituds no necessita emmagatzemar informació de la comunicació, i alguns sistemes realitzen peticions a altres serveis com si es tractessin d'intermediaris. Partint de les característiques mencionades anteriorment,

els agents REST implementats utilitzen serveis que es poden denominar amb arquitectura Restful.

Dins d'aquest sistema distribuït en capes, cal tindre en compte que els temps d'espera que hi han dins de cada sistema multi-agent, tant dels comportaments del tipus *Time-Out*, de les peticions que es realitzen a sistemes externs i de les sol·licituds que s'intenten resoldre dins d'un sistema, han de ser superior als dels sistemes situats en la capa inferior. També cal tindre en compte quant han de durar els diferents temps d'espera dins d'un sistema multi-agent. Les següents normes s'han de complir per a que funcione de manera correcta el sistema distribuït implementat:

- El temps d'espera de la resolució i resposta d'una petició dins d'un sistema ha de ser superior al del temps necessari que necessita el comportament de l'agent representant del mateix sistema, que rep la informació dels diferents agents representants de la capa inferior.

- Al mateix temps, el temps d'espera del comportament, mencionat en la norma anterior, ha de ser superior al temps necessari per a que l'agent REST mantinga la connexió externa amb altres sistemes d'un ordre jeràrquic inferior.

- El temps d'espera que manté la comunicació entre l'agent REST amb un sistema de la capa inferior ha de ser superior al temps que requereix el sistema de la capa inferior per a intentar resoldre i reenviar la resposta que li sol·liciten.

4.1.3 Validació

Per a validar el funcionament del modul REST en aquest sistema distribuït, s'ha utilitzat una mètrica que relaciona la quantitat d'agents locals que aconsegueixen realitzar l'enviament de la temperatura obtinguda a l'agent representant de la ciutat, sense importar el fet de que requerisca o no convertir-la a una altra unitat de temperatura, amb el número d'agents locals que formen en total una, dos o tres ciutats diferents. Amb aquesta mètrica s'intenta demostrar que la distribució d'agents en diverses plataformes permet una millora en l'eficiència i escalabilitat referent a la realització de comunicacions dins d'un sistema de la plataforma SPADE.

Per a aplicar aquesta mètrica, els sistemes distribuïts utilitzats es componen d'un sistema de país, un sistema d'autonomia i diversos sistemes de ciutat. Cadascun dels sistemes utilitzats s'executen en màquines físiques diferents, excepte en el cas dels sistemes de ciutat que s'executen en diferents màquines virtuals mitjançat VirtualBox[6].

6 Oracle. *Oracle VM VirtualBox*. https://www.virtualbox.org/

L'ordinador que s'encarrega d'executar el sistema de país conté les següents característiques principals, que afecten més en el rendiment de l'execució:

Microprocessador: Intel Core Duo Quad Q6600. 4 nuclis físics amb 2.4 GHz de velocitat de còmput.

Memòria RAM: 4 GB de capacitat de tipus DDR2 amb 800 MHz de velocitat de còmput.

Sistema operatiu: Ubuntu Gnome Desktop 16.04 de 64 bits.

La màquina física que executa el sistema d'autonomia utilitza els components que es mostren a continuació:

Microprocessador: Intel Core i7-2670M. 4 nuclis físics, que poden simular 8 nuclis, amb 2.2 GHz de velocitat de còmput, que pot augmentar fins a 3.1 GHz.

Memòria RAM: 6 GB de capacitat de tipus DDR3 amb 1333 MHz de velocitat de còmput.

Sistema operatiu: Ubuntu Desktop 16.04 de 64 bits.

Respecte a les màquines virtuals, les tres que es volen utilitzar s'utilitzen en l'ultim ordinador mencionat, per a poder executar els diferents sistemes de ciutat, amb les següents propietats:

Microprocessador: Intel Core i7-2670M. 1 nucli amb 2.2 GHz de velocitat de còmput, que pot augmentar fins a 3.1 GHz.

Memòria RAM: 1024 MB de capacitat de tipus DDR3 amb 1333 MHz de velocitat de còmput.

Sistema operatiu: Ubuntu Gnome Desktop 16.04 de 64 bits.

La mètrica s'ha realitzat executant una vegada cada part del sistema distribuït per a cada quantitat diferent d'agents locals, excepte la part del sistema referent al país, que s'executa 10 vegades per a realitzar diverses vegades la petició d'obtindre la mitjana de temperatures de l'únic sistema d'autonomia, que al mateix temps ho sol·licita als sistemes de ciutat. Partint de les 10 proves realitzades amb diferents quantitats d'agents locals distribuïts en els diversos sistemes de ciutat, s'obté la mitjana de respostes rebudes correctament per part de cada ciutat. En les següents taules es mostren els resultats obtinguts amb 10, 20, 30, 40 i 50 agents locals distribuïts en una, dos i tres ciutats. El fet de fer proves fins a 50 agents es degut a que un sòl sistema

comença a tindre problemes per a poder inicialitzar i registrar en la plataforma tota aquesta quantitat d'agents.

Número de prova	Respostes d'agents rebuts							
	1 Ciutat	2 Ciutats			3 Ciutats			
		Ciutat1	Ciutat2	Total	Ciutat1	Ciutat2	Ciutat3	Total
1	10	5	5	10	3	3	4	10
2	10	5	5	10	3	3	4	10
3	10	5	5	10	3	3	4	10
4	10	5	5	10	3	3	4	10
5	10	5	5	10	3	3	4	10
6	10	5	5	10	3	3	4	10
7	10	5	5	10	3	3	4	10
8	10	5	5	10	3	3	4	10
9	10	5	5	10	3	3	4	10
10	10	5	5	10	3	3	4	10
Mitjana	**10**	5	5	**10**	3	3	4	**10**

Fig. 32. Resultats obtinguts amb 10 agents locals

Partint dels resultats d'aquest cas(Fig. 32), tots els agents locals en els diferents casos realitzen la resposta al representant de la seua ciutat de manera correcta. Cal tindre en compte que en el cas d'utilitzar tres ciutats no s'ha pogut repartir equitativament els 10 agents locals, per raons obvies respecte a la divisibilitat dels agents.

Número de prova	Respostes d'agents rebuts							
	1 Ciutat	2 Ciutats			3 Ciutats			
		Ciutat1	Ciutat2	Total	Ciutat1	Ciutat2	Ciutat3	Total
1	20	10	10	20	7	7	6	20
2	19	10	10	20	7	7	6	20
3	13	10	10	20	7	7	6	20
4	19	10	10	20	7	7	6	20
5	19	10	10	20	7	7	6	20
6	20	10	10	20	7	7	6	20
7	20	10	10	20	7	7	6	20
8	20	10	10	20	7	7	6	20
9	13	10	10	20	7	7	6	20
10	19	10	10	20	7	7	6	20
Mitjana	**18,2**	10	10	**20**	7	7	6	**20**

Fig. 33. Resultats obtinguts amb 20 agents locals

A partir de 20 agents locals(Fig. 33), la distribució en una única ciutat comença a perdre missatges de les diferents temperatures obtingudes, mentres que amb les distribucions de dos i tres ciutats encara es reben correctament tots els missatges. En aquest cas també sorgeix el mateix problema d'equitativitat en la distribució dels agents locals entre tres ciutats.

Número de prova	Respostes d'agents rebuts							
	1 Ciutat	2 Ciutats			3 Ciutats			
		Ciutat1	Ciutat2	Total	Ciutat1	Ciutat2	Ciutat3	Total
1	13	15	15	30	10	10	10	30
2	28	15	14	29	10	10	10	30
3	29	15	15	30	10	10	10	30
4	29	15	15	30	10	10	10	30
5	15	15	15	30	10	10	10	30
6	29	14	15	29	10	10	10	30
7	30	15	15	30	10	10	10	30
8	16	14	15	29	10	10	10	30
9	30	15	15	30	10	10	10	30
10	29	11	14	25	10	10	10	30
Mitjana	**24,8**	14,4	14,8	**29,2**	10	10	10	**30**

Fig. 34. Resultats obtinguts amb 30 agents locals

Tal com es mostra en la taula(Fig. 34), amb una ciutat es comença a perdre almenys 5 de cada 30 missatges, mentres que amb dos ciutats es tendeix a perdre casi un missatge que s'ha enviat incorrectament o que l'agent local no ha pogut realitzar. Per al cas de la utilització distribuïda de tres ciutats encara es reben correctament tots els missatges.

Número de prova	Respostes d'agents rebuts							
	1 Ciutat	2 Ciutats			3 Ciutats			
		Ciutat1	Ciutat2	Total	Ciutat1	Ciutat2	Ciutat3	Total
1	40	20	20	40	13	13	14	40
2	39	19	20	39	13	13	14	40
3	40	20	11	31	13	13	14	40
4	39	16	20	36	13	13	13	39
5	39	16	20	36	13	13	13	39
6	39	20	19	39	13	13	14	40
7	34	19	19	38	13	13	14	40
8	38	20	17	37	12	13	14	39
9	25	14	20	34	12	13	13	38
10	22	19	20	39	12	12	14	38
Mitjana	**35,5**	18,3	18,6	**36,9**	12,7	12,9	13,7	**39,3**

Fig. 35. Resultats obtinguts amb 40 agents locals

En aquest cas(Fig. 35) qualsevol de les distribucions ja no aconsegueix que els seus agents locals aconseguisquen realitzar les seues úniques tasques correctament, on la distribució amb una ciutat aconsegueix rebre correctament fins a 35.5 missatges de 40, mentres que amb dos sistemes de ciutat s'arriba a rebre fins a 36.9 i amb tres ciutats es manté per damunt dels altres amb 39.3 missatges de mitjana rebuts correctament.

Número de prova	Respostes d'agents rebuts							
	1 Ciutat	2 Ciutats			3 Ciutats			
		Ciutat1	Ciutat2	Total	Ciutat1	Ciutat2	Ciutat3	Total
1	39	25	25	50	17	17	16	50
2	37	24	24	48	14	17	13	44
3	36	25	24	49	17	17	16	50
4	49	25	22	47	17	16	15	48
5	48	20	22	42	16	17	15	48
6	42	24	24	48	16	16	15	47
7	33	25	25	50	17	17	16	50
8	49	24	24	48	16	17	16	49
9	49	24	25	49	17	17	16	50
10	50	25	25	50	16	16	16	48
Mitjana	**43,2**	24,1	24	**48,1**	16,3	16,7	15,4	**48,4**

Fig. 36. Resultats obtinguts amb 50 agents locals

Finalment, amb 50 agents locals(Fig. 36) els resultats ja no són prou estables, sobre tot per a la distribució d'una ciutat, on es mostra que es perden casi fins a 7 missatges d'agents locals. En les distribucions de dos i tres ciutats es tendeix a perdre fins a 2 missatges, encara que només es diferencien per tres dècimes.

Partint d'aquests resultats, es mostra a continuació una taula que resumeix els resultats anteriors(Fig. 37), seguit d'una gràfica per poder visualitzar les diferencies obtingudes amb les diferents distribucions.

Agents locals	Respostes rebudes		
	1 ciutat	2 ciutats	3 ciutats
10	10	10	10
20	18,2	20	20
30	24,8	29,2	30
40	35,5	36,9	39,3
50	43,2	48,1	48,4

Fig. 37. Resum dels resultats obtinguts

Per antelació es pot comprovar que un sistema multi-agent en SPADE deixa de ser estable quan es compon per més de 10 agents. Això pot explicar l'obtenció correcta de tots els missatges fins al cas de 10 agents distribuïts en una ciutat, 20 agents distribuïts en dos ciutats i 30 organitzats en tres sistemes diferents del mateix tipus.

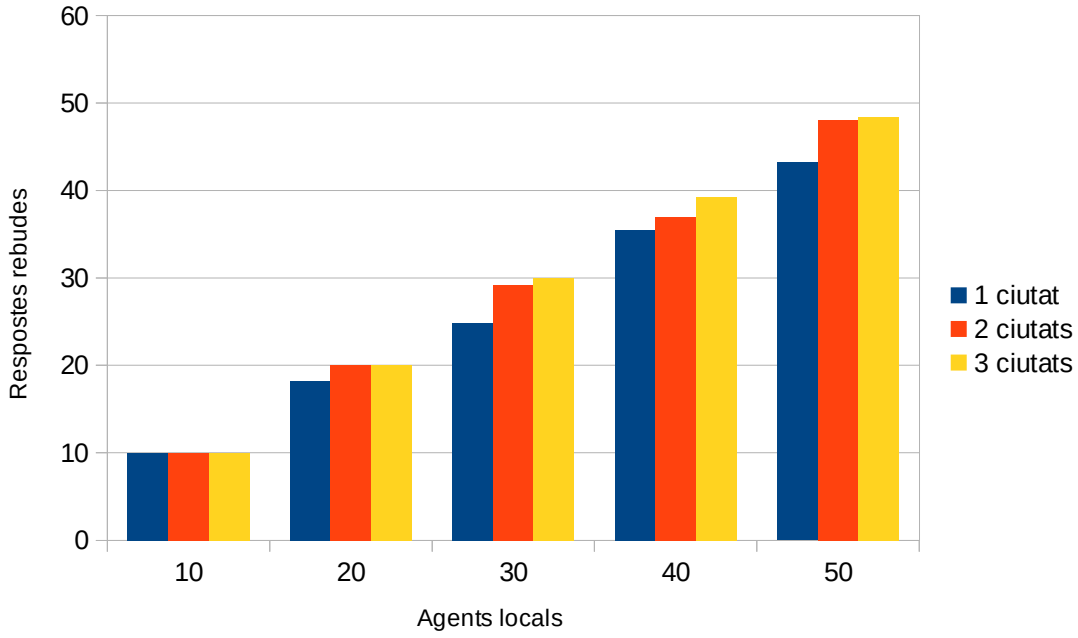

Fig. 38. Representació dels resultats obtinguts

Partint de l'anterior gràfica de barres(Fig. 38) i les conclusions tretes en les taules de resultats, es pot comprovar que la utilització d'un sistema distribuït per a reduir el nombre d'agents que compon un sistema multi-agent en SPADE permet una major estabilitat en cadascun dels sistemes, tant en les comunicacions entre agents del sistema com en el funcionament correcte de cada agent. També es pot vore en la

gràfica que entre les distribucions de dos i tres sistemes de ciutat existeixen casos on hi ha menys diferències que en altres, tal com es mostra en els casos de 30 i 50 agents, on només es diferencien com a molt per 8 dècimes mentres que en el cas de 40 agents hi ha un marge de 2.4 missatges rebuts. Això es pot explicar degut a l'obtenció de casos aïllats, on el sistema pot haver inutilitzat alguns agents locals o que hagen fallat les comunicacions amb l'agent representant de la ciutat. Si es realitzés les mateixes proves amb més sistemes de ciutats, de manera que la distribució dels agents locals en cada ciutat seria més baixa, es possible que s'obtinguen millors resultats degut a que cada sistema multi-agent només s'ha d'encarregar d'executar un grup menor d'agents.

Respecte a altres proves realitzades, també s'ha intentat realitzar partint d'una altra mètrica, que relaciona el temps que es tarda a rebre correctament la mitjana de temperatures des del sistema de país amb el número d'agents locals que es distribueixen en els possibles sistemes de ciutat. Però les proves no s'han arribat a realitzar del tot degut a que el temps que tarda a respondre el sistema de país depén directament dels temps d'espera que s'han definit per a rebre les possibles respostes de diferents representants d'autonomia i per a esperar respostes de les sol·licituds realitzades als sistemes d'autonomia.

Encara així, la utilització del mòdul REST per a la comunicació entre sistemes distribuïts permet un millora de rendiment i escalabilitat gracies a la distribució dels agents en diversos sistemes multi-agent en el cas de SPADE i també permet avançar en la possibilitat de realitzar canals de comunicació amb sistemes multi-agent diferents i altres serveis web o dispositius amb connexió sense fil.

4.2 Sistema multi-agent per a la gestió de trasplantaments d'òrgans

4.2.1 Descripció del problema

El sistema distribuït que es proposa en aquest apartat consisteix en crear una xarxa de sistemes multi-agent que coordinen entre ells per a decidir quin pacient ha de rebre el trasplant d'un òrgan disponible des d'un hospital d'Espanya. Aquest tipus de sistema proposa la substitució del sistema que s'utilitza actualment entre els diferents hospitals del país per un altre dissenyat amb sistemes multi-agent, pensat pel Grup de Recerca en Intel·ligència Artificial del Departament d'Enginyeria Informàtica i Matemàtiques de la Universitat Rovira i Virgili i el Departament d'Electrònica, Informàtica i Automàtica de la Universitat de Girona[22].

Segons aquesta proposta, la coordinació del trasplant d'òrgans humans es tracta d'una tasca difícil que implica aspectes legals, clínics, d'organització i humans amb la complexitat d'entorns distribuïts d'hospitals i d'institucions governamentals. En Espanya hi han dos raons amb els quals es justifica l'èxit del trasplant d'òrgans, que són les noves tècniques de cirurgia amb tractaments mèdics i la societat actual que permet informar d'un increment important del número d'òrgans que es poden trasplantar des d'un donant mort a una persona viva. Aquesta segona raó està relacionada amb l'existència d'una organització estructurada que coordina tots els passos a realitzar en el procés de la donació i el trasplantament d'acord a les normes i lleis locals, regionals, nacionals i internacionals. La estructura mencionada i la idea de tindre un coordinador de trasplant en cada hospital amb una unitat mèdica de trasplantament defineix l'actual Model Espanyol, el qual és un dels més eficients del món.

L'organització i el trasplantament són tasques complexes que requereixen diverses activitats, que involucren gent i grups de treball, i un procés administratiu. Les activitats principals del procés de trasplantament són la detecció d'un donant d'òrgan, examinació de l'òrgan a donar, confirmació legal de la mort del donant, manteniment de l'òrgan, confirmació del consentiment i l'autorització legal de la família, organització de l'extracció i el trasplant de l'òrgan, i l'examinació clínica de l'evolució en l'estat del receptor. La gent que es té en compte en aquestes activitats són el pacient receptor de l'òrgan, els metges, el donant, la família tant del donant com del receptor, el coordinador del trasplant, el personal clínic, els equips de cirurgia d'extracció i d'implantació, els assessors legals, les autoritats, i l'equip logístic entre molts altres. Aquesta gent ha de portar les tasques administratives al corrent amb les tasques clíniques i ha de seguir les normes i la legislació establerta per les organitzacions i autoritats sanitàries.

La ONT o Organització Nacional de Trasplantaments ha identificat diverses raons que confirmen els casos de pèrdua de donants d'òrgans, demostrats en uns estudis que informen de la gent que mor en Europa mentres espera el trasplant d'un òrgan, com poden ser de cor, fetge o pulmó, fluctua entre el 15% i el 30% de la població. Aquestes raons són la no detecció de donant, les contradiccions mèdiques, les contradiccions familiars, les contradiccions judicials, la falta de manteniment d'òrgans, i els errors comesos durant les operacions quirúrgiques. Partint d'aquestes raons, es necessari el desenvolupament de mecanismes que redueixin el percentatge de pèrdues en cadascuna de les etapes del procés del trasplant. Una manera de millorar aquest procés és elaborant un sistema amb suport de decisions en la coordinació dels trasplantaments. Aquest sistema ha de tindre funcions de comunicació i informació que garanteixen una interacció distribuïda per a mantindre un sistema d'arxius històrics i

per a extraure nous coneixements a partir dels comportaments del sistema i de l'anàlisi d'informació.

El sistema descrit anteriorment estaria implementat amb sistemes multi-agent, que es dissenyaria per a suportar les activitats principals del procés de trasplant d'òrgans, sent compatible amb l'estructura d'organització actual d'Espanya, a part de que consistirà en un sistema distribuït obert que es puga adaptar fàcilment a canvis estructurals.

4.2.2 Implementació

L'estructura jeràrquica del sistema distribuït a desenvolupar conté els següents nivells, en l'ordre seguit des del que es situa més a dalt fins al que es troba a baix del tot:

Nivell nacional. Format per un únic sistema multi-agent, el coordinador nacional, que s'encarrega de les comunicacions entre els diferents sistemes del nivell inferior.

Nivell zonal. Aquest està format per un conjunt de 6 sistemes que representen cada zona del país, que es poden comunicar amb el sistema de la capa superior i amb les regions que el formen cadascun d'ells.

Nivell regional. És el nivell on es situen les 19 regions autònomes que es distribueixen entre les zones del nivell superior. Els sistemes multi-agent que representen les diverses regions es poden comunicar amb la zona en la que es troba i amb els diferents hospitals que contenen en el nivell inferior.

Nivell hospital. Consisteix en el grup d'hospitals que hi han al país i que estan agrupades per diferents ciutats, que al mateix temps es separen per regions. Els hospitals es poden comunicar amb la regió que el representa i els diferents hospitals que estiguen en la mateixa ciutat.

A continuació es mostra la representació de l'estructura jeràrquica del sistema distribuït que es vol implementar:

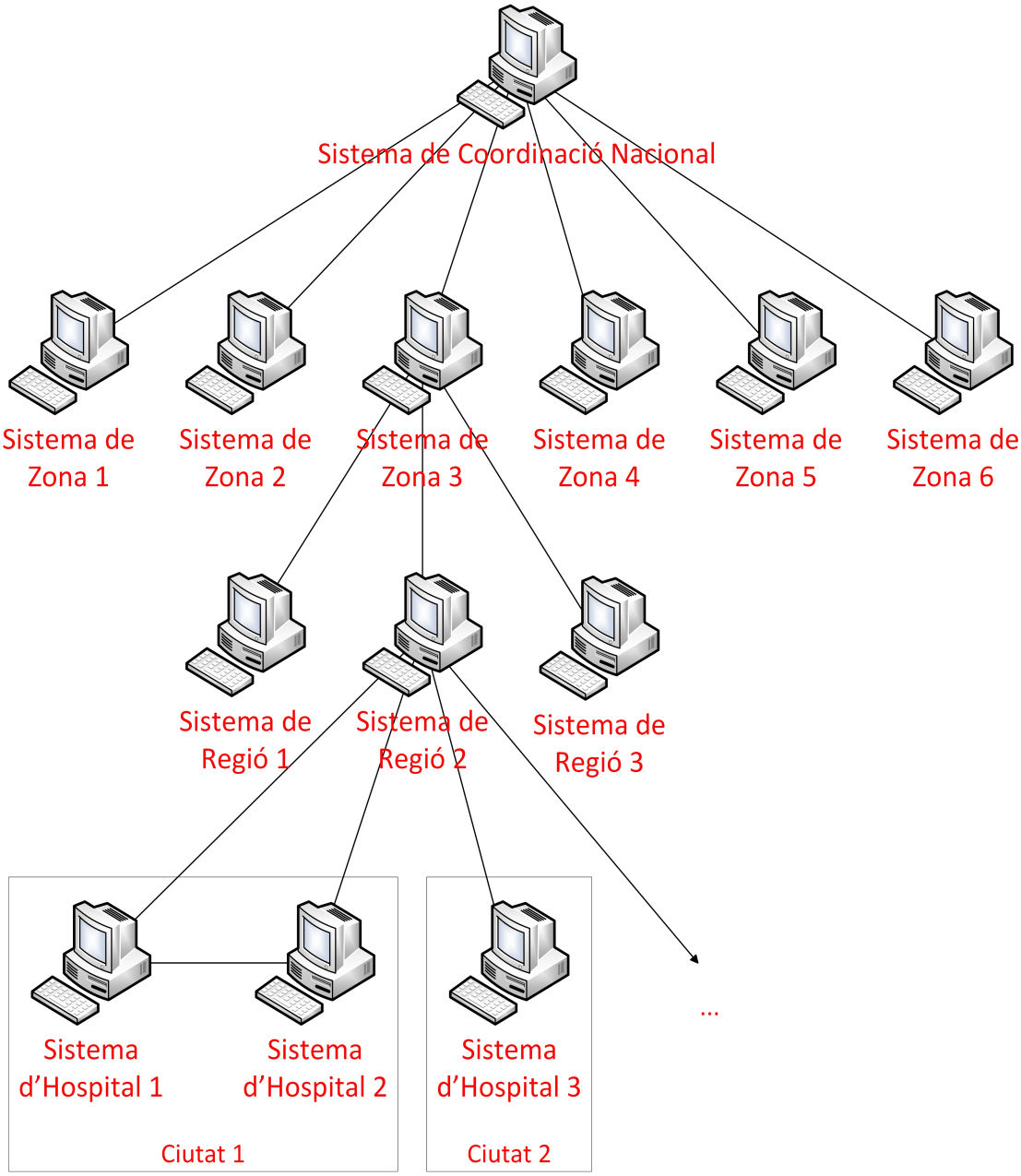

Fig. 39. Disseny del sistema distribuït de coordinació

Cadascun dels agents que componen cada sistema té diferents coneixements i rols, de manera que han de coordinar les seues activitats per a realitzar en el menor temps possible la gestió del trasplant d'òrgans. Existeixen en el sistema distribuït dos agents intel·ligents especials que han de ser únics:

Agent coordinador d'emergència. És l'agent coordinador dels casos d'emergència 0. Aquests casos el formen aquells pacients receptors que estan esperant l'òrgan i es troben en una condició molt crítica, on la seua vida està en risc si el trasplant no es realitza a temps.

Agent històric. És el que s'encarrega de rebre informació de tots els trasplantaments que s'hagen realitzat en Espanya. Amb la informació de cada trasplant, l'agent pot

elaborar arxius històrics, crear estadístiques i aplicar tècniques de mineria de dades amb l'objectiu d'obtindre coneixements útils.

Aquests agents, com que s'ha de poder comunicar amb ells fàcilment des de qualsevol hospital del país, es situaran en el sistema multi-agent de coordinació nacional, situat en el nivell nacional.

El sistema multi-agent que representa a cada hospital està dissenyat amb un conjunt de tres agents, que són:

Agent interfície. Aquest agent s'encarrega d'interactuar amb l'usuari, que en aquest cas es tractaria d'un metge, per a rebre informació dels pacients receptors i òrgans, que demana a altres agents del sistema per a emmagatzemar-los en una base de dades i informar de la existència d'un donant disponible. També permet informar al metge de les accions importants realitzades pel sistema, com poden ser l'actualització de la base de dades, l'estat de la cerca de pacients receptors compatibles amb l'òrgan disponible o dels errors que s'hagen produït en la comunicació amb altres hospitals.

Agent base de dades. S'encarrega d'actualitzar la base de dades, referent als pacients en espera de rebre un òrgan i la llista de donants de l'hospital. També es comunica amb els agents per a informar del contingut de la base de dades o retornar la llista de receptors que són compatibles amb l'òrgan que està disponible en el mateix hospital o en un altre.

Agent coordinador de trasplantaments. És l'agent que s'encarrega d'interactuar amb els agents del sistema i realitzar la selecció del pacient al que es vol fer el trasplant de l'òrgan que tenen disponible, aplicant el procés SMART o *Search for the Most Appropiate Receptor of the Transplant*. Permet avisar a l'agent interfície de l'estat en que es troba la cerca d'un òrgan disponible en el mateix hospital i de quan se l'avisa de retornar la llista de receptors compatibles amb l'òrgan d'un altre hospital. També es comunica amb l'agent encarregat de la base de dades per a obtindre la llista de pacients compatibles i comprovar si en l'hospital existeix algun donant. En la comunicació amb l'agent REST, intenta realitzar sol·licituds externes, per a cercar en diferents hospitals l'existència d'un receptor compatible amb l'òrgan que es té en l'hospital. També realitza la petició amb aquell hospital en concret al que correspon el pacient, al que s'ha decidit realitzar el trasplant. Altres sol·licituds que realitza són les d'actualitzar la llista de pacients en estat crític de l'agent coordinador d'emergència i la d'afegir informació a la base de dades que utilitza l'agent històric de les coordinacions realitzades amb èxit. Respecte a les accions a realitzar per part d'altres hospitals, l'agent coordinador ha de retornar la llista de pacients compatibles i confirmar a un hospital si es vol realitzar el trasplant a un dels seus pacients, en el cas de que un dels pacients ha sigut seleccionat per a rebre l'òrgan.

Respecte a altres detalls, cada hospital té una llista dels hospitals que es troben en la mateixa ciutat, amb la seua adreça per poder comunicar-se. També contenen l'adreça web del sistema de la regió en la que es troba per a interactuar amb el sistema coordinador de la regió.

En quant a les representacions del pacients i els òrgans, consisteixen en diccionaris, on en el cas dels pacients es defineixen les característiques de la seua identitat, que és única, el seu estat, l'òrgan que està esperant a rebre i l'hospital on es troba entre moltes altres. En el cas de l'òrgan, es defineix el seu tipus i l'estat en el que es troba.

L'agent REST d'un sistema d'hospital s'encarrega de rebre les peticions, que obté tant del sistema que representa la regió com d'altres hospitals de la ciutat, relacionades amb retornar la llista de pacients receptors compatibles amb l'òrgan disponible que li descriuen i la confirmació de si es vol realitzar el trasplant d'òrgan al pacient escollit en eixe hospital. En la següent taula(Fig. 40) es mostren les diferents URIs que accepta un hospital, amb el seu tipus, la informació requerida i una breu descripció de la seua utilitat. Cal avisar que s'ometen aquelles que fan referència a l'obtenció de serveis del sistema i la llista de possibles peticions que pot acceptar, ja que s'han descrit en l'apartat anterior i se suposa que ho tenen implementat per defecte qualsevol sistema.

URI	Tipus	Informació requerida	Descripció
/patients	GET	Característiques de l'òrgan	Retorna la llista de receptors compatibles amb l'òrgan descrit.
/confirmation	GET	Característiques del pacient	Retorna una resposta respecte a la realització del trasplant al pacient descrit.

Fig. 40. Llista de peticions que es poden realitzar a un hospital

En quant a les sol·licituds que es poden realitzar al sistema de la regió o als hospitals propers estan les de demanar una llista dels pacients compatibles amb l'òrgan que té l'hospital disponible tant a l'agent coordinador d'emergència com als diferents nivells del sistema distribuït, afegir informació a l'agent històric, actualitzar la llista de pacients de l'agent coordinador d'emergència i obtindre la confirmació de l'hospital on es troba el pacient receptor que s'ha escollit per a realitzar el trasplant. En la taula que ve a continuació(Fig. 41) s'expliquen les diferents peticions que es poden realitzar a sistemes externs, amb el seu receptor, tipus de sol·licitud, les dades addicionals a enviar i una descripció del que es demana. En aquest cas s'utilitzen URIs dinàmiques, on es pot especificar el nom de l'hospital que realitza la petició, per a que el sistema extern tinga informació sobre quin hospital li ho ha demanat. En aquest cas també s'omet les sol·licituds que es poden realitzar per defecte en qualsevol sistema.

Petició	Tipus	Sistema receptor	Dades addicional	Descripció
/<hospital>/0Emergency	GET	Regió	Característiques de l'òrgan	Demana la llista de pacients en estat crític compatibles amb l'òrgan.
/patients	GET	Hospital	Característiques de l'òrgan	Demana la llista de pacients compatibles amb l'òrgan a un hospital de la mateixa ciutat.
/<hospital>/region	GET	Regió	Característiques de l'òrgan	Demana la llista de pacients compatibles amb l'òrgan d'hospitals de la mateixa regió.
/<hospital>/zone	GET	Regió	Característiques de l'òrgan	Demana la llista de pacients compatibles amb l'òrgan d'hospitals de diferents regions.
/<hospital>/country	GET	Regió	Característiques de l'òrgan	Demana la llista de pacients compatibles amb l'òrgan d'hospitals de diferents zones.
/<hospital>/hAgent	PUT	Regió	Característiques de l'òrgan i del pacient	Demana afegir dades noves a l'agent històric.

/<hospital>/ecAgent	POST	Regió	Característiques del pacient	Demana afegir un pacient critic a la llista de pacients de l'agent coordinador d'emergència.
/confirmation	GET	Hospital	Característiques del pacient	Demana la confirmació a un hospital de la mateixa ciutat per a realitzar el trasplant al seu pacient.
/<hospital>/region/confirmation	GET	Regió	Característiques del pacient	Demana la confirmació a un hospital de fora la ciutat per a realitzar el trasplant al seu pacient.
/0Emergency/confirmation	GET	Regió	Característiques del pacient	Demana la confirmació a l'hospital, on es troba el pacient obtingut per l'agent coordinador d'emergència.

Fig. 41. Sol·licituds que es poden realitzar des d'un hospital

Per a que es comuniquen els agents d'aquest tipus de sistema, s'han implementat diversos mètodes de comportament per a cadascun del agents, que s'activen quan reben un missatge d'un agent amb un tipus de contingut determinat.

La representació d'un sistema multi-agent d'hospital es pot mostrar en la següent imatge(Fig. 42), on es representen els diversos agents del sistema i els enllaços de comunicació que es poden realitzar entre ells mateixos i amb sistemes externs:

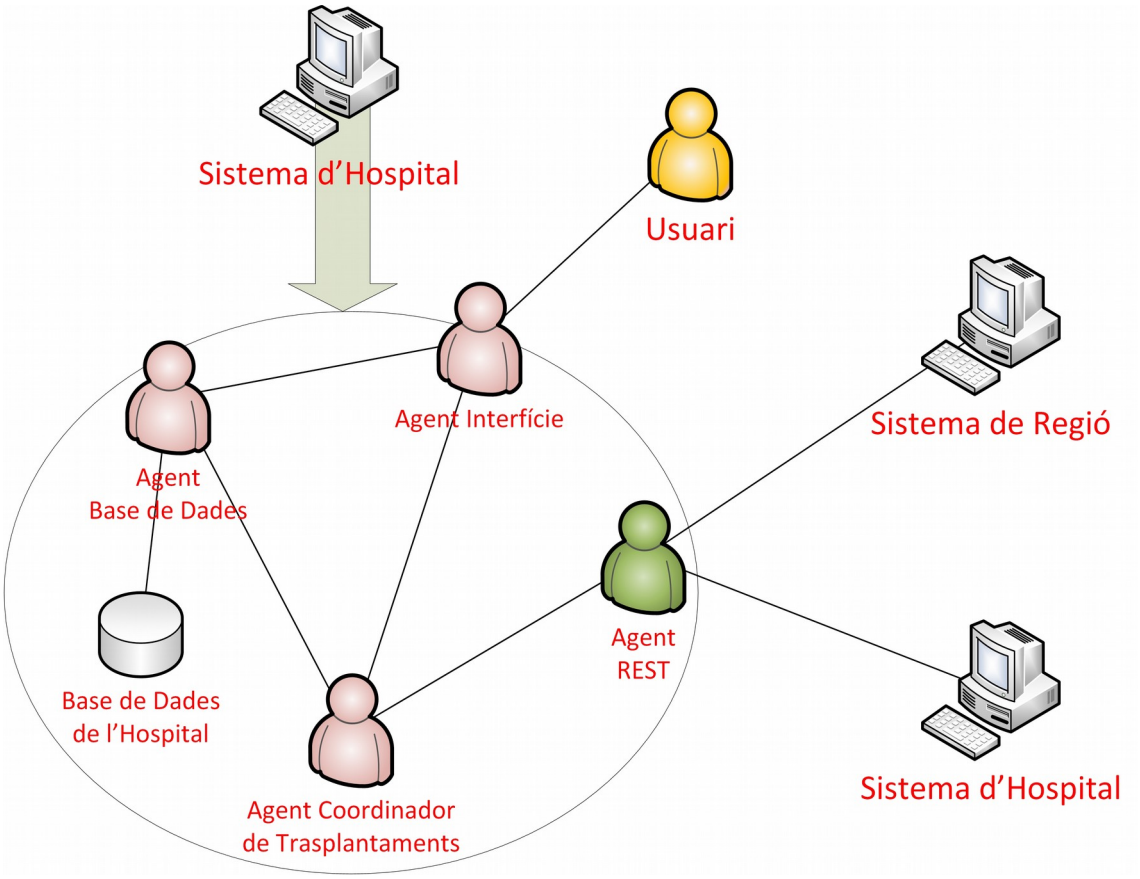

Fig. 42. Representació del sistema multi-agent d'hospital

El sistema que representa les distintes regions del país es compon per dos tipus d'agents, que són els següents:

Agent coordinador de la regió. És l'agent que s'encarrega de la coordinació dels diferents hospitals que representa i interactua amb la resta d'agents del sistema per a poder comunicar-se amb els hospitals i amb el sistema que representa la zona en la que es troba. En el sistema de la regió només pot haver-ne un d'aquest tipus. Es comunica amb l'agent REST per a realitzar peticions al sistema de la zona en la que es troba i per rebre sol·licituds del mateix.

Agent representant d'hospital. Aquest tipus d'agent s'encarrega de les comunicacions entre el sistema d'hospital, al que representa en la regió, i l'agent coordinador de la regió. Les comunicacions que realitza amb l'agent REST consisteixen en realitzar sol·licituds al seu hospital i rebre d'altres del mateix per a informar-ho al coordinador de la regió.

En aquest sistema, l'agent REST es comunica amb tots els agents que formen el sistema de la regió, de manera que per part del coordinador de regió haurà d'informar-lo de les peticions i respostes que relacionen al sistema que representa la zona en la que es situa, i que per part dels diferents representants necessitarà informar a cadascun d'ells de les sol·licituds realitzades i les respostes dels diferents hospitals de

la regió. A continuació es representa en una taula les possibles peticions que es poden acceptar en el sistema de regió(Fig. 43):

URI	Tipus	Informació requerida	Descripció
/<hospital>/<level>	GET	Característiques de l'òrgan	Retorna la llista de receptors compatibles amb l'òrgan descrit depenent del nivell al que es vol realitzar la petició.
/<hospital>/hAgent	PUT	Característiques de l'òrgan i del pacient	Retorna la possible resposta per part de l'agent històric.
/<hospital>/ecAgent	POST	Característiques del pacient	Retorna la possible resposta per part de l'agent coordinador d'emergència.
/<hospital>/region/confirmation	GET	Característiques del pacient	Retorna una resposta respecte a la realització del trasplant al pacient descrit d'un hospital del país
/patients	GET	Característiques de l'òrgan	Retorna la llista de receptors compatibles amb l'òrgan descrit.
/confirmation	GET	Característiques del pacient	Retorna una resposta respecte a la realització del trasplant al pacient descrit d'un hospital de la regió.
/0Emergency/confirmation	GET	Característiques del pacient	Retorna la resposta d'un hospital respecte a la realització del trasplant a un pacient, que es troba en la llista de pacients crítics.

Fig. 43. Sol·licituds que es poden realitzar al sistema de regió

Com s'ha vist en la taula, existeix el cas d'una URI dinàmica amb dos variables, que representen l'hospital que realitza la sol·licitud i el nivell amb el que s'avisa si cal cercar pacients en la mateixa regió, en diferents regions, en altres zones o directament de la llista dels casos d'emergència 0. Partint de la representació de l'hospital, la regió pot saber a quin hospital no cal demanar la llista de pacients receptors compatibles, incloent a aquelles que es troben en la mateixa ciutat. També cal destacar que existeixen peticions molt paregudes, amb la diferència de que cadascuna d'elles les realitza un hospital de la regió o el sistema de la zona, tant per a obtindre informació sobre pacients compatibles com per a redirigir la confirmació de la realització d'un trasplant a un pacient.

Les sol·licituds que pot realitzar un sistema de regió, on la majoria consisteixen en redirigir peticions realitzades des d'un nivell a un altre, són les que es visualitzen en la taula que hi ha a continuació(Fig. 44):

Petició	Tipus	Sistema receptor	Dades addicional	Descripció
/<region>/0Emergency	GET	Zona	Característiques de l'òrgan	Demana la llista de pacients en estat crític compatibles amb l'òrgan.
/patients	GET	Hospital	Característiques de l'òrgan	Demana la llista de pacients compatibles amb l'òrgan a un hospital de la mateixa regió.
/<region>/zone	GET	Zona	Característiques de l'òrgan	Demana la llista de pacients compatibles amb l'òrgan d'hospitals de diferents regions.
/<region>/country	GET	Zona	Característiques de l'òrgan	Demana la llista de pacients compatibles amb l'òrgan d'hospitals de diferents zones.
/<region>/hAgent	PUT	Zona	Característiques de l'òrgan i del pacient	Demana afegir dades noves a l'agent històric.
/<region>/ecAgent	POST	Zona	Característiques del pacient	Demana afegir un pacient critic a la llista de pacients de l'agent coordinador d'emergència.

/confirmation	GET	Hospital	Característiques del pacient	Demana la confirmació a un hospital de la regió per a realitzar el trasplantament a un pacient.
/<region>/zone/confirmation	GET	Zona	Característiques del pacient	Demana la confirmació a un hospital de fora la regió per a realitzar el trasplantament a un pacient.
/0Emergency/confirmation	GET	Zona	Característiques del pacient	Demana la confirmació a l'hospital, on es troba el pacient obtingut de l'agent coordinador d'emergència.

Fig. 44. Llista de sol·licituds que es poden realitzar des d'un sistema de regió

A part, en el sistema que representa a una regió existeix una base de dades on s'emmagatzema la llista dels diferents hospitals amb els que es coordina, junt amb la seua llista de pacients en la espera d'un òrgan a trasplantar, obtingudes mitjançat les peticions que realitzen els hospitals, per a facilitar la cerca de l'hospital al que cal informar de la confirmació, que pot demanar un altre per a realitzar o no el trasplant. Aquesta base de dades es va actualitzant cada vegada que es realitza la cerca dels diferents llistats de pacients de cada hospital.

Per a la coordinació entre l'agent coordinador del sistema de regió i els representants de cada hospital, s'ha utilitzat el mateix protocol de comunicació que s'havia implementat en el sistema distribuït de l'apartat anterior, on s'utilitza un comportament del tipus *Time-Out* per a esperar durant un temps determinat en rebre totes les possibles respostes dels agents representants que hi ha en el sistema. Respecte a la resta de comunicació, s'ha utilitzat el mateix mètode que en el sistema d'hospital, que es igual als dels exemples que s'han mostrat anteriorment.

En la següent imatge(Fig. 45) es mostra com s'organitza un sistema multi-agent d'aquest tipus:

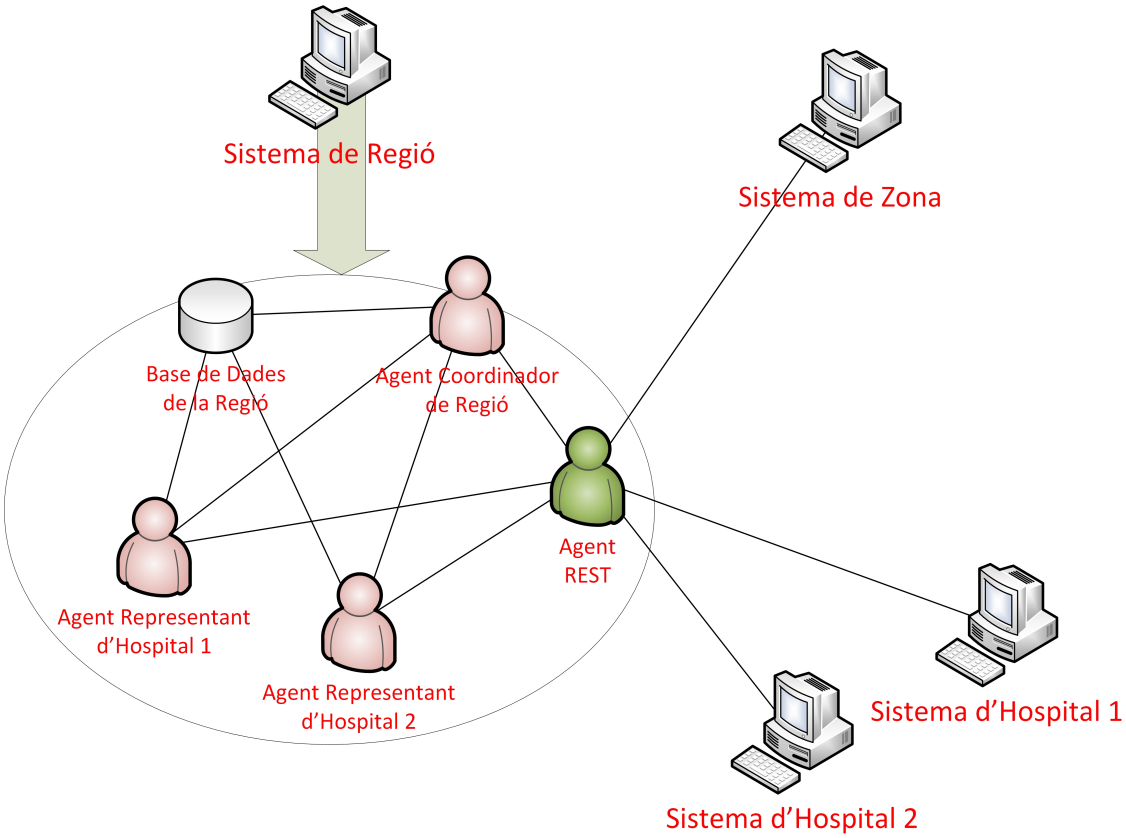

Fig. 45. Organització del sistema del tipus regió

A partir d'aquest moment, els sistemes que representen les zones i el coordinador nacional s'implementen d'una manera molt pareguda al del sistema de tipus regió, tant en la composició del sistema multi-agent com en les sol·licituds que poden realitzar i rebre, amb unes quantes diferències que s'explicaran més endavant en l'explicació de cadascun d'ells.

Els agents que formen el sistema de tipus zona són dels següents tipus:

Agent coordinador de zona. S'encarrega de la coordinació amb les seues regions, mitjançat les comunicacions que realitza amb cadascun dels seus representants situats en el mateix sistema. Es comunica amb l'agent REST per a contactar amb el sistema de coordinació nacional. En el sistema només pot existir un coordinador d'aquest tipus.

Agent representant de regió. És el tipus d'agent que representa a cadascuna de les regions que formen la zona. Tracta de comunicar-se amb el coordinador de la zona per a rebre ordres de peticions a realitzar a la seua regió i per a notificar-li de les respostes o sol·licituds obtingudes. Amb l'agent REST tracta de realitzar aquestes peticions ordenades pel coordinador o de rebre informació de la seua regió.

En el sistema també existeix una base de dades, controlada pel coordinador del sistema, on es guarda la llista de les diferents regions de la zona, amb els seus

hospitals. També s'emmagatzema una llista ordenada de tots els hospitals que hi han en la zona amb els seus pacients, on l'ordre que es segueix es basa en un protocol establert per la ONT, de manera que cada vegada que un hospital realitza el trasplant d'un òrgan que prové de la mateixa zona però d'una ciutat diferent, aquest hospital es posiciona al final de la llista. Partint d'aquesta última llista, la zona retorna a la regió una llista de pacients ordenada.

Respecte a l'agent REST en el sistema que representa la zona, també interactua amb tots els agents del sistema, on comunica als representants de cada regió sobre peticions que rep per part de les regions, realitza sol·licituds a aquests sistemes per part de cada representant, avisa al coordinador de la zona de les peticions realitzades per part del sistema coordinador de nació i realitza sol·licituds a aquest per part del coordinador del sistema. Les següents peticions són les que pot acceptar el sistema del tipus zona(Fig. 46):

URI	Tipus	Informació requerida	Descripció
/<region>/<level>	GET	Característiques de l'òrgan	Retorna la llista de receptors compatibles amb l'òrgan descrit depenent del nivell al que es vol realitzar la petició.
/<region>/hAgent	PUT	Característiques de l'òrgan i del pacient	Retorna la possible resposta per part de l'agent històric.
/<region>/ecAgent	POST	Característiques del pacient	Retorna la possible resposta per part de l'agent coordinador d'emergència.
/<region>/zone/confirmation	GET	Característiques del pacient	Retorna una resposta respecte a la realització del trasplant al pacient descrit d'un hospital fora de la zona.
/patients	GET	Característiques de l'òrgan	Retorna la llista de receptors compatibles amb l'òrgan descrit.
/confirmation	GET	Característiques del pacient	Retorna una resposta respecte a la realització del trasplant al pacient descrit d'un hospital de la zona.
/0Emergency/confirmation	GET	Característiques del pacient	Retorna la resposta d'un hospital respecte a la realització del trasplant a un pacient, que es troba en la llista de pacients crítics.

Fig. 46. Llista de peticions que pot acceptar el sistema de zona

Les accions que realitzen cadascuna d'aquestes sol·licituds són idèntiques a les del sistema de regió, amb la diferència de que s'executen a nivell de zona i l'obtenció de la llista de pacients compatibles amb l'òrgan, sol·licitat per part d'una regió, només es pot fer a nivell de zona o redirigint la petició al sistema de coordinació nacional, on pot cercar en altres zones o demanar la llista de pacients a l'agent coordinador d'emergència.

Les peticions que pot realitzar aquest sistema són també paregudes al que representa una regió(Fig. 47), descartant-ne aquella que fa referència a la cerca de pacients compatibles de la mateixa zona, ja que no té sentit la sol·licitud en aquest nivell del sistema distribuït.

Petició	Tipus	Sistema receptor	Dades addicional	Descripció
/<zone>/0Emergency	GET	Coordinador nacional	Característiques de l'òrgan	Demana la llista de pacients en estat crític compatibles amb l'òrgan.
/patients	GET	Regió	Característiques de l'òrgan	Demana la llista de pacients compatibles amb l'òrgan d'hospitals de una regió de la zona.
/<zone>/country	GET	Coordinador nacional	Característiques de l'òrgan	Demana la llista de pacients compatibles amb l'òrgan d'hospitals de diferents zones.
/<zone>/hAgent	PUT	Coordinador nacional	Característiques de l'òrgan i del pacient	Demana afegir dades noves a l'agent històric.
/<zone>/ecAgent	POST	Coordinador nacional	Característiques del pacient	Demana afegir un pacient critic a la llista de pacients de l'agent coordinador d'emergència.
/confirmation	GET	Regió	Característiques del pacient	Demana la confirmació a un hospital de la zona per a realitzar el trasplant al pacient.

/ <zone>/country/ confirmation	GET	Coordinador nacional	Característiques del pacient	Demana la confirmació a un hospital de fora la zona per a realitzar el trasplant a un pacient seu.
/ 0Emergency/co nfirmation	GET	Coordinador nacional	Característiques del pacient	Demana la confirmació a l'hospital, on es troba el pacient obtingut de l'agent coordinador d'emergència.

Fig. 47. Peticions que pot formalitzar el sistema de zona

Com es pot vore en la taula anterior, la majoria de les sol·licituds que realitza la zona consisteixen en redirigir les sol·licituds que rep per part de les regions que el formen cap al coordinador nacional i que rep per part del coordinador nacional per a dirigir-ho a les diferent regions. La resta de peticions es realitzen de la mateixa manera com s'ha explicat en el sistema de coordinació anterior.

El desenvolupament de les comunicacions d'aquest sistema és idèntic al del sistema del tipus regió. A continuació es mostra la representació, tant dels agents com de les comunicacions, del sistema representant de la zona(Fig. 48):

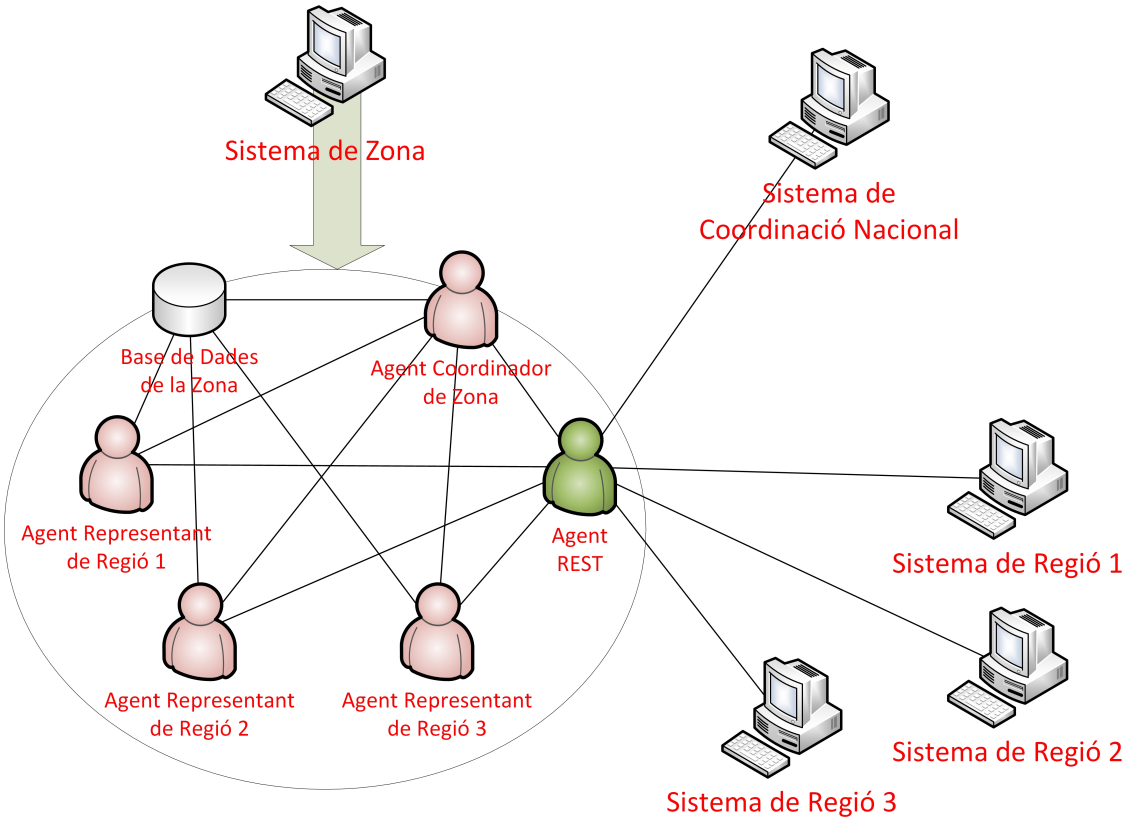

Fig. 48. Representació del sistema de zona

Finalment, l'últim tipus de sistema que s'utilitza i es situa en el nivell més alt del sistema distribuït de la coordinació de trasplantament d'òrgans, que es tracta del coordinador nacional, es compon també per dos tipus d'agent:

Agent coordinador de la nació. Aquest agent s'utilitza per a realitzar comunicacions amb les diferents zones del país, per mitjans de cadascun dels seus representants. A diferència dels coordinadors dels altres sistemes, aquest no interactua directament amb l'agent REST, sinó a partir d'intermediaris, que són els representants de cada zona.

Agent representant de zona. S'utilitza per a rebre informació del sistema del tipus zona al que representa l'agent en el sistema de coordinació nacional. Es comunica amb l'agent coordinador de la nació per a realitzar ordres al sistema de zona i notificar-lo de les sol·licituds i respostes obtingudes del nivell inferior. També es comunica amb els agents coordinador d'emergència i històric per a actualitzar diferents dades o obtindre la llista dels pacients que esperen d'un òrgan i estan en estat crític.

Agent coordinador d'emergència. Tal com s'havia explicat abans, és l'agent coordinador dels casos d'emergència 0. Es comunica amb els representants de cada zona, per a rebre les peticions i actualitzacions a realitzar sobre la seua llista de pacients en estat crític.

Agent històric. És el que s'encarrega de rebre informació de tots els trasplantaments que s'hagen realitzat en Espanya. També es comunica amb els agents representants, per a actualitzar la seua base de dades de trasplantaments realitzats amb èxit.

La base de dades que utilitza l'agent coordinador d'emergència està formada per una llista de pacients amb les dades necessàries per a conéixer per exemple l'òrgan que necessita o l'hospital on es troba amb la seua adreça web per a que es puga comunicar directament amb l'hospital si el pacient s'escollix per a realitzar el trasplant. Respecte a la base de dades de l'agent històric, es compon d'una llista amb els diferents casos dels trasplantaments d'òrgan realitzat en el país, detallant la informació del pacient receptor, l'òrgan donant i els hospitals que s'han encarregat del trasplant, tant del transport de l'òrgan com de l'equip quirúrgic. També existeix una tercera base de dades, on s'emmagatzema la llista de les zones ordenades que componen el país, on l'ordre utilitzat, que també està establert per la ONT, situa al final de la llista aquella zona on s'haja realitzat el trasplant d'un òrgan, que prové d'un hospital que es troba en una zona distinta. Al igual que en el nivell de zona, aquesta llista permet retornar a l'hospital que ho demana el conjunt de pacients receptors compatibles seguint un ordre.

L'agent REST en aquest tipus de sistema només necessita comunicar-se amb els representants de cada zona, ja que les úniques comunicacions que es poden realitzar fan referència als sistemes del nivell inferior. Per al sistema de coordinació nacional només s'accepten les següents peticions(Fig. 49):

URI	Tipus	Informació requerida	Descripció
/<zone>/<level>	GET	Característiques de l'òrgan	Retorna la llista de receptors compatibles amb l'òrgan descrit depenent del nivell al que es vol realitzar la petició.
/<zone>/hAgent	PUT	Característiques de l'òrgan i del pacient	Retorna la possible resposta per part de l'agent històric.
/<zone>/ecAgent	POST	Característiques del pacient	Retorna la possible resposta per part de l'agent coordinador d'emergència.
/<zone>/country/confirmation	GET	Característiques del pacient	Retorna una resposta respecte a la realització del trasplant al pacient descrit d'un hospital del país.
/0Emergency/confirmation	GET	Característiques del pacient	Retorna la resposta d'un hospital respecte a la realització del trasplant a un pacient seu, que es troba en la llista de pacients crítics.

Fig. 49. Llista de peticions que admet el sistema de coordinació nacional

En aquest nivell, el sistema pot retornar la llista de possibles receptors tant de la base de dades de l'agent coordinador d'emergència com de la resta de zones a les que ho ha demanat. A part d'actualitzar les bases de dades dels diferents agents, també permet retornar la resposta feta per aquell hospital que conté un receptor, al que s'ha acceptat per a realitzar el trasplant d'òrgan, de manera que abans ha de reenviar la petició a la zona on es troba aquest hospital.

En quant a les sol·licituds que es poden realitzar des del coordinador nacional, només fan falta les que es mostren en la següent taula per a redirigir peticions a altres sistemes(Fig. 50):

Petició	Tipus	Sistema receptor	Dades addicional	Descripció
/patients	GET	Zona	Característiques de l'òrgan	Demana la llista de pacients compatibles amb l'òrgan d'hospitals de una zona.
/confirmation	GET	Zona i Hospital	Característiques del pacient	Demana la confirmació a un hospital de una zona per a realitzar el trasplant al pacient.

Fig. 50. Sol·licituds que pot realitzar el sistema de coordinació nacional

Cadascuna d'aquestes sol·licituds es realitzen quan el sistema obté peticions referents a l'obtenció del possibles receptors d'un òrgan i a la confirmació d'un trasplant a la resta de zones, degut a que no s'han pogut resoldre en les capes inferiors.

Respecte a la implementació de la forma en que es comuniquen els agents d'aquest sistema, s'utilitzen els mateixos mètodes que en els sistemes coordinadors del tipus regió i zona. En la següent imatge es mostra com s'organitza aquest sistema multi-agent(Fig. 51). Cal tindre en compte que en aquest imatge s'ha reduït el número de representants de zones, degut a que podria dificultar l'enteniment de l'arquitectura.

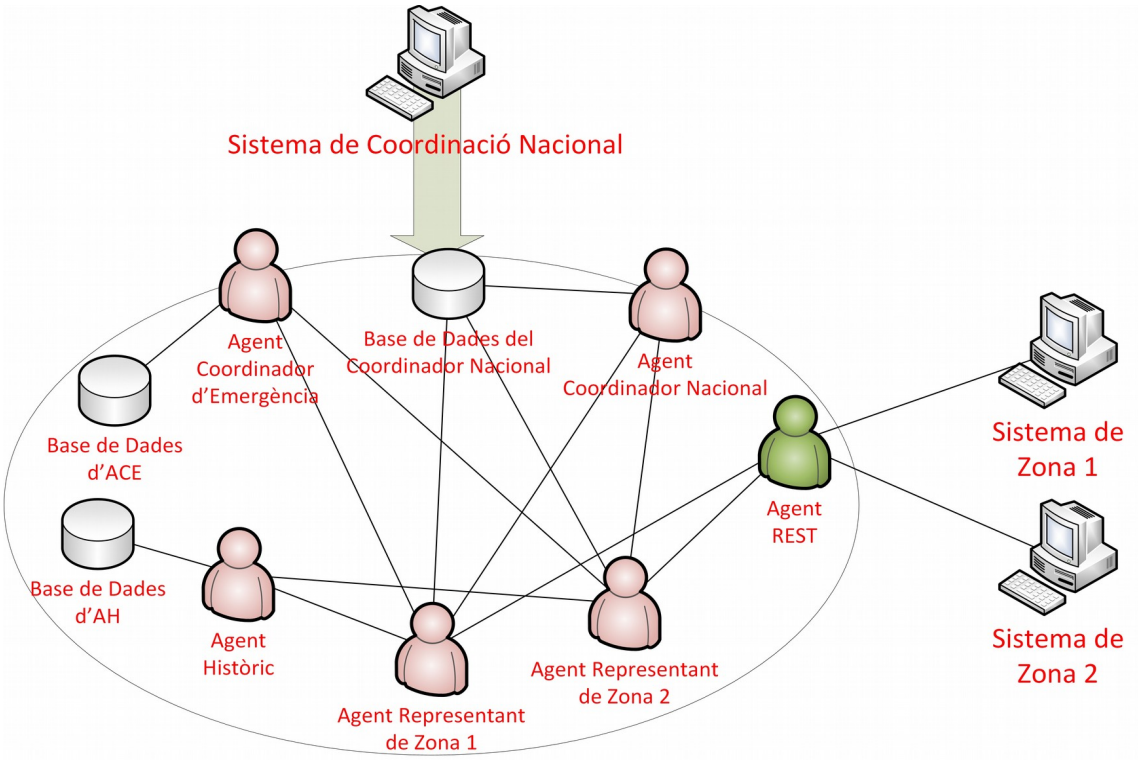

Fig. 51. Organització del sistema de coordinació nacional

Una vegada explicat com estan dissenyats cada tipus de sistema multi-agent que formen el sistema distribuït, es contarà el procés de coordinació que es realitza entre els diferents nivells de la distribució per a que un hospital tracte de trobar i confirmar el trasplant d'un òrgan que tenen disponible.

El primer pas consisteix en comprovar en cada hospital si en la seua base de dades existeix un òrgan disponible, on el metge ho confirma actualitzant la base de dades d'òrgans i l'agent coordinador de trasplantaments ho comprova, demanant-ho a l'agent representant de la base de dades. Si n'hi ha un òrgan disponible, l'agent coordinador interactuarà amb l'agent REST per a comunicar-se amb l'agent coordinador d'emergència.

Per a comunicar-se amb aquest agent, l'hospital realitza una petició al sistema de la regió, que al mateix temps ho informa al seu agent coordinador per a que reenvie la sol·licitud al sistema de la zona, que realitza el mateixos passos per a realitzar la petició al sistema de coordinació nacional. En aquest sistema, es comunica de l'avís a l'agent coordinador d'emergència, que realitzarà una llista d'aquells pacients de la seua base de dades que són compatibles amb l'òrgan disponible. Una vegada realitzada la llista, el sistema de coordinació nacional retorna la resposta al sistema de la zona, que avisa al seu coordinador per a retornar aquesta informació al sistema de la regió que ho havia demanat, que al mateix temps ho envia a l'hospital on tenen l'òrgan.

Una vegada que l'hospital tinga la llista de possibles receptors, l'agent coordinador de trasplantaments aplica en ella el procés SMART per a decidir quin serà el primer pacient al que s'ha de demanar permís de realitzar el trasplant d'òrgan. Una vegada decidit, es torna a intentar comunicar-se amb l'agent coordinador d'emergència per a poder informar de la decisió a l'hospital on es troba el pacient, que després retorna la seua resposta a l'hospital que ho demana. Si l'hospital del pacient accepta la realització del trasplant, el procés de coordinació finalitza amb l'actualització de la base de dades que manipula l'agent històric, de manera que s'envia una petició al sistema de coordinació nacional per a actualitzar-lo. En el cas de que l'altre hospital es nega a realitzar el trasplant al seu pacient, l'agent coordinador de trasplantaments repeteix el procés de confirmació per al següent pacient receptor escollit de la llista. Si la llista es queda buida, sense haver pogut trobar cap pacient receptor que hagen acceptat en el seu hospital per a realitzar el trasplant, l'agent coordinador de trasplantament consulta en la base de dades de pacients del seu hospital, per a comprovar si existeix algun receptor compatible amb l'òrgan.

Si no s'han pogut trobar receptors compatibles o l'hospital no accepta realitzar els trasplantaments en aquells possibles pacients, l'agent coordinador intentarà demanar els pacients compatibles dels hospitals de la mateixa ciutat, els quals es poden comunicar entre ells sense haver d'interactuar amb el sistema representant de la regió. Una vegada obtingut els possibles receptors, es torna a aplicar el procés SMART, de manera que es torna a interactuar amb els hospitals de la ciutat per a obtindre les seues confirmacions. Si en els hospitals de la mateixa ciutat tampoc es troba un pacient receptor compatible amb l'òrgan que accepten en el seu hospital el trasplant, l'agent coordinador de l'hospital tractarà de demanar a la regió la llista de pacients.

Quan l'agent coordinador de la regió reba aquesta sol·licitud per part del seu representant en el sistema, que s'encarrega de rebre les peticions que realitza el seu hospital, demanarà a la resta de representants que li donen la llista de pacients receptors compatibles amb l'òrgan disponible, excepte aquells representants d'hospitals que es troben en la mateixa ciutat d'aquella que ho ha sol·licitat. Una vegada que el representant de la regió ha rebut totes les llistes, les retorna a l'hospital que ho demanava. Partint d'aquests pacients, l'agent coordinador de l'hospital intentarà comunicar-se amb amb l'hospital on es troba el pacient escollit per a realitzar el trasplant, demanant-ho primer al sistema de la regió per a que redirigisca la petició a l'hospital. Al igual que en els anteriors casos, si tampoc accepten el trasplant cap hospital de la regió, aleshores es realitza la cerca en les diferents regions de la zona, on no es troba l'hospital que ho demana.

Per a aquesta cerca, es realitza la sol·licitud de cerca al sistema de la regió, que s'encarrega d'avisar a l'agent representant de la regió per a que redirigisca la petició al

sistema de tipus zona. El representant de la zona intentarà comunicar-se amb els diferents representants de les distintes regions per a que obtinguen la llista de pacients compatibles, on al mateix temps cada regió diferent realitza la cerca en els seus hospitals. Una vegada que el representant de la zona obtinga les diferents llistes de cada regió, ordena la llista de pacients segons l'hospital, seguint el protocol establit per la ONT, i la retorna al sistema de la regió, que al mateix temps ho envia a l'hospital que ho demanava. Si d'aquesta llista no s'accepta tampoc cap pacient per part dels hospitals on es troben, l'hospital realitza una recerca en la resta d'hospitals als que encara no s'han demanat la seua consulta.

Per a aquesta última cerca, l'agent coordinador de trasplantaments realitza la sol·licitud d'obtindre els receptors possible de la resta del país al sistema de coordinació de nació, que abans s'ha de redirigir pels nivells que hi han entre ells. Una vegada que l'agent coordinador del sistema del nivell nacional reba la petició, el mateix agent demana a la resta d'agents representants de zones que es comuniquen amb les seues zones per a obtindre els diferents pacients de cada hospital compatibles amb l'òrgan que està disponible. Una vegada que l'agent coordinador obtinga les llistes de les distintes zones, les ordena segons les zones i el protocol de la ONT, de manera que ho retorna a la zona que ho demanava, que al mateix temps ho retorna a la regió que també ho sol·licitava, que ho retorna a l'hospital que havia realitzat la petició. Si partint d'aquesta llista de possibles receptors no es confirma cap realització de trasplant, l'agent coordinador de trasplantament informa a l'agent interfície de que comunique a l'hospital que no s'ha pogut trobar cap pacient receptor compatible.

El codi referent al desenvolupament de cadascun dels sistemes multi-agent d'aquest sistema distribuït es troba en un repositori public[7]. La implementació s'ha realitzat amb algunes excepcions que no s'han pogut realitzar com toca, de manera que s'han simplificat en el seu desenvolupament, com són els casos del procés SMART en la selecció del possible pacient al que s'ha de realitzar el trasplanta d'òrgan o la forma en la que ha de rebre les dades l'agent històric. També s'ha descartat la implementació en el sistema d'hospital de l'agent traductor d'ontologies. Aquest agent se suposa que permet traduir la informació que es rep d'un hospital extern a un llenguatge que es puga entendre en l'hospital on es situa, però com que tots els sistemes d'hospital que s'utilitzen gasten el mateix llenguatge en la informació que envien, sa descartat aquest agent dins del sistema.

En aquest sistema distribuït, l'API web implementada en cada agent REST és semblant als que s'han utilitzat en el sistema distribuït de l'obtenció de la mitjana de temperatures de diferents zones, explicat en l'apartat anterior. Les úniques diferències

7 GitHub. *SPADE-with-REST/hospital_coordinator/*. https://github.com/joasanm/SPADE-with-REST/tree/master/hospital_coordinator

que hi han són que cada petició s'envia amb un conjunt de dades i que alguns mètodes dels receptors de sol·licituds permeten emmagatzemar aquestes dades en les seues bases de dades. Encara que existeixin aquest canvis, les APIs creades compleixen amb totes les restriccions, per a que es puguen denominar del tipus Restful.

4.2.3 Validació

Respecte a les proves realitzades, no s'ha pogut trobar una mètrica per a analitzar el funcionament el sistema distribuït, encara que s'ha pogut comprovar l'estabilitat de les comunicacions entre diverses combinacions de sistemes multi-agent sense utilitzar tots els sistemes distints, degut a la falta de màquines físiques i potència de còmput per a executar al mateix temps un sistema distribuït amb tots els sistemes, tant d'hospitals com de diferents coordinadors, que facin falta en cada nivell.

Una vegada explicat tot l'anterior, ja es poden treure les conclusions obtingudes amb cada part realitzada en aquest projecte, que s'explicaran en el següent apartat.

5 Conclusions

5.1 Objectius complerts

La idea d'implementar aquest canal de comunicació havia sorgit per la necessitat de permetre la interacció entre diferents plataformes multi-agent, de manera que s'ha arribat a implementar en una plataforma aquest protocol de comunicació, que es basa en els principis de REST.

D'acord amb el subobjectius indicats al principi de la memòria, s'ha realitzat un anàlisi de les diferents plataformes de sistemes multi-agent que existeixen actualment, on s'ha arribat a la conclusió d'utilitzar la plataforma de SPADE per a afegir-li el nou canal de comunicació.

Després s'ha analitzat i comparat els diferents *frameworks* web que es pogueren utilitzar amb la plataforma multi-agent de SPADE, de manera que s'ha escollit Flask per a desenvolupar el canal de comunicació seguint el protocol REST.

Partint dels subobjectius anteriors, s'ha implementat el mòdul REST mitjançat la coordinació d'un agent del sistema de SPADE i un servei web de Flask, de manera que es poguera comunicar amb la resta dels agents de la plataforma i amb sistemes externs.

Una vegada realitzat aquest mòdul, s'ha dissenyat un exemple bàsic de comunicació entre dos sistemes de la mateixa plataforma, però en diferents màquines per a comprovar el seu funcionament i corregir diferents aspectes que poden ocasionar problemes. L'exemple dissenyat consisteix en que un sistema realitza una sol·licitud a l'altra per a que aquesta li torne una cadena de caràcters.

S'ha creat després un sistema distribuït jeràrquic complexe, on un sistema multi-agent ha d'interactuar amb altres sistemes mitjançat el mòdul de comunicació REST dissenyat. El sistema dissenyat consisteix en obtindre la mitjana de temperatures d'aquells valor que li tornen diferents sistemes, que al mateix temps ho demanen a altres. També s'han realitzat proves, amb el *hardware* que es tenia disponible en aquell moment, depenent del número d'agents que obtenen la seua temperatura distribuïts en diferents quantitats de sistemes, situats en la part inferior de la distribució jeràrquica. Partint d'aquestes proves s'ha comprovat que hi ha més estabilitat en el funcionament dels diversos agents d'un sistema quan aquest està format per menys agents, de manera que es millor utilitzar diferents sistemes d'agents que al mateix temps es puguen interactuar entre ells.

Finalment, s'ha implementat un altre sistema distribuït, però d'àmbit més professional, que consisteix en la coordinació en el trasplant d'òrgans entre diferents hospitals d'un país. El desenvolupament d'aquest sistema s'ha basat en el sistema de coordinació implementat en el subobjectiu anterior, i en el disseny explicat en un article realitzat entre diferents departaments de distintes universitats, de manera que funciona correctament i es possible aplicar en el sistema de coordinació que s'utilitza actualment en Espanya.

Com a resum, s'ha aconseguit implementar un nou agent en la plataforma de SPADE que permet comunicar-se amb altres sistemes de la mateixa plataforma multi-agent mitjançat el protocol REST.

5.2 Treballs futurs

En aquest projecte s'han realitzat diferents tasques per a poder realitzar els objectius proposats, de manera que s'han obtingut els resultats esperats i que al mateix temps permeten continuar amb el seu desenvolupament del projecte. Els possibles treballs futurs que es podrien realitzar són els següents:

Millorar el mòdul de comunicació realitzat. El sistema de comunicació implementat funciona correctament, però s'han detectat casos en els que es ralentitza el seu funcionament, degut a la recepció simultània de diferents sol·licituds per part de molts sistemes, que obliguen al mòdul utilitzar una cua de peticions. Per a solucionar aquest problema, s'ha pensat proposar un treball futur on s'intentarà paral·lelitzar el procés de recepció de peticions, el qual pot demanar tornar a implementar moltes parts del mòdul REST.

Implementar el canal de comunicació en altres plataformes. Com que ha tingut èxit el desenvolupament del canal de comunicació REST en la plataforma de SPADE, es pot continuar implementant el mateix canal a altres plataformes de sistemes multi-agent, com poden ser Magentix2, JADE o Jack.

Realitzar proves amb altres serveis. Amb aquest treball futur es planteja utilitzar el mateix canal de comunicació per a que puguen interactuar una plataforma multi-agent amb altres serveis web o dispositius, com poden ser aquells relacionats amb la Internet de les coses.

Utilitzar el mòdul REST en un escenari real. Encara que s'ha utilitzat el mòdul en diferents sistemes distribuït, aquests consistien en entorns simulats i controlats. Aquest treball futur consistiria en utilitzar la plataforma de SPADE en un entorn real amb diferents sistemes distribuïts, com és el cas de l'ultim sistema implementat, que es podria utilitzar en el camp de la medicina.

6 Bibliografia

[1]: Juan Cuesta Contreras, Borja Muñoz Bosch. *Plataformas de Agentes Inteligentes.*

[2]: Ana María Arrufat Sánchez. *Plataformas de agentes.*

[3]: GTI-IA. *Magentix2.* http://www.gti-ia.upv.es/sma/tools/magentix2/ [Consulta: 31 de març de 2017]

[4]: JADE Site. *JAVA Agents DEvelopment Framework.* http://jade.tilab.com/ [Consulta: 31 de març de 2017]

[5]: AOS Group. *JACK.* http://www.agent-software.com.au/products/jack/ [Consulta: 24 d'abril de 2017]

[6]: Jose Sebastián Canós. *La plataforma de agentes SPADE.*

[7]: Jomi Fred Hübner, Rafael Heitor Bordini. *Jason | A Java-based interpreter for an extended version of AgentSpeak.* http://jason.sourceforge.net/wp/ [Consulta: 8 de maig de 2017]

[8]: Wikimedia Foundation Inc.. *Representational state transfer.* https://en.wikipedia.org/wiki/Representational_state_transfer [Consulta: 15 d'abril de 2017]

[9]: Roy Thomas Fielding. 2000. *Architectural Styles and the Design of Network-based Software Architectures.* pp. 76-86.

[10]: Todd Fredrich. *REST API Tutorial.* http://www.restapitutorial.com/ [Consulta: 10 de febrer de 2016]

[11]: Juha Koskelainen. *WebFrameworks.* https://wiki.python.org/moin/WebFrameworks/ [Consulta: 16 de febrer de 2017]

[12]: Armin Ronacher. *Flask (A Python Microframework).* http://flask.pocoo.org/ [Consulta: 6 de maig de 2016]

[13]: Marcel Hellkamp. *Bottle: Python Web Framework.* http://bottlepy.org/docs/dev/index.html [Consulta: 6 de maig de 2016]

[14]: Falcon Contributors. *Falcon - The minimalist Python WSGI framework.* https://falconframework.org/ [Consulta: 8 de maig de 2016]

[15]: Django Software Foundation. *The Web framework for perfectionists with deadlines | Django.* https://www.djangoproject.com/ [Consulta: 9 de maig de 2016]

[16]: The CherryPy team. *CherryPy - A Minimalist Python Web Framework.* http://cherrypy.org/ [Consulta: 9 de maig de 2016]

[17]: Agendaless Consulting. *The Pyramid Web Framework.* http://docs.pylonsproject.org/projects/pyramid/en/latest/index.html [Consulta: 2 de maig de 2017]

[18]: TurboGears2 Website Doc Team. *The Web Framework that scales with you. - TurboGears Website.* http://www.turbogears.org/index.html [Consulta: 2 de maig de 2017]

[19]: web2py Team. *web2py Web.* http://www.web2py.com/ [Consulta: 9 de maig de 2017]

[20]: The Tornado Authors. *Tornado Web Server.* http://www.tornadoweb.org/en/stable/ [Consulta: 9 de maig de 2017]

[21]: Kenneth Reitz. *Requests: HTTP for Humans.* http://docs.python-requests.org/en/master/ [Consulta: 3 de març de 2017]

[22]: A. Aldea, B. López, A. Moreno, D. Riaño, A. Valls. *A Multi-Agent System for Organ Transplant Coordination.*

7 Glossari

API: *Aplicacion Programing Interface*. Conjunt de subrutines, funcions i procediments que ofereix una biblioteca de programació per a ser utilitzada per un altre *software* com una capa d'abstracció.

BDI: *Belief Desire Intention*. Model de *software* desenvolupat per a la programació d'agents amb intel·ligència artificial.

Cacheable: Acció de poder emmagatzemar informació en una memòria cau.

Cookie: Informació enviada per un servidor web i emmagatzemada en el navegador de l'usuari, que pot ser retornada pel navegador en posteriors accessos al mateix servidor.

FIPA: *Foundation for Intelligent Physical Agents*. Organització que desenvolupa i estableix estàndards de *software* per a agents intel·ligents heterogenis que interactuen i sistemes basats en agents.

Framework: Estructura conceptual i tecnològica d'assistència definida amb mòduls concrets de *software,* que poden servir de base per a l'organització i desenvolupament de *software* i que pot incloure suport de programes, llibreries i un llenguatge interpretatiu per a poder crear un projecte.

Hiperenllaç: Element de referència o navegació en un document electrònic que fa referència a un altre recurs.

Hipermèdia: Terme que es designa al conjunt de mètodes o procediments per a escriure, dissenyar o compondre continguts que integren suports tals com texts, imatges, vídeos, àudios, mapes, de manera que el resultat obtingut tinga la possibilitat d'interactuar amb l'usuari.

Hipertext: Sistema d'organització de la informació basat en la possibilitat de moure's per dins d'un text i cap a altres de diferents mitjançat paraules clau.

Host: Terme que s'utilitza per a fer referència a computadors o altres dispositius connectats en una xarxa, que proveïxen i utilitzen serveis en ella.

HTTP: *Hipertext Transfer Protocol*. Protocol de comunicació que permet la transferència d'informació en la *World Wide Web*.

Jabber: Protocol de missatgeria instantània, conegut actualment com XMPP.

JSON: *JavaScript Object Notation*. Format de text lleuger per a l'intercanvi de dades, considerat com un llenguatge independents degut al seu ús alternatiu al XML.

Memòria cau: Memòria d'alta velocitat instal·lada en el mateix processador on s'emmagatzemen aquelles dades que necessiten ser llegides pel processador amb més freqüència.

Metadada: Dada que descriu continguts informatius d'altres dades.

Middleware: *Software* que assisteix a una aplicació per a poder interactuar o comunicar-se amb altres aplicacions, paquets de programes, xarxes, *hardware* o sistemes operatius. Funciona com una capa d'abstracció de *software* distribuït que es situa entre les capes d'aplicacions i inferiors del sistema operatiu i xarxa.

Mime-type: Identificador amb dos parts, formats de l'arxiu i format del contingut, que es transmet per Internet.

MTP: *Media Transfer Protocol*. Conjunt d'extensions sobre el protocol de transferència d'imatges, que permeten el seu ús en altres dispositius, com poden ser càmeres digitals, reproductors multimèdia o dispositius digitals portàtils en general.

Plugin: Complement d'una aplicació que es relaciona amb una altra per a afegir una nova funció a un programa

SASL: *Simple Authentication and Security Layer*. *Framework* per a l'autentificació i l'autorització en protocols de Internet.

Script: Programa usualment simple que s'emmagatzema en un arxiu de text pla que s'executa directament des del seu codi font.

Socket: Concepte abstracte on dos programes, situats en computadors diferents, poden intercanviar qualsevol flux de dades de manera fiable i ordenada.

SSL: *Segure Sockets Layer*. Protocol criptogràfic que proporciona comunicacions segures en una xarxa, antecessor del protocol TLS.

TLS: *Transport Layer Security*. Protocol criptogràfic que proporciona la comunicació segura per una xarxa.

URI: *Uniform Resource Identifier*. Cadena de caràcters que identifica els recursos d'una xarxa de manera unívoca.

XML: *Extensible Markup Langage*. Meta-llenguatge que permet definir llenguatges de marques, utilitzat per a desar dades de forma que es puguen llegir.

XMPP: *Extensible Messaging and Presence Protocol*. Protocol obert basat en XML idealitzat per a la missatgeria instantània. Abans conegut com Jabber.