UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

LENGUAJE DE ESPECIFICACIÓN PARA LA DELEGACIÓN DE TAREAS EN SERVIDORES WEB MEDIANTE AGENTES

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS, MENCIÓN COMPUTACIÓN

TEÓFILO CHAMBILLA AQUINO

PROFESOR GUÍA:
CLAUDIO GUTIÉRREZ GALLARDO

MIEMBROS DE LA COMISIÓN:
AIDAN HOGAN
PABLO BARCELÓ BAEZA
MARCELO ARENAS SAAVEDRA

Este trabajo ha sido financiado por PRONABEC

SANTIAGO DE CHILE
2016

Resumen

La tecnología de los agentes se ha convertido en la base de una gran cantidad de aplicaciones ya que permite la incorporación de bases de conocimiento de acciones y tareas para resolver problemas complejos. Por otro lado, se sabe que los Servidores Web se sustentan en el protocolo HTTP, protocolo que solo permite las solicitudes y respuestas entre Cliente y Servidor y no delegar funciones a otros Servidores separados geográficamente.

Esta investigación consiste en un estudio exploratorio del concepto de la delegación en el contexto de la Web, donde agentes que residen en diferentes Servidores Web puedan cooperan entre sí para resolver tareas complejas. Para ello, se propone un lenguaje de especificación para la delegación de tareas en Servidores Web mediante agentes, con propiedades necesarias para su autonomía y que puedan ser utilizados con flexibilidad en entornos distribuidos bajo la restricción del protocolo de comunicación HTTP.

En primer lugar, se presenta el modelo abstracto de la delegación en el entorno de la Web y los componentes necesarios para la elaboración del lenguaje especificación propuesto, mediante la definición de acciones básicas y opcionales que son implementadas por los agentes participantes en el proceso de la delegación.

En segundo lugar, como caso de estudio, se desarrolla la implementación de NautiLOD de manera distribuida mediante agentes. NautiLOD es un lenguaje de expresión declarativo que está diseñado para especificar patrones de navegación en la red Linked Open Data, donde sus primeras propuestas de implementación han sido con un enfoque centralizado.

En un tercer lugar, se presenta *Agent Server*, una plataforma flexible y escalable para Sistema MultiAgentes basados en el ambiente de la Web, desarrollado bajo los principios de REST, que permite gestionar agentes distribuidos.

La principal conclusión de la tesis es la validación del lenguaje de especificación en una plataforma homogénea como es Linked Data que gracias a su semántica permite a los agentes procesar su contenido, razonar sobre este y realizar deducciones lógicas. Esto se realizó con consultas propias en los Endpoints SPARQL expresados en NautiLOD.

Agradecimientos

Antes que todo quiero agradecer a Dios por haberme dado una vida, por haberme acompañado y guiado a lo largo de mis estudios, por ser mi fortaleza en los momentos de debilidad y por brindarme una vida llena de aprendizajes, experiencias y sobre todo felicidad.

Quiero agradecer a mi padre quien siempre me enseñó que todo se puede lograr con el esfuerzo. A mi madre, cuyo recuerdo llevo en mi corazón.

Agradecimientos a los miembros de mi familia y amigos. A mi hermana Jacinta quien siempre cumplió el rol de segunda madre y me inculcó los principios y valores para enfrentar la sociedad. A mi novia Roxana, por brindarme el amor de pareja, el apoyo de compañera, por su paciencia, por darme esos ánimos de fuerza y valor para seguir adelante y por su valentía de superar todo obstáculo durante estos dos años de esfuerzo. A mi mejor amigo Joffre por los grandes momentos vividos durante las clases. A Daniel H. que siempre estuvo dispuesto en responder cualquier duda. A mi amigo Hernan, por sus consejos y comentarios imparciales. A los amigos becarios de Pronabec Luis Angel, Richard, Andy, Iris, Yenny quienes en todo momento me hicieron sentir como si estuviésemos en casa.

Al DCC de la Universidad de Chile en pleno, por brindar al extranjero un lugar donde pueda sentirse como en casa. A los profesores, por el apoyo y conocimientos entregados. Un agradecimiento especial a Angélica, por apoyo incondicional desde el primer día de clases.

Por ultimo, un agradecimiento especial a Profesor Claudio Gutierrez, profesor guía, amigo y consejero. Por su paciencia en todas las reuniones que tuvimos y por sus grandes enseñanzas.

Teófilo

Referencia de siglas y abreviaturas

- **HTTP**: Hypertext Transfer Protocol.
- **BDI**: Belief–Desire–Intention.
- **SMA**: Sistema MultiAgente.
- **REST**: Representational State Transfer.
- **SOAP**: Simple Object Access Protocol.
- **SPARQL**: SPARQL Protocol and RDF Query Language.
- **FIPA**: Foundation for Intelligent Physical Agents.
- **ACL**: Agent Communication Language .
- **JADE**: Java Agent Development Framework.
- **KIF**: Knowledge Interchange Format.
- **KQML**: Knowledge Query Manipulation Language.
- **NautiLOD**: Nauti Linked Open Data.
- **AUML**: Agent Unified Modeling Language

Tabla de contenido

Resumen I

Agradecimientos II

Referencia de siglas y abreviaturas III

1. Introducción 1

1.1. Objetivo General . . . 2

1.2. Objetivos específicos . . . 2

1.3. Metodología . . . 3

1.4. Contribuciones . . . 3

1.5. Estructura de este documento . . . 4

2. Antecedentes preliminares 5

2.1. Agentes . . . 5

2.2. Características de los Agentes . . . 6

2.3. Arquitecturas de Agentes . . . 7

2.4. Sistema MultiAgente . . . 8

2.5. Comunicación entre agentes . . . 9

2.6. Plataformas Sistema MultiAgente . . . 11

2.7. Delegación de tareas . . . 12

2.7.1. Tipos de delegación . . . 13

2.7.2. Estrategias de delegación . . . 13

2.8. Lenguajes de Especificación de Tareas . . . 14

3. Especificación Formal 16

3.1. Modelo abstracto de la delegación en la Web . . . 16

3.1.1. Agentes . . . 18

3.1.2. Acciones . . . 18

3.1.3. Tareas . . . 18

3.2. Definición de acciones 18
3.3. Lenguaje y protocolo de especificación 20
3.3.1. Lenguaje 20
3.3.2. Delegación como un acto comunicativo 21
3.3.3. Sintaxis 22
3.3.4. Semántica 23
3.3.5. Especificación de Protocolos 23
3.3.6. Protocolo de delegación con *Attribute Global Types* 25

4. Caso de estudio **29**
4.1. Descripción NautiLOD Distribuido 29
4.2. Arquitectura para NautiLOD Distribuido 30
4.3. Ejecución de NautiLOD Distribuido 33
4.4. Resultado preliminar de NautiLOD Distribuido 36

5. Plataforma Agent Server **41**
5.1. Descripción de Agente Server 41
5.2. Background y trabajos relacionados 42
5.3. Arquitectura de Agent Server 43
5.4. Funcionamiento de Agent Server 45
5.5. Implementación de Agent Server 47
5.6. Aplicación de la plataforma Agent Server 50

6. Resultados y Discusión **52**

7. Conclusiones y trabajos futuros **54**

Apéndices **56**

Bibliografía **57**

Índice de figuras

2.1. *Estructura del mensaje ACL de FIPA* 10

3.1. *Descripción de Agentes en los Servidores Web* 17
3.2. *Protocolo de delegación AUML entre $Agent_A$ y $Agent_B$* 25
3.3. *Sintaxis de protocolo de delegación expresado en Attribute Global Types* 28

4.1. *Endpoints* distribuido mediante agentes 30
4.2. *AgentDefinition para leer cola mensajes ACL* 31
4.3. *NautiLOD* distribuido mediante agentes 33
4.4. *Contenido del mensaje con acciones* 34
4.5. *Autómata inicial del lenguaje NautiLOD* 36
4.6. *Interacción entre agentes distribuidos* 37
4.7. *Resultado obtenido en el Endpoint dbpedia.org* 38
4.8. *Comunicación mediante mensajes ACL* 39
4.9. *Resultado obtenido en el Endpoint geonames.org* 39
4.10. *Resultado entregado al Endpoint yago.org* 40
4.11. *Resultado final de NautiLod Distribuido* 40

5.1. *Arquitectura de Agent Server Platform* 44
5.2. *Secuencia de ejecución de la plataforma "Agent Server"* 46
5.3. *Definición de los valores y propiedades de AgentDefinition* 48
5.4. *Composición del mensaje ACL estipulado por FIPA* 50

6.1. *Enlaces a sitios con resultado de la Tesis.* 53

Capítulo 1

Introducción

Los avances tecnológicos de los últimos años y la creciente evolución de la Web ha promovido el desarrollo de nuevas propuestas tecnológicas basadas en la interacción entre diferentes entidades, todo esto enfocadas en un nuevo paradigma de computación donde las tareas requieren ser resueltas de manera distribuida.

Satisfacer las condiciones de este nuevo paradigma exige que la tecnología a utilizar cuente con características para la interacción y de alguna manera con un grado de autonomía, que se adapten, coordinen y se organicen entre ellas. Por su parte, la tecnología de Sistemas MultiAgente (SMA) resulta especialmente prometedora como soporte a este paradigma, pues una de las varias características principales de los agentes consiste en la cooperación y la sociabilidad, es decir, en su capacidad para comunicarse con otros agentes y cooperar entre sí, a través del uso de ontologías, lenguajes y protocolos de interacción comunes. El objetivo principal de tal SMA es lograr, a través de la cooperación entre sus miembros, los objetivos y las tareas que son difíciles de alcanzar para un solo agente [50].

Por otra parte, la delegación es un aspecto fundamental para la distribución y la transferencia de tareas dentro de una organización. En la interacción de Sistemas MultiAgentes, la delegación es uno de los componentes básicos para la coordinación, colaboración y organización entre agentes. De hecho, la mayoría de los Sistemas Distribuidos Inteligentes, sistemas de negociación y agentes de software cooperativos, se basan en la idea que la cooperación funciona a través de la asignación de una tarea por un determinado agente, ya sea individual o grupal, a otro agente a través de alguna solicitud, oferta, propuesta o anuncio. En las organizaciones complejas, ciertas tareas requieren agentes para coordinar con otros y delegar tareas si se quiere tener éxito. Por ejemplo, una tienda virtual para mantener su competitividad, debe delegar la entrega de sus productos a equipos competentes y fiables. Por lo tanto, la noción de la delegación está presente de manera explícita en los SMA.

Por otro lado, sabemos que la Web es el trabajo colectivo más grande del conocimiento nunca producida y está disponible públicamente como un espacio de información universal. La Web prácticamente nos ha cambiado nuestro percibir el día a día, permitiendo que nuestra comunicación sea cada vez más simple y rápida, hoy en día disponemos millones de recursos que son imposible de tener acceso a todos ellos ya sea por el formato o su estructura. Pero, la Web Semántica ayuda a resolver estos problemas permitiendo al usuario delegar algunas tareas a agentes de software [11].

Esta tesis busca dar una introducción a noción de la delegación en el entorno de la Web mediante agentes. Para ello, se considera como punto de partida el trabajo realizado por Castelfranchi & Falcone [20, 33], donde se presenta un enfoque sobre la teoría de la delegación basado en agentes con arquitectura BDI (Belief, Desire, Intentions) [71]. Sin embargo, en este trabajo de tesis se implementa con Sistema MultiAgentes de arquitectura reactivas, esto debido a la simplicidad de su razonamiento que posee. Así mismo se propone la elaboración de un lenguaje de especificación para la delegación de tareas en el entorno de la Web para que los agentes puedan interactuar entre si delegando tareas especificas.

A continuación se presentan los objetivos de la tesis como fueron propuestas al inicio de esta investigación y la metodología utilizada para llevar acabo la investigación propuesta.

1.1. Objetivo General

El objetivo central de esta investigación consiste en la elaboración de un lenguaje de especificación para la delegación de tareas en Servidores Web mediante agentes y su consecuente aplicación en los Endpoints SPARQL.

1.2. Objetivos específicos

- Elaborar un lenguaje de especificación para definir las acciones en la delegación de tareas con propiedades necesarias para su autonomía y bajo la restricción del protocolo de comunicaciones HTTP.
- Diseñar un Sistema MultiAgente de comportamiento flexible, autónomo y robusto, de tal forma que estos agentes puedan interactuar entre sí, delegando tareas específicas.
- Validar el trabajo propuesto en una infraestructura que tenga datos semánticos homogéneos como es Linked Data.

1.3. Metodología

- La primera parte de este trabajo consiste en el estudio del estado del arte de Sistema Multi Agente, la exploración de los lenguajes de especificación para la delegación que han sido propuestos y el protocolo de comunicación HTTP.
- La segunda etapa consiste en la elaboración de una especificación formal del lenguaje para la delegación de tareas basado en el protocolo de comunicaciones HTTP.
- En la tercera etapa se describe el caso de estudio en Linked Data, que permite demostrar cómo es factible la delegación de tareas en una infraestructura homogénea.
- En una cuarta etapa se describe el desarrollo de una plataforma de Sistema MultiAgente que es capaz de delegar tareas a través del protocolo de comunicaciones HTTP.
- Por último se realiza la validación en una infraestructura con datos homogéneos como es Linked Data. Para ello se trabajó con datos del Endpoint SPARQL. Se copió una cantidad significativa de datos de los principales Endpoints SPARQL (dbpedia.org, freebase.org, geonames.org). Estos fueron almacenados en cada uno de los Servidores Web que están separados geográficamente. Sobre ello se testeó las consultas y delegaciones.

1.4. Contribuciones

Las contribuciones más importantes de esta tesis se pueden resumir en:

- Desarrollo un lenguaje de especificación para la delegación de tareas en Servidores Web mediante agente y bajo la restricción del protocolo HTTP y formatos de datos RDF. Es decir, delegar tareas mediante los principales métodos POST, GET y PUT.
- Diseño e implementación de una plataforma donde los agentes puedan coexistir de manera persistente y enfocada en los principios REST. Esta cuenta con funcionalidades tales como: Sistema de comunicación entre agentes utilizando los métodos del protocolo HTTP que permite la delegación de tareas hacia otro agentes; soporte para la Web Semántica, es decir, la capacidad de interpretar el lenguaje NautiLOD; diseño de interfaz de usuario basado en la Web que permite la gestión de la plataforma de agentes.
- Implementación del lenguaje NautiLOD de manera distribuida y mediante Sistema Multi-Agente.

1.5. Estructura de este documento

El documento de esta tesis se estructura en ocho capítulos:

1. *Introducción:* (este mismo capítulo) Describe el problema a resolver, la metodología y la estructura de este documento.

2. *Antecedentes preliminares:* Se describe el contexto en el que se desarrolla esta tesis, tanto en los aspectos técnicos y científicos. Se describe el estado del arte de Agentes y Sistemas MultiAgentes, del proceso de la delegación y los lenguajes de especificaciones de tareas.

3. *Modelo:* Se presenta el modelo del lenguaje de especificación de tareas que permita a los agentes delegar tareas específicas, comunicarse e intercambiar mensajes o recursos entre sí bajo la restricción del protocolo HTTP.

4. *Casos de estudio:* Se plantea el caso concreto del Lenguaje NautiLOD para resolver de manera distribuida mediante agentes. NautiLOD es un lenguaje de expresión declarativo que está diseñado para especificar patrones de navegación en la red Linked Open Data.

5. *Plataforma Agent Server:* Se presenta el diseño y la implementación de la plataforma de Sistemas MultiAgente *Agent Server*. Una plataforma flexible y escalable para Sistema Multi Agentes basados en el ambiente de la Web, desarrollado bajo los principios de REST.

6. *Resultados y Discusión:* Se presenta los resultados de los diversos experimentos realizados de nuestro modelo en base a la plataforma *Agent Server*.

7. *Conclusiones y trabajo a futuro:* presenta las principales conclusiones del desarrollo de la tesis y trabajos futuros a desarrollar.

Capítulo 2

Antecedentes preliminares

En el presente capítulo se presenta una breve descripción de los Agentes y Sistema MultiAgente, así como algunos temas relacionados a agentes. En primer lugar, se señala los conceptos de agente, luego la caracterización y la descripción de algunas arquitecturas de agentes. Posteriormente, se presenta el concepto y aplicaciones de la noción de Sistema MultiAgente, los sistemas de comunicación y se describe algunas plataformas para el desarrollo de Sistema MultiAgente. En la sección 2.1.7 se da una introducción a la delegación de tareas en los Sistemas MultiAgente. Finalmente se describe los lenguajes de especificación de tareas.

2.1. Agentes

Según descripciones dadas por varios autores, los agentes son entidades conceptuales que perciben y actúan de una forma pro-activa o reactiva [80], en un entorno donde otros agentes existen e interactúan basados en una comunicación y representación del conocimiento común. Un aspecto de los agentes que es ampliamente mencionado en la literatura es la visión de un agente como una entidad interactiva que existe como parte de un entorno compartido con otros agentes.

Otros autores como Frankin [45] y Huhns [53] agregan a la definición anterior la propiedad de autonomía, expresando de esta forma la capacidad del agente de pertenecer a un sistema donde percibe y actúa sobre ese entorno a través del tiempo, persiguiendo los objetivos que tenga usando su propia agenda, para así, actuar sobre lo que percibirá en el futuro sin la intervención de otras entidades. Podemos definir como agentes racionales [72] a aquellos agentes que en cada posible secuencia de percepciones deberán emprender aquella acción que supuestamente maximice su medida de rendimiento, basándose en la evidencia aportadas por la secuencia de percepciones y en el conocimiento que el agente mantiene almacenado, comportándose así de forma "inteligente". Por tanto, existen muchas formas de definir a un agente, pero lo que está claro es que debe cumplir una serie de características que definen su comportamiento dentro del sistema. A continuación se muestra las

principales características que es posible identificar en el concepto de agente.

2.2. Características de los Agentes

Las características que deben de poseer los agentes, en mayor o menor grado, para resolver problemas particulares y que han sido descritas por autores tales como Frankin y Graesser [45] Nwana [66], Wooldridge y Jennings [80] o Bradshaw [14]:

- **Autonomía**: Un agente es autónomo si es capaz de actuar basándose en su experiencia, sin la intervención de un ser humano y/u otro agente. El agente es capaz de continuar actuando aunque el entorno cambie severamente. Por tanto, se puede decir que un agente es autónomo si está dirigido por sus objetivos, de una forma pro-activa y con un comportamiento propio.

- **Sociabilidad**: Un agente deben ser capas de interactuar con otros agentes, posiblemente tan complejos como los seres humanos, con miras a la satisfacción de sus objetivos.

- **Reactividad:** Esta característica se refiere a la capacidad de los agentes para reaccionar rápidamente frente a los cambios en el medio ambiente, ajustándose a ellos. El interés sobre la propiedad de reactividad condujo a una gran cantidad de investigación en el área [35]. Por ejemplo, se utilizaron ejemplos con las colonias de hormigas para demostrar que los agentes reactivos simples pueden mostrar trazas colectivas de comportamiento inteligente, incluso tareas de resolución consideradas compleja.

- **Racionalidad**: Es la propiedad que, para cada secuencia de percepciones, un agente debe emprender aquella acción que supuestamente maximice su medida de rendimiento, basándose en las evidencias aportadas por la secuencia de percepciones y en el conocimiento que el agente mantiene almacenado.

- **Pro-actividad**: Un agente es pro-activo cuando es capaz de controlar sus propios objetivos a pesar de los cambios en el entorno. Esta definición es complementaria a la reactividad. El comportamiento del agente es resultado de dos tipos de comportamiento: el receptivo y el descubrimiento. En un comportamiento receptivo, el agente es guiado por el entorno. El comportamiento de descubrimiento usa procesos internos del agente para obtener sus propios objetivos. El agente debe de tener un grado de comportamiento receptivo (propiedad reactiva) y un grado de comportamiento de descubrimiento (propiedad pro-activa).

- **Adaptabilidad**: Habilidad de aprender y mejorar, con la experiencia, lo que dispondría un agente a la hora de integrarse en el entorno.

- **Movilidad**: Capacidad del agente para viajar por redes de computadores visitando distintos nodos a fin de realizar las tareas necesarias para cumplir con sus objetivos de diseño. Una vez concluido su trabajo pueden regresar a su lugar de origen o eliminarse.

- **Veracidad**: Un agente no comunica intencionadamente información falsa.

- **Cooperación**: La cooperación es la capacidad de trabajar junto con otros agentes, para lograr un objetivo común, o completar una tarea común. El agente debe poseer habilidades sociales, con el fin de comunicarse con otros agentes.

- **Benevolencia**: Siempre y cuando no perjudique la resolución de objetivos propios, un agente debe estar dispuesto ayudar a otros agentes.

No está determinado el grado en el que se deben de presentar estas características en los agentes, aunque sí se puede hacer una taxonomía de agentes [45] dependiendo de la disponibilidad de estas propiedades. Lo que no cabe ninguna duda es que estas son las características que diferencian a un agente de un mero programa computacional.

2.3. Arquitecturas de Agentes

Existe un gran diversidad de tipos de agentes. Estos podrán tomar decisiones más racionales al disponer de tiempo para su deliberación o, por el contrario, el tiempo se convierte en un recurso crítico y es necesario disponer de una solución de forma más reactiva. En este sentido, no existe una arquitectura única que defina una metodología particular para construir agentes inteligentes y que defina los mecanismos que utiliza el agente para reaccionar a los estímulos, actuar, comunicarse, etc. La estructura concreta de la arquitectura depende de las tareas, el entorno donde éstas se desarrollen y la forma de actuar ante las necesidades del agente. Wooldridge y Jennings [79] proponen tres categorías de arquitectura:

- **Arquitecturas deliberativas** En este tipo de arquitectura se sigue el enfoque de la Inteligencia Artificial clásica [52], donde los agentes utilizan modelos de representación simbólica del conocimiento y las decisiones se toman a través del razonamiento lógico. Estos agentes parten de un estado inicial y son capaces de generar planes para alcanzar sus objetivos. Por tanto, el proceso deliberativo del agente se encarga de encadenar los pasos necesarios para pasar del estado inicial a un estado final que cumpla los requisitos que satisfagan el objetivo del agente. Podemos distinguir dos tipos de arquitecturas deliberativas: *Arquitecturas intencionales*, donde los agentes son capaces de razonar sobre sus propias creencias e intenciones. Estos agentes

están dotados de modelos de planificación capaces de generar planes a partir de las creencias e intenciones [55]; *Arquitecturas sociales*, donde los agentes deben poseer modelos explícitos de otros agentes y razonar sobre estos modelos [68]. Dentro de las arquitecturas intencionales podemos destacar como máximo exponente la arquitectura BDI (Belief, Desire, Intentions) donde la implementación de los agentes los dota de los estados mentales de Creencias, Deseos e Intenciones [71]. Las creencias representan información, describiendo así el medio ambiente. Los deseos son los objetivos que el agente debe lograr y las intenciones corresponden a un conjunto de acciones o tareas seleccionadas por el agente con el fin de alcanzar los objetivos deseados.

- **Arquitecturas reactivas** Estas arquitecturas se caracterizan por no tener como elemento central de razonamiento un modelo simbólico y por no utilizar razonamiento simbólico complejo [16, 23], tratándose de tomar una decisión en tiempo real. El conocimiento necesario por el agente para la consecución de los objetivos es obtenido mediante la observación e interacción con el entorno, necesitando para ello un conjunto de acciones muy básicas. La arquitectura reactiva más conocida es Arquitectura de Brooks [15], que predice la existencia de varios comportamientos independientes y un mecanismo para seleccionar la mejor acción posible en cada momento, lo que produce un comportamiento con jerarquías de subordinación.

- **Arquitecturas híbridas** Como su nombre indica, estas arquitecturas hacen uso de las características de ambas arquitecturas, las reactivas y las deliberativas. Las arquitecturas híbridas surgieron con el propósito de utilizar la capacidad de realizar acciones de una forma reactiva a los cambios que surgen en el entorno que nos aporta las arquitecturas reactivas, y por otro lado, el utilizar modelos simbólicos y la generación de planes para la consecución de objetivos que propone la arquitectura deliberativa.

2.4. Sistema MultiAgente

Cuando se tiene una tarea que es demasiado complejo para ser realizado por un solo agente se puede plantear el dividirla entre varios agentes, tomando cada agente una parte menos compleja y de más fácil resolución. Con esta idea surgen los Sistemas Multi-Agente [53], sistemas compuestos por varios agentes que interactúan, cooperan y coordinan entre ellos buscando satisfacer sus objetivos propios y los establecidos para el sistema.

Durfee [29] define un Sistema Multi-Agente (SMA) como una red de elementos no acoplados que resuelven problemas y que trabajan juntos para solucionar un problema que está por encima de

la capacidad de cada uno ellos por separado. Por otro lado, la investigación y la práctica científica en SMA, que en el pasado ha sido llamado Inteligencia Artificial Distribuida (DAI), se centra en el desarrollo de la computación, principios y modelos para construir, describir, aplicar y analizar los patrones de interacción y coordinación, tanto en sociedades grandes como pequeñas de agentes [58]. DAI se centra en problemas donde varios agentes realizan partes de una tarea y se comunican en un lenguaje de alto nivel. DAI se puede dividir en dos áreas principales de investigación [30]:

- **Resolución de problemas distribuida**. El problema en cuestión se divide en un número más pequeño de módulos o sub-problemas, que son resueltos por entidades independientes (agentes) que cooperan sólo en la carga de trabajo y en compartir el conocimiento y el intercambio de resultados.

- **Sistemas Multi-Agente**. El objetivo es coordinar una serie de agentes autónomos, mediante la coordinación de conocimientos, objetivos, competencias y planes con el fin de resolver colectivamente los problemas y realizar tareas.

Según Wesson [78], un SMA viene caracterizado de la siguiente manera:

- Un SMA está formado por un conjunto de agentes, cada uno de los cuales mantiene sus propias habilidades: adquisición de datos, comunicación, planificación y actuación.

- El SMA tiene una misión común. La misión puede descomponerse en diferentes tareas independientes, de forma que se puedan ejecutar en paralelo. El sistema multi-agente debe ser capaz de asignar a cada uno de sus componentes una o varias tareas concretas teniendo en cuenta cual es el objetivo común.

- Cada agente del sistema tiene un conocimiento limitado. Esta limitación puede ser tanto del conocimiento del entorno, como de la misión del grupo, como de las intenciones de los demás agentes a la hora de realizar sus propias tareas.

- Cada agente del sistema tiene cierta especialización para realizar determinadas tareas, en función de lo que conoce, la capacidad de proceso y la habilidad requerida.

2.5. Comunicación entre agentes

Con el fin de comunicar, existe la necesidad de desarrollar un lenguaje común, especificando la sintaxis, la semántica, el vocabulario, modelo de dominio de la pragmática y el discurso. A principios de los años noventa, se lograron dos acontecimientos principales en esta dirección [37]:

- KIF (Knowledge Interchange Format). KIF está destinado a representar el conocimiento en un dominio específico. Se basa en la lógica de primer orden, utilizando una notación de prefijo, por lo que es posible ser interpretado por los seres humanos y los programas de ordenador. En él se describen las propiedades de dominio y relaciones entre los objetos, que proporcionan los operadores booleanos lógicos, los cuantificadores, universal y existencial, y los tipos de datos más comunes. Ha sido desarrollado principalmente para expresar el contenido de los mensajes KQML [51].

- KQML (Knowledge and Query Manipulation Language). KQML es un lenguaje para la comunicación basada en mensajes entre agentes. En él se especifica toda la información necesaria para contenido de la comprensión del mensaje. Cada mensaje está compuesto de un *performativo* (tipo de mensaje), y un número de parámetros con valor respectivo. El crecimiento natural de este lenguaje, sin embargo, ha traído algunos problemas, como los problemas de compatibilidad hacia atrás, su origen por modificaciones en el número y tipos de *performativos* [57].


```
{
  "conversationId": "geonames-123581844",
  "sender": "agent1@dbpedias.cloudapp.net",
  "receiver": "agent2@geonames.cloudapp.net",
  "replyTo": "agent3@yagos.cloudapp.net",
  "content": "::putTo(agent3@yagos.cloudapp.net,
              ::exec(http://sws.geonames.org/3165322/ -p
               <http://www.w3.org/2000/01/rdf-schema#isDefinedBy>
              [ASK {?ctx <http://www.geonames.org/ontology#population>
              ?pop. FILTER (?pop >10000).}] -f files.rdf))",
  "language": "",
  "encoding": "123525235",
  "ontology": "2",
  "protocol": "",
  "replyWith": "",
  "inReplyTo": "",
  "replyBy": "",
  "performative": "REQUEST"
 },
```


Figura 2.1: *Estructura del mensaje ACL de FIPA*

A mediados y finales de los noventa, la Fundación para Agentes Físicos Inteligentes (FIPA) comenzó a desarrollar normas para SMA. Una de esas normas es el Lenguaje de Comunicación para Agente FIPA (ACL), un lenguaje similar al KQML, pero que contiene mucho menos *performativos* (sólo veinte, en comparación con los más de cuarenta especificados por KQML), y más adecuada a la negociación de procesos [43]. Con el fin de definir un vocabulario común y aceptado en todo el

mundo, con las mismas expresiones y significados, se utilizan normalmente ontologías, especificando no sólo taxonomía de clase, sino también conceptos, propiedades y relaciones.

Para entender mejor FIPA ACL, veamos un ejemplo representado en la Figura 2.1. La interpretación de este mensaje es como sigue: El emisor *agent*1@*dbpedias.cloudapp.net* requiere del *agent*2@*geonames.cloudapp.net* para el envío de un mensaje. Los atributos más relevantes de este mensaje son: *sender*, en este campo se especifica el agente que envía el mensaje; *receiver*, en este campo se especifica a quien se desea enviar el mensaje; *content*, especifica el contenido del mensaje; *language*, aquí se especifica el lenguaje en el cual está expresado el contenido; *ontology*, define los terminos usados en el mensaje; *performative*, en este campo se especifica los performativos estipulados por FIPA.

2.6. Plataformas Sistema MultiAgente

El desarrollo de Sistemas MultiAgente se ha incrementado en los últimos años, en los campos de la investigación y la industria. Desarrolladores de SMA necesitan un software que les ayude en el proceso de desarrollo. Plataformas Multiagente (MAP) proporcionan algunas herramientas que mejoran el desarrollo y la aplicación de los SMA. Debido a la gran cantidad de SMA existentes, la elección de la MAP mas adecuada para desarrollar un SMA convierte en una tarea difícil para los desarrolladores SMA.

Estas MAP's proporcionan soporte para el desarrollo de aplicaciones basadas en agente, es decir, mediante la abstracción de las características específicas para la implementación de la comunicación. Desde que aparecieron los estándares para la comunicación de los SMA, se han propuesto y desarrollado por la comunidad científica, varias plataformas implementando la comunicación del agente con el uso de estos estándares y liberando a los desarrolladores de tener que aplicarlos.

Estas MAP's por lo general cuentan con una serie de servicios, según lo especificado por FIPA [2]. Los servicios más básicos son el Servicio de Transporte del Mensaje (MTS) y el Servicio de Gestión de Agente (AMS). El MTS proporciona una infraestructura que garantiza que los mensajes enviados desde un agente a otro se entregan, independiente de la ubicación del agente receptor el cual puede estar en la misma plataforma o en otro. El AMS, también llamado el Servicio de Páginas Blancas, es un servicio de registro centralizado, la recopilación de información sobre todos los agentes presentes en el plataforma, que puede ser consultada por otros agentes. Otro servicio deseable es el Directorio Facilitador (DF), que, aunque no es obligatorio, está generalmente presente en la mayoría de plataformas. El servicio DF, también llamado servicio de Páginas Amarillas, es utilizado por los agentes para registrar los servicios que ofrecen a otros agentes y para consultar

por los servicios ofrecidos. Las características adicionales como la movilidad agente, balanceo de carga, la persistencia y otros son también algunas de las características deseables de la plataforma de agentes.

En los últimos años, muchos investigadores se han centrado en probar el rendimiento y principalmente la escabilidad de las plataformas Multi Agente. Una de las principales propiedades necesarias para este trabajo es la escalabilidad de las plataformas SMAs. Como se indica en [4, 8, 59], se ha desarrollado una gran cantidad de plataformas SMA a través de los años y solo unos cuantos de estos sistemas siguen siendo desarrollados activamente. Sin embargo, en la actualidad es difícil encontrar alguna plataforma SMA con enfoque al ambiente de la Web. Por ello dedicamos el capítulo 5 al desarrollo de una plataforma para la gestión de Sistema MultiAgente basado en la Web.

2.7. Delegación de tareas

En un estudio temprano, realizado por Castelfranchi & Falcone [20, 32], se presenta un enfoque sobre la teoría de la delegación basado en el modelo de agentes cognitivos BDI, es decir, agentes que tienen creencias, metas, intenciones, y planes [71]. En ese trabajo se introdujo una definición informal de delegación mediante el operador ***Delegate(A, B,*** τ***, d)***, donde ***A*** *es el agente que delega*, ***B*** *el agente contratador*, τ es la tarea que está formada por un posible plan y la meta que se quiere lograr, y ***d*** es el tiempo límite para ejecutar la tarea. El operador significa que ***A*** delega la tarea τ a ***B*** con un tiempo límite ***d***.

Posteriormente, Doherty & Jules [26] presentan un Framework de delegación enfocado en Vehículos aéreos no tripulados (UAV). En ese trabajo presentan la formalizacion de la teoría de delegación de Castelfranchi & Falcone basada en el formalismo KARO [77]. KARO es una combinación de lógica dinámica y epistémica/lógica doxástica. El trabajo toma como caso de estudio catástrofes y desastres naturales tales como el terremoto ocurrido en Sumatra, tsunami de India, Tailandia, Indonesia donde los UAVs realizan tareas de búsqueda de personas accidentadas de forma colaborativa con los humanos, delegando tareas entre sí. La implementación está basada en la arquitectura CORBA y aprovecha las funcionalidades de JADE [9]. JADE es el FrameWork de SMA más estable, ampliamente utilizado y es compatible con la arquitectura FIPA. La comunicación entre agentes se realiza mediante el lenguaje de comunicación ACL especificado por FIPA de la misma manera cada agente tiene asociado un número determinado de servicios (acciones).

Siendo la delegación un campo abierto para la investigación, existe una gran literatura desarrollada sobre los fundamentos teóricos de la delegación: La especificación de las formas y niveles de delegación [18, 65], aspectos organizativos de la delegación, análisis del impacto de la relación entre

los agentes en la delegación [12, 13, 18, 19], técnicas y estrategias para delegar [17, 76] y los tipos de tareas para delegar [60]. En la siguiente sección, examinamos una serie de investigaciones sobre estos conceptos.

2.7.1. Tipos de delegación

En la teoría de la delegación de Castelfranchi & Falcone [20] se plantean dos tipos de delegación basada en la especificación de tareas:

- *Delegación cerrada:* que significa que en una delegación la tarea está completamente especificada, y que el $Agente_A$ solo espera que el $Agente_B$ ejecute la acción delegada, es decir, tanto la tarea, plan y la meta deben ser respetados.

- *Delegación abierta:* la tarea no está completamente especificada, esto es, únicamente la meta tiene que ser respetada mientras que el plan puede ser elegido por el $Agente_B$, o el plan especificado puede contener acciones que necesiten una mayor elaboración por parte del el $Agente_B$.

En la delegación abierta, el $Agente_B$ tiene cierta libertad sobre la forma de realizar la tarea delegada y por tanto proporciona un grado mayor de flexibilidad para la planificación en un SMA, lo que permite una verdadera planificación distribuida.

2.7.2. Estrategias de delegación

La estrategia de delegación es un conjunto de reglas que ayuda a un agente a seleccionar otro agente para la delegación de la tarea. Falcone et al. [34] centraron su investigación en el uso de *el protocolo Contract Net* para comparar el rendimiento de la diversas estrategias de delegación. Dentro de ellas evaluaron: *La estrategia aleatoria*, donde el agente escoge al azar a quien va a delegar la tarea; *La estrategia estadística*, el agente se basa en modelos fiables y actuaciones de encuentros anteriores; *La estrategia cognitiva*, el agente considera las características específicas y el efectos del medio ambiente sobre el agente que va a delegar la tarea.

Por otro lado, Norman et al. [64] investigaron una serie de estrategias que los agentes pueden adoptar en la delegación de tareas en entornos dinámicos. En su enfoque, los autores identificaron cinco tipos estrategias de delegación, que incluyen:

- *Delegación simple*: En esta estrategia, un agente delega una tarea sin considerar ningún intento de interferir.

- *Delegación con supervisión*: En este caso, el agente de delegación ofrece un contrato para invocar al agente contratante a adoptar el nivel de esfuerzo deseado. En esta estrategia hay un costo asociado con el seguimiento, por lo que el agente de la delegación incurre en costos adicionales durante el uso de esta estrategia.

- *Delegación sin supervisión*: De la misma manera que la anterior, se hace un acuerdo de un contrato. Sin embargo, el agente de la delegación no tiene por qué pagar los costos de supervisión, pero pierde la capacidad de aprender acerca de los comportamientos de los agentes que delegan, en diferentes niveles de esfuerzo.

- *Abstenerse de delegar*: En esta estrategia, si ningún agente se encuentra en la situación que los beneficios superen los costos/riesgos de delegar entonces el agente se abstiene de delegación por completo.

Estas estrategias se basan en la suposición de que el agente delegante tiene suficiente información (por ejemplo, los costos de supervisión) para decidir qué estrategia adoptar. Además, supone que el nivel de rendimiento (o esfuerzo) de delegados son fácilmente observables. En situaciones en las que existen limitaciones subyacentes que informan el comportamiento de los agentes, este enfoque se puede utilizar para desentrañar las limitaciones e implementar estrategias apropiadas para manejarlos [31].

2.8. Lenguajes de Especificación de Tareas

Un lenguaje de especificación permite diseñar agentes por medio de sus estados mentales y capacidades efecto-sensoriales. El lenguaje de especificación impone cierta arquitectura interna al agente. Esta debe ser lo suficientemente flexible como para poder diseñar agentes con diferentes arquitecturas mentales (reactivos, de estado Interno o BDI). Una de las ventajas de usar un lenguaje de especificación para desarrollar los agentes es que proveen una forma de comunicación entre agentes que está completamente integrada a las capacidades cognitivas de estos, tratándola como un componente más en el razonamiento [47].

En el estudio realizado por Gottifredi & Garcia [48], se presentan condiciones deseables que deben cumplir un lenguaje de especificación para agentes: Ser suficientemente flexible como para poder diseñar agentes con diferentes arquitecturas mentales; proveer reglas para modelar la comunicación; dar soporte para que el agente pueda hacer planificación; y dar soporte para el manejo de la base de conocimiento.

Otro estudio, realizado por Doherty et al. [25], influenciado por la teoría de delegación de

Castelfranchi & Falcone, presenta un lenguaje de especificación de tareas con una estructura de datos abstracta denominada *Task Specification Tree (TST)*, que ayuda a representar la delegación de tareas y que cuenta con propiedades requeridas para solucionar problemas de iniciativas mixta y con autonomía ajustable para entornos distribuidos. Este lenguaje de especificación está enfocado en la arquitectura de agentes BDI e implementado para Vehículos Aéreos no Tripulados (AUVs). Por tanto, el lenguaje de especificación de tareas que se propone en esta tesis será influenciado por esta propuesta.

Duarte et al. [27,28] presenta un lenguaje de especificación, denominado *Common Control Language (CCL)*, diseñado para establecer un estándar para el intercambio de información y delegación de tareas entre los agentes, y que fue utilizado para especificar tareas sencillas a un grupo de Vehículos Aéreos no Tripulados.

Simmons & Apfelbaum [75] presentan otro lenguaje de descripción de tareas denominado *Task Description Language (TDL)* que simplifica el desarrollo de programas de control de robots móviles mediante la inclusión de soporte sintáctico explícito para las capacidades de control a nivel de tarea. TDL está diseñado para apoyar directamente en la descomposición de tareas, sincronización detallada de sub tareas, seguimiento de la ejecución y el manejo de excepciones. Las tareas se especifican en forma de estructura de datos de árbol, cada tarea tiene parámetros que pueden ser una meta o simplemente un comando, donde el comando es similar a una acción y esta acción puede contener código condicional o recursivo.

Capítulo 3

Especificación Formal

En este capítulo, se presenta el análisis completo de nuestro modelo, que permite a los agentes delegar tareas hacia otros agentes utilizando el protocolo HTTP. En primer lugar, se introduce los componentes necesarios y se describe las principales notaciones que serán utilizados en los siguientes capítulos. Seguidamente, formalizamos la noción de la delegación en el entorno de la Web mediante agentes y se discute con más detalle cómo se puede modelar la noción de delegación con los Sistema MultiAgente. Por último, se formula el lenguaje de especificación y el protocolo de interacción que regula el comportamiento de los agentes.

3.1. Modelo abstracto de la delegación en la Web

En esta sección se formula el mecanismo de delegación para identificar las principales características que puedan ocurrir durante la delegación en el entorno de la Web mediante el Sistema MultiAgente. Para ello, y a modo de motivación mediante la Figura 3.1 se representa a un grupo organizado de Servidores Web $Server_A$, $Server_B$, $Server_C$, $Server_D$, $Server_E$ y en cada de ellos se instala los agentes $Agent_A$, $Agent_B$, $Agent_C$, $Agent_D$ y $Agent_E$ respectivamente. Lo anterior describe un posible escenario de interacción de los agentes; el $Agent_A$ delega una tarea a $Agent_B$ y $Agent_C$, quienes deben de informar de una tarea realizada al $Agent_E$, quien centralizará y procesará el resultado recibido de los $Agent_B$ y $Agent_C$, para finalmente entregar el resultado a $Agent_D$ quien a su vez puede entregar el resultado a $Agent_A$ o alternativamente notificar mediante algún tipo sistema de mensajería (email, redes sociales) al usuario que generó la tarea.

Un requisito previo para el buen funcionamiento del escenario descrito sería la existencia de una infraestructura de software de Sistema Multi Agente de manera que los agentes que lo conforman puedan interactuar entre sí de manera colaborativa. Por lo menos, se requiere una plataforma basada completamente en el entorno de la Web que permita una interacción de manera robusta, segura y confiable. En lo que respecta a las plataforma individuales, estas podrían requerir diferentes

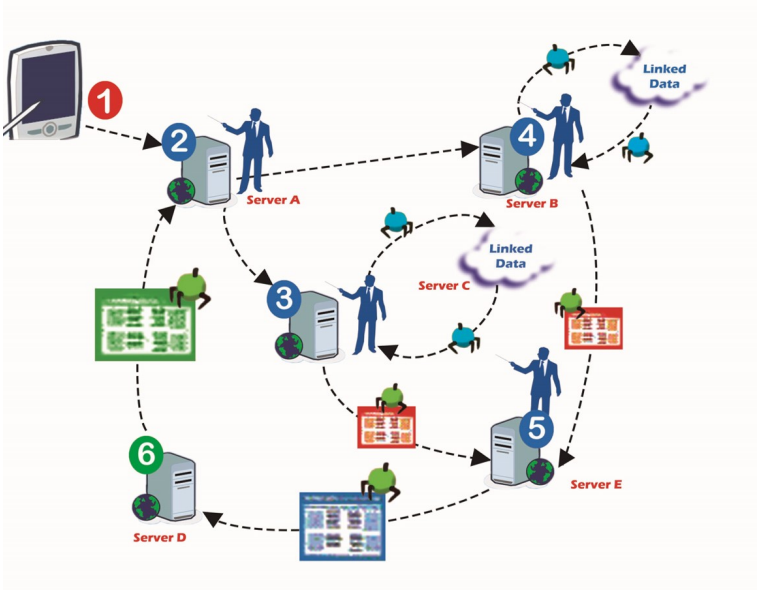

Figura 3.1: *Descripción de Agentes en los Servidores Web*

capacidades que no necesariamente podrían ser compartidas por cada plataforma individual, si no por todos los agentes que conforma el Sistema MultiAgente. Estas deberían incluirían por lo menos las siguientes capacidades:

- La capacidad de procesar la tarea delegada.
- La capacidad de subdelegar tareas hacia otros agentes.
- La capacidad de combinar los resultados recibidos.
- La capacidad de razonar sobre los resultados obtenidos y determinar si el resultado aún es necesario delegar hacia otro agente o se termina el proceso.
- La capacidad de poner los resultados en el lugar que se le indique. Esta puede ser directamente al agente solicitante o a un tercero.
- La capacidad de notificar el estado de la tarea al agente que delegó la tarea o directamente al correo electrónico del usuario.

Como podemos observar el conjunto de capacidades es bastante ambicioso. Estas capacidades tendrían que necesariamente ser resueltas de manera cooperativa entre los agentes, lo que implica tener un alto nivel de infraestructura de comunicación entre plataformas.

A continuación se presenta los componentes necesarios para el desarrollo de la especificación formal.

3.1.1. Agentes

Tal como se discutió en el capítulo 2, un agente es una entidad autónoma capaz de realizar alguna actividad. Cada agente en un dominio debe ser identificado. Denotamos al conjunto de agentes como un conjunto finito $Agent = \{A, B, ...\}$, asumiendo que hay por lo menos 2 agentes en el dominio. Los agentes tienen implementada diferentes acciones que permiten desarrollar sus objetivos y percepciones.

3.1.2. Acciones

Definimos una acción como una pieza de identificación de una actividad, es decir, las capacidades y habilidades. Asumimos que todos los agentes de un dominio vienen dotados de una serie de acciones. Para nuestro modelo una acción se utiliza de manera general. Denotaremos por $\mathcal{A} = \{\alpha, \beta, \dots\}$ el conjunto de todas las acciones.

3.1.3. Tareas

En el enfoque de la delegación de Castelfranchi & Falcone [20] y Doherty & Meyer [26] una tarea es representada como una tupla $(\alpha, \phi, cons)$, que consiste de una composición de acciones α, un objetivo ϕ y un conjunto de restricciones $cons$ que está directamente asociado a α. Para nuestro modelo, $\tau = (\alpha_1, \alpha_2, \dots)$ representa una tarea, que es una lista de acciones. Por otro lado, $T = \{\tau_1, \tau_2, \dots\}$ denotará el conjunto de las tareas. El cumplimiento de cada tarea requiere una serie de acciones a realizar. En otras palabras, se dice que una tarea que se cumple sólo después de las acciones correspondientes se ha realizado con éxito.

3.2. Definición de acciones

A continuación se introduce las principales acciones, básicas y opcionales, que deben tener los agentes que conforman el Sistema MultiAgente que son parte del proceso de la delegación. Los agentes deberían tener implementada, por lo menos, estas acciones que describiremos de manera general:

- **Exec()**, esta acción es considerada como básica. El agente que tiene esta acción implementada tiene la capacidad de ejecutar una determinada expresión (ejemplo, una expresión SPARQL). La acción es representada de la siguiente manera:

$$\mathbf{Exec}(Expr_i, Mdata_i)$$

Donde $Expr_i$ representa una determinada expresión, que puede ser representada por una consulta SPARQL, y por otro lado $Mdata_i$ representa la Meta Data que es requerida para la ejecución de la expresión $Expr_i$.

- **PutTo()**, esta acción es considerada como básica. El agente que tiene esta acción implementada tiene la capacidad de entregar el resultado obtenido, después de haber ejecutado la acción **Exec()**, a un determinado agente que es especificado inicialmente cuando se inicia el proceso de la delegación. Este agente puede ser quien inició la tarea τ_i o un tercero que se encuentra en otra plataforma distinta pero dentro del mismo dominio. La acción es representada de la siguiente manera:

$$\textbf{PutTo}(Agents, R_i)$$

Donde *Agents* representa al agente que recibe el resultado y R_i representa el resultado obtenido después de haber ejecutado la acción $\textbf{Exec}(Expr_i, Mdata_i)$.

- **Join()**, esta acción es considerada como básica. El agente que tiene esta acción implementada tiene la capacidad de unir los resultados recibidos y convertirla en una sola. La composición de esta acción es representada de la siguiente manera:

$$\textbf{Join}(R_i, ..., R_n)$$

Donde $R_i, ..., R_n$ representan los resultados obtenidos después de haber ejecutado las acciones $\textbf{Exec}(Expr_i, Mdata_i), ..., \textbf{Exec}(Expr_n, Mdata_n)$ respectivamente.

- **Result()**, Esta acción es considerada como básica. Esta acción es similar **PutTo()**, la diferencia principal está en que solo se encarga de informar sobre el estado de la tarea realizada al agente que delegó la tarea, es decir, si la acción se ejecutó correctamente o presentó algún problema. La composición de la acción es representada de la siguiente manera:

$$\textbf{Result}(Agents, R_i)$$

Donde *Agents* representa al agente que recibe el resultado de la ejecución de la tarea y R_i representa el resultado obtenido después de haber ejecutado la tarea τ.

- **SendMessage()**, Esta acción es considerada como opcional y es muy similar a **PutTo()** con la diferencia que el agente que realiza esta acción puede tener la capacidad de notificar mediante algún tipo de sistema de mensajería (Redes Sociales, correo electrónico y otros) el

estado y el resultado de la tarea realizada. La composición de la acción está compuesta de la siguiente manera:

$$\textbf{SendMessage}(\texttt{x@xyz}, R_i)$$

Donde R_i representa el resultado obtenido después de haber ejecutado la acción $\textbf{Exec}(Expr_i, Mdata_i)$ y `x@xyz` representa la cuenta o correo electrónico del usuario a quien se le notificará el resultado.

Cada una de estas acciones son completamente parametrizables y el grado de asignación de estos parámetros también reflejará el grado de delegación. Mientras menos parámetros se proporcionen el grado de delegación será más abierto.

3.3. Lenguaje y protocolo de especificación

En esta sección, se define el lenguaje y la especificación del protocolo de interacción entre los agentes participantes en el proceso de la delegación y de manera complementaria, ya que no es parte de los objetivos originales de las tesis, nos concentramos en formalizar utilizando el formalismo *Atribute Global Types* presentado por Mascardi & Ancona [61] y Ancona et al. [6].

3.3.1. Lenguaje

En el proceso de la delegación se requieren agentes que puedan especificar argumentos y comunicarse de manera estructurada. Estos argumentos pueden ser sobre la naturaleza de búsqueda de información, oferta o contra oferta, propuesta o simplemente expresada en desafíos que el agente debe lograr. Se utilizan los *performativos* definidos en la *Teoría de Actos del habla* (en inglés *speech acts theory*) presentada por Austin [7], que son frases que no solamente describen una realidad dada, sino que también cómo estas transforman la realidad. En el Lenguaje de Comunicación para Agentes (ACL) establecido por FIPA [44] o KQML [37] los *performativos* están denotados como *Actos Comunicativos* del mensaje ACL expresado como pregunta, recomendación o llamado.

Para nuestro propósito, nos enfocamos en el conjunto de *performativos* definido por FIPA, descrito en la tabla 7.1 del Apéndice 7, que es el más apropiado para la infraestructura de comunicación y su aplicación al protocolo de comunicaciones HTTP. Los *performativos* están definidos con el enfoque a los métodos GET POST, PUT y DELETE, lo que significa que cuando se utiliza un determinado *performativo* se utilizará su equivalencia con el método del protocolo de comunicaciones HTTP. (Por ejemplo, si se requiere usar el *performativo* INFORM esto se representa utilizado el método POST que en nuestro modelo permite entregar información a un determinado Servidor).

Por otro lado, la comunicación de los agentes utilizando los métodos, GET, POST, PUT y DELETE del protocolo de comunicaciones HTTP, ha sido estudiando recientemente por Abdelkader y Bergeret [10, 49]. Allí se presenta REST-Agent, un framework basado en la combinación de los conceptos de SMA y los principios de *Representational State Transfer* (REST) definido por Roy Fielding [36]. Así mismo, Althagafi [5] propone un modelo de agentes que siguen los principios REST en un esfuerzo para hacer frente a los desafíos de la escalabilidad, especificando las expresiones de los agentes mediante *Actos Comunicativos* de la arquitectura FIPA y los métodos GET, POST y PUT del protocolo de comunicaciones HTTP.

Para nuestro propósito, se adopta una notación similar a la de Althagafi para especificar las expresiones de los agentes, utilizando los *Actos Comunicativos* de la arquitectura FIPA y los métodos GET, POST y PUT del protocolo de comunicaciones HTTP, que es presentado en la tabla 7.1 del Apéndice 7. Sin embargo, para nuestro modelo solo se considera los *performativos* $QUERY\_IF$, $REQUEST$, $INFORM$, $NOT\_UNDERSTOOD$, $FAILURE$ y $REFUSE$. La elección de estos *performativos* se debe a la simplicidad de su implementación y también que en nuestro modelo solo se implementa los protocolos de interacción *FIPA Request Interaction Protocol* y *FIPA Query Interaction Protocol*, los cuales son definidos en la sección 3.3.5 de este capítulo.

3.3.2. Delegación como un acto comunicativo

La propuesta central de esta tesis es la delegación en el entorno de la Web. En consecuencia, se considera como punto de partida la noción de delegación como *Acto Comunicativo* presentado por Doherty et al. [24], el mismo que es formalizado bajo el enfoque de KARO [77] notación que es presentada como una fuerte delegación y que está enfocada a agentes con arquitectura DBI. Es decir, agentes que tienen creencias, metas, intenciones, y planes [71]. De igual forma la notación presenta pre y post condiciones que los agentes deben de cumplir.

En nuestro enfoque usamos una versión simplificada y está orientado a agentes completamente **reactivos**, es decir, agentes con razonamientos simples y bajo la restricción del protocolo de comunicaciones HTTP. Para ello, se define como *Acto Comunicativo* de la arquitectura FIPA, la siguiente notación:

$$\textbf{S-Delegate}(Agent_A, Agent_B, \text{performative}(\tau, Mdata)), donde\, \tau = (\alpha_1, \alpha_2, \alpha_3, \dots).$$

Lo que significa que el agente $Agent_A$ delega la tarea τ al $Agent_B$, utilizando los *performativos* de la arquitectura FIPA especificados en el tabla 7.1 del Apéndice 7. Para ello, se debe considerar necesariamente las siguientes condiciones:

- Todos los agentes tienen las mismas capacidades, es decir, tienen implementada las mismas acciones.
- Todos los agentes exhiben la lista de sus acciones implementadas.
- Los agentes pertenecientes al mismo dominio saben de la existencia de otros agentes y en consecuencia conocen la ubicación.
- Al pertenecer al mismo dominio todos los agentes tienen los permisos necesarios para realizar una acción.

La especificación de la delegación está enfocada al tipo de delegación abierta, lo que nos da mayor grado de flexibilidad y nos permite realizar una planificación para un trabajo distribuido.

3.3.3. Sintaxis

La sintaxis del lenguaje de especificación tiene el siguiente BNF:

$$\langle \text{SPEC} \rangle ::= \textbf{DELEGATE}(\langle \text{AGENT} \rangle, \langle \text{AGENT} \rangle, \langle \text{PER}(\langle \text{TASK} \rangle, \langle \text{METADATA} \rangle)))$$

$$\langle \text{TASK} \rangle ::= \langle \text{ACTION} \rangle \,|\, \langle \text{ACTION} \rangle^*$$

$$\langle \text{EXEC} \rangle ::= \langle \textbf{Exec}(\langle \text{EXPR} \rangle, \langle \text{METADATA} \rangle)$$

$$\begin{aligned} \langle \text{ACTION} \rangle ::= {} & \langle \text{EXEC} \rangle | \\ & \langle \textbf{Result}(\langle \text{AGENT} \rangle, \langle \text{EXEC} \rangle)\rangle | \\ & \langle \textbf{PutTo}(\langle \text{AGENT} \rangle, \langle \text{EXEC} \rangle)\rangle | \\ & \langle \textbf{Join}(\langle \text{EXEC} \rangle, \langle \text{EXEC} \rangle)\rangle | \\ & \langle \textbf{SendMessage}(\langle \text{xyz@com} \rangle, \langle \text{EXEC} \rangle)\rangle \end{aligned}$$

$$\langle \text{METADATA} \rangle ::= \langle \text{CONS} \rangle | \langle \text{ARG} \rangle$$

$$\langle \text{ARG} \rangle ::= \langle \text{VAR} \rangle | \langle \text{VALUE} \rangle$$

$$\langle \text{CONS} \rangle ::= \langle \text{constraint} \rangle$$

$$\langle \text{VAR} \rangle ::= \langle \text{variable name} \rangle$$

$$\langle \text{VALUE} \rangle ::= \langle \text{value} \rangle$$

$$\langle \text{AGENT} \rangle ::= \langle \text{Agent name} \rangle$$

$$\langle \text{PER} \rangle ::= \langle \text{Performative} \rangle$$

$$\langle \text{EXPR} \rangle ::= \langle \text{Expression} \rangle$$

Donde: ⟨Expression⟩ representa a una determinada expresión (Por ejemplo, una expresión NautiLOD, una expresión SQL) ; ⟨Performative⟩, representa a lista de *Performativos* especificadas en

la arquitectura FIPA y *xyz@com* representa la cuenta o correo electrónico del usuario.

3.3.4. Semántica

Durante el proceso de delegación, el lenguaje de especificación, está bien proporcionado para lograr un conjunto específico de tareas. Para que el proceso de delegación tenga éxito cada agente tiene implementada las mismas capacidades.

Para definir la semántica formal se parte desde el *Acto Comunicativo S-Delegate()* descrito en la sección 3.3.2, que se utiliza para definir que un $Agent_X$ tiene la capacidad y los recursos para lograr la tarea. Una condición importante para la aplicación exitosa del *Acto Comunicativo* es que el agente $Agent_A$ conozca la capacidad del agente $Agent_B$ para lograr la tarea. A continuación se provee una descripción detallada de la semántica. Para ello, la estructura básica del lenguaje de especificación es:

$$\langle \text{SPEC} \rangle ::= \textbf{DELEGATE}(\langle \text{AGENT} \rangle, \langle \text{AGENT} \rangle, \langle \text{PER}(\langle \text{TASK} \rangle, \langle \text{METADATA} \rangle) \rangle)$$

Donde **DELEGATE** denota el operador del proceso de la delegación, $\langle AGENT \rangle$ denota a los agentes delegante y contratante, $\langle PER \rangle$ denota el *performativo* estipulado en la arquitectura FIPA que es utilizado para el envío del mensaje tipo ACL, $\langle TASK \rangle$ denota la tarea a delegar y $\langle METADATA \rangle$ denota parámetros adicionales y limitaciones asociadas a la tarea. Una tarea $\langle TASK \rangle$ puede estar compuesta por la ejecución de una o varias acciones $\langle ACTION \rangle$. Dentro de la lista de acciones que se dispone, la acción $\langle EXEC \rangle$ que es la más importante ya que es el encargado de resolver la tarea especificada.

Con el fin de captar el proceso sistemáticamente, se extiende la semántica proporcionando el significado en términos de *Acto Comunicativo*:

$$\textbf{S-Delegate}(Agent_A, Agent_B, \text{performative}(\tau, Mdata)),$$

Que representa que cualquier agente $Agent_A$ puede lograr una tarea o puede encontrar una agente $Agent_B$ a la que se puede delegar la tarea, mediante los *performativos* definidos por FIPA.

3.3.5. Especificación de Protocolos

Para asegurar el comportamiento real de SMA, es de suma importancia especificar el protocolo de interacción de los participantes a fin de garantizar la interoperabilidad y seguridad de los participantes. Por ello, en esta sección se presenta la especificación del protocolo de interacción de los participantes en el proceso de la delegación.

Protocolo de delegación, En la figura 3.2, expresado en el formalismo AUML [67], se presenta el protocolo de delegación. Este protocolo se basa en la combinación entre *FIPA Request Interaction Protocol* [81] y *FIPA Query Interaction Protocol* [73]. El protocolo describe la interacción entre $Agent_A$ y $Agent_B$ y se desarrolla en las siguientes dos fases:

En la **fase 1**, $Agent_A$ verifica la lista de acciones que están implementadas en $Agent_B$. Para ello, se utiliza los *Performativos* $QUERY\_IF$ para consultar las acciones a desarrollar e $INFORM$ para recibir el resultado de la consulta. A continuación, se describe el uso de los *Performativos* implementados en esta primera fase:

- **QUERY_IF**$(\tau, Mdata)$, $Agent_A$ envía la tarea τ al $Agent_B$ mediante un mensaje de tipo $msg(Agent_A, Agent_B, QUERY\_IF(\tau, Mdata))$, donde τ representa la lista de acciones a verificar y $Mdata$ son los metadatos que se puede especificar para la ejecución de la tarea.

- **INFORM(Result())**, si el $Agent_B$ cumple con la lista de acciones solicitadas, este responde con un mensaje de tipo $msg(Agent_B, Agent_A, INFORM(Result(R_i), \tau))$, donde R_i contiene el resultado de la lista de acciones implementadas en $Agent_B$. Este resultado se utiliza en la Fase 2.

En la **fase 2**, si el resultado recibido satisface la lista de acciones a desarrollar, el $Agent_A$ cumple el rol del agente delegante que delega la tarea τ y el $Agent_B$ cumple el rol de agente contratante. Para ello, se utiliza los *Performativos* $REQUEST$ y $INFORM$. Mediante $REQUEST$ se envía la tarea τ al agente contratante $Agent_B$ quien se ve comprometido a ejecutar la tarea. Una vez que acaba de ejecutarla envía un mensaje de conclusión mediante $INFORM$ a $Agent_A$, con el fin de que el agente delegante no se quede esperando indefinidamente. De manera más detallada se describe a continuación el uso de los *Performativos* para la interacción de los agentes $Agent_A$ y $Agent_B$:

- **REQUEST**$(\tau, Mdata)$, el agente delegante $Agent_A$ selecciona al agente contratante $Agent_B$, quien se compromete a ejecutar la tarea τ. Este envía la tarea τ mediante un mensaje de tipo $msg(Agent_A, Agent_B, REQUEST(\tau), Mdata)$, donde τ representa la tarea a realizar, en este caso $\tau = PutTo(Exec(Expr, Mdata))$, y $Mdata$ representa a los argumentos y limitaciones adicionales.

- **INFORM**$(Result(R_i))$, cuando el agente contratante $Agent_B$ realiza la tarea, en este caso, $\tau = PutTo(Exec(Expr, Mdata))$, y de manera que el agente delegante $Agent_A$ no se quede esperando indefinidamente, el $Agent_B$ envía una conclusión sobre la tarea realizada mediante un mensaje de tipo $msg(Agent_B, Agent_A, INFORM(Result(R_i)), cid(Cid)))$, donde $Result$

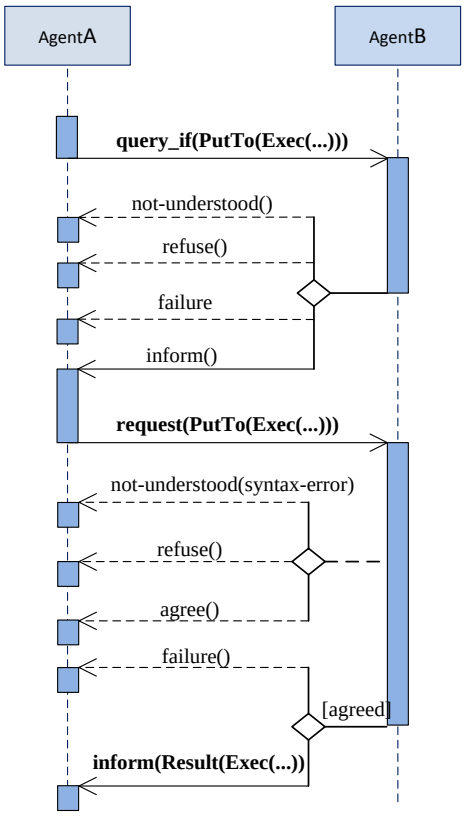

Figura 3.2: *Protocolo de delegación AUML entre $Agent_A$ y $Agent_B$*

es la acción básica implementada por los agentes, R_i es el resultado obtenido y Cid es el identificador del mensaje.

Cuando el agente $Agent_A$ envía una tarea τ al agente $Agent_B$, el agente $Agent_A$ envía un mensaje ACL de tipo $REQUEST(Exec())$ con esto se inicia una conversación entre ellos y en consecuencia se genera un identificador de la conversación por $Agent_A$ para realizar el seguimiento de la misma. Si el mismo agente $Agent_A$ envía un mensaje al agente $Agent_B$ la conversación que se empezó se considera como nueva y se generar una nueva identificación. Este identificador es almacenado como metadatos.

Este protocolo se repite cuando el agente $Agent_B$ requiere delegar una tarea a un tercer agente $Agent_X$, aplicando el mismo razonamiento en cada interacción que pueda ocurrir.

3.3.6. Protocolo de delegación con *Attribute Global Types*

Atribute Global Types, es un formalismo que permite verificar que los agentes interactue, negocien y discutan sobre las tareas asignadas de forma correcta. *Atribute Global Types* representa el estado de un protocolo de interacción en el que son posibles varios pasos de transición a otros

estados (es decir, a otros Atribute Global Types) con la resultante de una acción de envío. La sintaxis se basa en los siguientes bloques de componentes: *Acción de envío*, es un evento comunicativo que tiene lugar entre dos agentes, y que consiste en el emisor y el receptor del mensaje que consiste en que el performativo es expresado en algún lenguaje de comunicación de agentes como FIPA-ACL o KQML y el contenido real del mensaje expresado en algún lenguaje de contenido compartido entre agentes; *Tipos de acción de envío*, permite modelar el mensaje esperado en un determinado punto de la conversación. Un tipo de acción de envío α es un predicado en las acciones de envío; *Productores y consumidores*, con el fin de modelar las restricciones a través de diferentes ramas, se introduce dos tipos de acción de envío llamados productores y consumidores. En cada aparición de un tipo de acción de envío que se envía el productor debe corresponder a una nueva acción de envío. En contraste, la acción de envío del consumidor debe corresponder a la misma acción de envío especificado por un cierto productor; *Tipos globales limitados*, un tipo global limitado τ representa un conjunto infinito de posibles secuencias de acciones de envío, y es definido con los siguientes tipos de constructores:

- λ *(secuencia vacía)*, que representa el conjunto singleton $\{\epsilon\}$ que contiene la secuencia vacía de ϵ.
- $\alpha^n : \tau$*(seq-prod)* que representa el conjunto de todas las secuencias cuyo primer elemento es una acción de envío a vacío coincidente con tipo $\alpha(a \in \alpha)$, y la parte restante es una secuencia en el conjunto representado por τ. El exponente n especifica el número n de consumidores correspondientes que coinciden con el mismo tipo acción de envío α; por lo tanto, n es el número requerido de veces que $a \in \alpha$ tiene que ser consumido para permitir una transición marcada por a.
- $\alpha : \tau$ *(seq-cons)*, representa un consumidor de una acción de envió a coincidente con tipo $\alpha(a \in \alpha)$ y seguido por cualquier secuencia en el conjunto representado por τ.
- $\tau_1 + \tau_2$ *(elección)*, que representa la unión de las secuencias de τ_1 y τ_2.
- $\tau_1|\tau_2$ *(fork)*, que representa el conjunto obtenido por resolver las secuencias en τ_1 con las secuencias τ_2.
- $\tau_1.\tau_2$*(concat)*, que representa el conjunto de secuencias obtenidas mediante la concatenación de las secuencias de τ_1 con los de τ_2

Para diseñar y desarrollar *Attribute Global Types*, necesitamos saber tanto el flujo de información del protocolo y (opcionalmente) las limitaciones que los agentes deben respetar [62]. A continuación

podemos enumerar algunas limitaciones:

- *Limitaciones "local" en mensajes*, cada mensaje debe tener el tipo correcto, el mensaje de solicitud enviado por el agente delegante $Agent_A$ a un agente contratante $Agent_B$, los argumentos de la acciones a realizar deben ser extractamente los requeridos.

- *Limitaciones "horizontales" en consultas para delegar una tarea*, Cuando el agente delegante $Agent_A$ se contacta con el agente contratante $Agent_B$ para enviar una solicitud de delegación, los argumentos de estos mensajes deben formar un camino coherente. Los argumentos en el mensaje del agente $Agent_A$ deben de ser los mismos que el agente receptor $Agent_B$ y el identificador de la conversación debe ser la misma.

- *Limitaciones "verticales" en conversaciones entre agentes*, si el agente contratante $Agent_B$ no puede realizar la tarea, debe informar esta situación al agente $Agent_A$ en un intervalo de tiempo.

A continuación se presenta la formalización del protocolo de interacción descrito en la sección anterior. La interacción entre $Agent_A$ y $Agent_B$ queda representado por el siguiente *Attribute Global Types*:

$$\begin{aligned} DELEGATE =& (DQUERY : (DFAILURE : \lambda + DNTUNDERST : \lambda + DREFUSE : \lambda + \\ & (DINFORM : (DREQUEST : ((RINFORM : \lambda | RPUTTO : \lambda) + DFAILURE : \lambda + \\ & DNTUNDERST : \lambda + DREFUSE : \lambda))))) \end{aligned}$$

El agente delegante $Agent_A$ primero emite una **query_if**(DQUERY) para verificar las acciones implementadas en el agente contratante $Agent_B$ del cual se puede dar a lugar cuatro situaciones diferentes: o bien el agente contratante no tiene las acciones implementadas (DFAILURE), o no entiende el mensaje recibido (DNTUNDERST), o simplemente decide rechazar la tarea (DREFUSE), o por el contrario si tiene implementada las acciones y responde con un **inform**(DINFORM) esta situación. Una vez que al agente delegante $Agent_A$ recibe una respuesta positiva mediante **inform**(DINFORM) este emite una **request**(DREQUEST) y de igual forma puede ocurrir las mismas situaciones que la anterior adicionándose un **inform**(RINFORM) para recibir el estado del resultado y **request**(RPUTTO) que permite entregar el resultado a un tercer agente o directamente al correo electrónico del usuario solicitante. En lo que respecta al protocolo para **request**(RPUTTO) se utiliza el mismo razonamiento donde el agente que entrega el resultado cumple el rol de agente delegante y el que recibe el resultado cumple el rol de agente contratante. Los datos del tercer agente y datos de usuario son especificados en la $Mdata$.

Las ecuaciones que definen los subtipos se dan a continuación:

$$DQUERY = msg(Agent_A, Agent_B, query\_if(Exec(Expr_i, Mdata)), cid(Cid))^1$$

$$DFAILURE = msg(Agent_A, Agent_B, failure(Exec(Expr_i, Mdata)), cid(Cid))^1$$

$$DNTUNDERST = msg(Agent_B, Agent_A, notUnderstood(Exec(Expr_i, Mdata)), cid(Cid))^1$$

$$DREFUSE = msg(Agent_A, Agent_B, refuse(Exec(Expr_i, Mdata)), cid(Cid))^1$$

$$DINFORM = msg(Agent_B, Agent_A, inform(Result(Exec(Expr_i, Mdata))), cid(Cid))^1$$

$$DREQUEST = msg(Agent_A, Agent_B, request(Exec(Expr_i, Mdata)), cid(Cid))^1$$

$$RINFORM = msg(Agent_B, Agent_A, inform(Result(Exec(Expr_i, Mdata))), cid(Cid))^1$$

$$RPUTTO = msg(Agent_B, Agent_A, request(putTo(Result(Exec(Expr_i, Mdata))), cid(Cid))^1$$

En cada tipo de mensajes que aparece en las ecuaciones, los mensajes tienen el exponente 1 que significa que es el número de consumidores que están presentes en el protocolo.

La Figura 3.3 representa la secuencia de la interacción de los agentes $Agent_A$ y $Agent_B$ para envío de las tareas por medio de mensajes ACL y a través del uso de los *performativos*.

$$\begin{aligned}
G_1 = \text{def}& \\
&X_0 = Agent_A \rightarrow Agent_B : query_if\langle\tau_i\rangle; X_1 \\
&X_1 = X_2 + X_3 \\
&X_2 = Agent_B \rightarrow Agent_A : inform\langle\tau_i\rangle; X_4 \\
&X_3 = Agent_B \rightarrow Agent_A : failure\langle\tau_i\rangle; X_{11} \\
&X_4 = Agent_A \rightarrow Agent_B : request\langle\tau_i\rangle; X_5 \\
&X_5 = X_6 + X_7 \\
&X_6 = Agent_B \rightarrow Agent_A : inform\langle\tau_i\rangle; X_8 \\
&X_7 = Agent_B \rightarrow Agent_A : failure\langle\tau_i\rangle; X_9 \\
X_8 + X_9 &= X_{10} \\
X_{10} + X_{11} &= X_{12} \\
&X_{12} = \text{end in } X_0
\end{aligned}$$

Figura 3.3: *Sintaxis de protocolo de delegación expresado en Attribute Global Types*

Capítulo 4

Caso de estudio

En este capítulo, se presenta el caso de estudio para la aplicación del lenguaje de especificación para la delegación de tareas en el entorno de la Web. Para la selección del caso de estudio, se realizó una evaluación de distintas opciones que complementen el dominio de interés. Durante el desarrollo de la tesis se trabajó en la implementación de *NautiLOD Distribuido* mediante agentes.

4.1. Descripción NautiLOD Distribuido

La plataforma *Linked Data* nos permite vincular distintos datos que están distribuidos en la Web. Por ello, muchas empresas e instituciones del Estado publican datos especializados relativos a su organización en la Web. Estos datos son almacenados en los *Endpoints SPARQL* y en un formato estándar como RDF. RDF es un formato que puede ser interpretado tanto por humanos y máquinas. En la Web podemos encontrar al rededor de quinientos *Endpoints SPARQL* cada uno especializado y enfocado a un determinado campo de interés por ejemplo: *Dbpedia.org*, es una información estructurada de Wikipedia; *Freebase.com*, es una base de conocimiento colaborativa compuesta por metadatos creados por los miembros de su comunidad; *GeoNames.org*, es una base de datos geográfica universal. Cada *Endpoint* es completamente independiente del otro e incluso estos pueden encontrarse separados geográficamente. Por ello, para obtener información almacenada en los *Endpoints* se deben realizar las consultas *(Query SPARQL)* de manera independiente a cada una de estas fuentes.

Navegar sobre esta red de fuentes de datos es un desafío. En la literatura podemos encontrar diferentes enfoques para resolver la navegación sobre la plataforma *Linked Open Data*. Dentro de ellas encontramos al Lenguaje NautiLOD. NautiLOD [39,41] es un lenguaje de expresión declarativo diseñado para especificar patrones de navegación en la plataforma Linked Open Data, este lenguaje está basado en expresiones regulares sobre predicados RDF entrelazados con pruebas *(Test)* del tipo *ASK* en SPARQL realizados sobre la descripción RDF de los recursos (URI). Por otro lado, se han

desarrollado dos herramientas que implementan el lenguaje NautiLOD. La primera es SWPORTAL [40], una plataforma Web donde se puede procesar expresiones NautiLOD mediante agentes. El agente personal aprovecha NautiLOD para atravesar fuentes de datos semánticos en busca de información relevante y proporciona notificaciones del resultado directamente al correo electrónico del usuario. La segunda herramienta es el comando SWGET [38] que está completamente desarrollada en JAVA, también disponible en su portal Web[1]. La herramienta está disponible tanto en línea de comando y como una interfaz gráfica de usuario (GUI). Sin embargo, tanto WSPORTAL y SWGET actualmente implementan todo el proceso de manera centralizada. Es decir, para procesar una expresión NautiLOD debe realizar consultas y solicitudes, utilizando solo el método GET del protocolo HTTP, a los diferentes *Endpoints SPARQL* para obtener información.

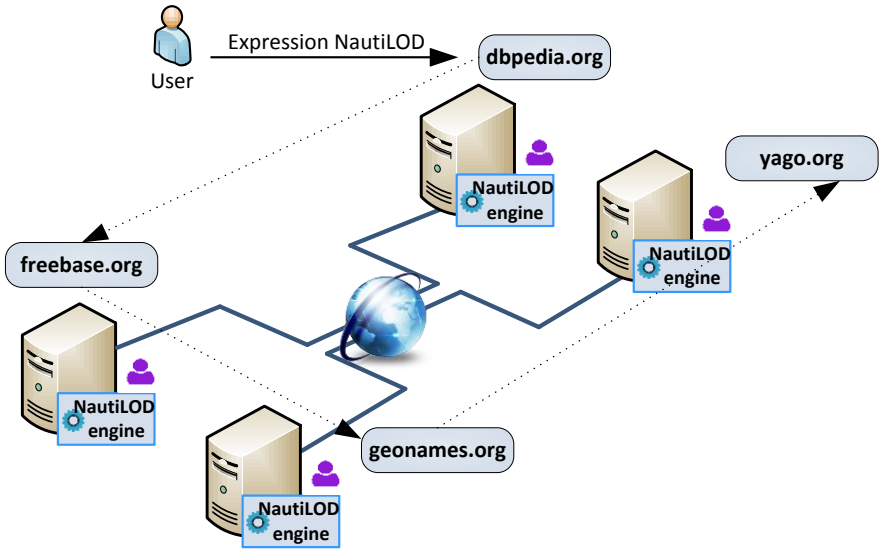

Figura 4.1: *Endpoints* distribuido mediante agentes

Para este caso de estudio planteamos desarrollar NautiLOD de manera **distribuida** mediante agentes. Se presenta el escenario descrito en la literatura [39] que consiste en la siguiente consulta: *"A partir del Endpoint de dbpedia.org, encontrar ciudades con más de 10.000 habitantes, junto con sus alias, en la que las personas sean músicos y que vivan actualmente en Italia"*.

4.2. Arquitectura para NautiLOD Distribuido

Mediante la Figura 4.1 se representa la arquitectura de NautiLOD Distribuido. Para que los agentes desarrollen sus capacidades se necesita tener un alto nivel de infraestructura de comunicación entre plataformas. El lenguaje de comunicación ACL establecido por FIPA sería el más

[1] https://swget.wordpress.com/portal

apropiado para esta infraestructura de comunicación. Por otro lado, una plataforma ideal para Sistemas MultiAgente sería JADE [9] ya que cuenta con características de una buena infraestructura de comunicación. Sin embargo no está orientado completamente al entorno de la Web a pesar que tiene soporte para el protocolo de comunicaciones HTTP. Por este motivo, se opta por la plataforma denominada *Agent Server*, descrita en el capitulo 5, que es completamente basada en la Web y desarrollada con características de una API REST. La plataforma *Agent Server* incorpora información relevante y funciones para la gestión de los agentes desde el comportamiento y el ciclo de vida. La arquitectura de esta plataforma incluye el Módulo del Protocolo de Transporte del Mensaje (MTP) para un procesamiento de mensajes fiable, para lo cual utiliza el Lenguaje de Comunicación de Agentes (ACL) estipulado por FIPA, y gestionado por los métodos GET, POST, PUT y DELETE del protocolo de comunicación HTTP. De la misma manera incorpora el módulo principal responsable de la gestión de la plataforma y los agentes. Los agentes que redicen en la plataforma *Agent Server* tienen implementada capacidades para cumplir el rol de:

- Un *Message Agent*, este agente es el encargado de la comunicación con los agentes externos de otras plataformas *Agent Server*, a través de este agente se centraliza toda la comunicación y las solicitudes de delegación. Para ello, el agente lee cada 4 segundos la cola de nuevos mensajes recibidos. Esta funcionalidad es definida en el modulo *AgentDefinitions* de *Agent Server* el cual se representa mediante la Figura 4.2

```
{
   "name" : "definitions1",
    "timers" : [
      {
         "name" : "message",
         "interval" : "seconds(4)",
         "script" : "message w; return w.read('only');",
      }
   ]
}
```

Figura 4.2: *AgentDefinition para leer cola mensajes ACL*

- Un *Delegation Agent*, este agente coordina las solicitudes de delegación propiamente dicha con los agentes, *Message Agent* y *Execution Agent*, que se encuentran en otras plataformas *Agent Server*. Todo el proceso se realiza mediante la verificación de las condiciones previas del operador *S-Delegate()*, definida en la sección 3.3.2, que deben de estar correctas.

- Un *Execution Agent*, una vez que una tarea τ fue aceptada por el agente contratante, debe ejecutar la tarea dada con las restricciones asociadas a ella. Este agente además coordina el proceso de ejecución.

Todos los agentes que participan en el proceso de la delegación tienen implementadas las mismas acciones, definidas en el Capitulo 3, lo que permite desarrollar sus capacidades de manera uniforme.

Se han configurado cuatro servidores virtuales de la plataforma Microsoft Azure en el cual se procedió a restaurar la base de datos de los *Endpoint SPARQL dbpedia.org*, *freebase.org*, *geonames.org* y *yago.org* respectivamente, las características de estos servidores son de la siguiente manera:

- **Para Endpoint dbpedia.org**

  Nombre de Host: dbpedias

  URL: `http://dbpedias.cloudapp.net/`

  IP: 40.84.150.156

  Ubicación: Centro y sur de EE. UU.

  S.O.: Ubuntu 14.10

  Hardware: Intel(R) Xeon(R) CPU E5-2673 v3 @ 2.40GHz, con 28 GB de RAM.

- **Para Endpoint freebase.org**

  Nombre de Host: freebases

  URL: `http://freebases.cloudapp.net/`

  IP: 40.84.150.157

  Ubicación: Este de EE. UU.

  S.O.: Ubuntu 14.10

  Hardware: Intel(R) Xeon(R) CPU E5-2673 v3 @ 2.40GHz, con 28 GB de RAM.

- **Para Endpoint geonames.org**

  Nombre de Host: geonames

  URL: `http://geonames.cloudapp.net/`

  IP: 100.78.14.82

  Ubicación: Este de EE. UU.

  S.O.: Ubuntu 14.10

  Hardware: Intel(R) Xeon(R) CPU E5-2673 v3 @ 2.40GHz, con 28 GB de RAM.

- **Para Endpoint yago.org**

  Nombre de Host: yagos

URL: `http://yagos.cloudapp.net/`

IP: 100.113.196.145

Ubicación: Norte de Europa

S.O.: Ubuntu 14.10

Hardware: Intel(R) Xeon(R) CPU E5-2673 v3 @ 2.40GHz, con 28 GB de RAM.

Por otro lado, mediante la Figura 4.3 se representa los *Endpoint SPARQL dbpedia.org*, *freebase.org*, *geonames.org* y *yago.org*, y los agentes $Agent_A$, $Agent_B$, $Agent_C$ y $Agent_D$ instalados en cada servidor. Cada uno de los agentes tiene implementado un *NautiLOD Engine* (Algoritmo 1) lo que, permite procesar las expresiones del lenguaje NautiLOD y consecuentemente delegar las tareas si en caso fuera necesario.

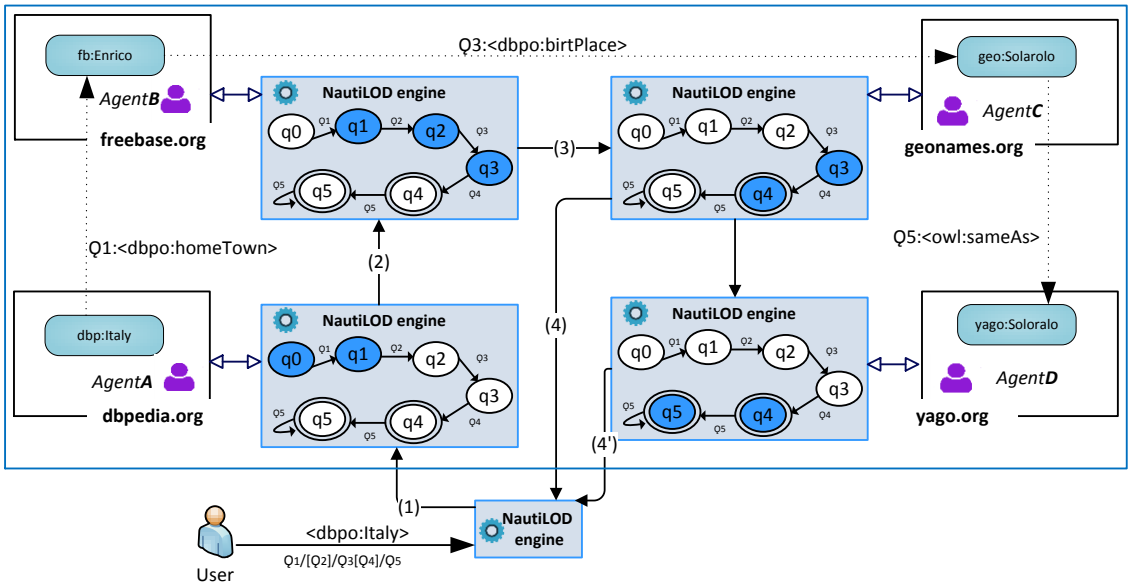

Figura 4.3: *NautiLOD* distribuido mediante agentes

Según las especificaciones del lenguaje NautiLOD, el usuario debe especificar una URI como punto de inicio. En la Figura 4.3 la URI de inicio se da en el Endpoint *dbpedia.org*. Así mismo, mediante *NautiLOD Engine*, los agentes participantes crean un nuevo autómata cada vez que se recibe una tarea.

4.3. Ejecución de NautiLOD Distribuido

Para la ejecución se planteó la siguiente consulta *"A partir del Endpoint de dbpedia.org, encontrar ciudades con más de 10.000 habitantes, junto con sus alias, en la que las personas sean músicos y que vivan actualmente en Italia"*. Para ello, mediante la Figura 4.4 se presenta la expresión del lenguaje

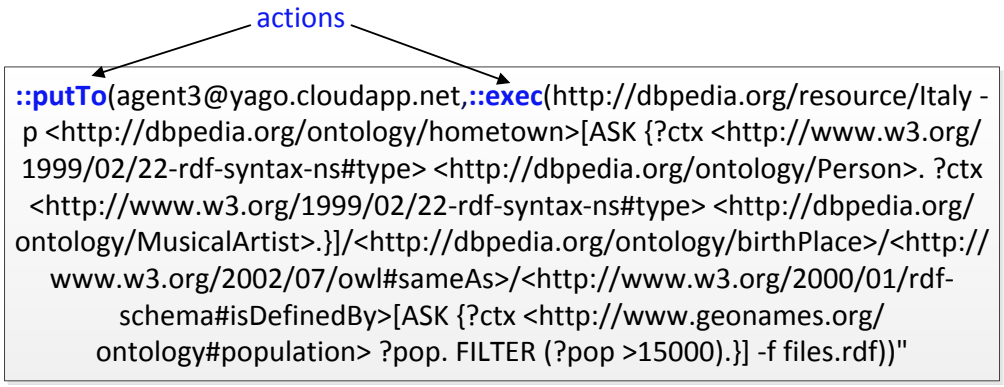

Figura 4.4: *Contenido del mensaje con acciones*

NautiLOD. Esta expresión consiste en un archivo RDF que tiene el Test ASK entrelazado y FILTER que permite evaluar triples que cumplen con el patrón establecido. Ademas, la expresión incorpora las acciones **putTo()** y **exec()** los cuales serán interpretadas por los agentes participantes. La acción **putTo()** indica que el resultado será entregado al agente *agent*3@*yagos.cloudapp.net* después de haber ejecutado la acción **exec()**.

Por otro lado, se presenta el Algoritmo 1 que permite la navegación del lenguaje NautiLOD. Este algoritmo es una extensión de SWGET, presentado en [39], que se agrega las funcionalidades: $prepareExpression()$ evalúa la expresión NautiLOD para determinar si la expresión es necesaria delegar hacia otro endpoint SPARQL; $constructNewQuery()$ construye la nueva expresión NautiLOD tomando como URI inicial (seed) la URI del resultado obtenido; $putTo()$ entrega el resultado de la ejecución al agente que fue especificado en el parámetro $Mdata$; $Delegate()$ que permite delegar la nueva $Expresion\ NautiLOD$ a otro Endpoint SPARQL.

A continuación se describe la ejecución de la Expresión NautiLOD, mediante el proceso de delegación:

- El *agent*1@*dbpedias.cloudapp.net* construye el autómata, de la figura 4.5, e inicia el procesamiento de la expresión NautiLOD en el estado $q0$. Obtiene desde su base de datos local la descripción de $\mathcal{D}(dbp)$ y busca las URI´s que tiene el predicado *dbp:hometown*. Obteniendo el resultado como resultado aquellos que satisfacen este patrón.

- El *agent*1@*dbpedias.cloudapp.net* realiza su primera transición de estado $\delta(q0, Q1) = q1$. El autómata no termina y el proceso continua.

- Como existen varias URI's que pertenecen a otros Endpoints, pero según la expresión inicial nuestro interés solo son URI's que pertenecen al Endpoint de *geonames.org*. El agente *agent*1@*dbpedias.cloudapp.net* se comunica con el agente *agent*2@*geonames.cloudapp.net*

Algoritmo 1: dswget pseudo-code

Input: e=NautiLOD expression; seed=URI; par=Parms<n,v>,Mdata
Output: set of URIs and literals conform to e and par;

```
1  Expr[]=prepareExpression(e);
2  a=buildAutomaton(Expr[0]);
3  addLookUpPair(seed, a.getInitial());
5  while ( ∃ p=<uri,state>to look up and checkNet(par)=OK) do
6      desc=getDescription(p.uri);
7      if (a.isFinal(p.state)) then
8          addToResult(p.uri);
9      if (not alreadyLookedUp(p)) then
10         setAlreadyLookedUp(p);
11         if (t=getTest(p.state)≠ θ and evalT(t,desc)=true) then
12             s=a.nextState(p.state,t));
13             addLookUpPair(p.uri,s);
14         if (act=getAction(p.state)≠ θ) then
15             if (evalA(act.test,desc)=true) then
16                 exeC(act.cmd);
17                s=a.nextState(p.state,act));
18                addLookUpPair(p.uri,s);
19         out=navigate(p,a,desc);
20         for (each URI pair p'=<uri,state>in out) do
21             addtoResults(p');
22         for (each literal pair lit=<literal,state>in out) do
23             if (a.isFinal(lit.state)) then
24                 addToResult(lit.literal);
25 newQuery[]=constructNewQuery(Expr[],Result);
26 if (newQuery.count >0) then
27     for (each newQuery in out) do
28         Delegate(URI,newQuery )
29 else
30     putTo(Mdata.Agent, newQuery);
31     sendResults(Mdata.Agent, lit.literal;
```

```
Q = ( 0 1 2 3 4 5 6 7 8 9 10 11 )
Q0 = ( 0 )
delta:
        0 -> 1 : <http://dbpedia.org/ontology/hometown>
        2 -> 3 : [ASK {?ctx <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                <http://dbpedia.org/ontology/Person>. ?ctx
                <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                <http://dbpedia.org/ontology/MusicalArtist>.}]
        1 -> 2 : ^
        4 -> 5 : <http://dbpedia.org/ontology/birthPlace>
        6 -> 7 : <http://www.w3.org/2002/07/owl#sameAs>
        8 -> 9 : <http://www.w3.org/2000/01/rdf-schema#isDefinedBy>
        10 -> 11 : [ASK {?ctx <http://www.geonames.org/ontology#population> ?pop.
                FILTER (?pop >10000).}]
        9 -> 10 : ^
        7 -> 8 : ^
        5 -> 6 : ^
        3 -> 4 : ^
A = ( 11 )
```

Figura 4.5: *Autómata inicial del lenguaje NautiLOD*

para el envío la expresión NautiLOD, mediante un mensaje de tipo ACL expresado como $msg(Agent_A, Agent_B, REQUEST(PutTo(Agent_C, Exec(Expr, Mdata)))$, donde $Agent_A$ representa a $agent1@dbpedias.cloudapp.net$, $Agent_B$ representa a $agent2@geonames.cloudapp.net$, $Agent_C$ representa a $agent3@yagos.cloudapp.net$, $Expr$ representa la expresión NautiLOD, $Mdata$ representa la metadata necesaria para la ejecución de la tarea. Cuando la solicitud llega al agente $agent2@geonames.cloudapp.net$, este evalúa la nueva expresión NautiLOD, con un razonamiento similar al agente inicial, y construye un nuevo autómata a partir de la URI enviado por el agente $agent1@dbpedias.cloudapp.net$. El agente $agent2@geonames.cloudapp.net$ tiene que comprobar las URIs que cumplen el Test ASK, es decir, si esta ciudad tiene más de 10 mil habitantes. A continuación se alcanza el estado final. El agente $agent2@geonames.cloudapp.net$ obtiene el resultado y finalmente se comunica con $agent3@yagos.cloudapp.net$, mediante mensajes tipo ACL expresado como $msg(Agent_B, Agent_C, REQUEST(Result(R_i)))$, para entregar el resultado.

- Opcionalmente, el agente $agent3@yagos.cloudapp.net$ mediante la acción **SendMessage**(`x@xyz`, R_i) notifica el resultado de la tarea, directamente a su correo electrónico del usuario solicitante.

4.4. Resultado preliminar de NautiLOD Distribuido

Cuando se inicia una tarea la plataforma genera un identificador, esto para hacer seguimiento de la misma sobre los demás Endpoints. La Figura 4.6 representa la interacción de los agentes partici-

pantes *agent*1@*dbpedias.cloudapp.net*, *agent*2@*geonames.cloudapp.net* y *agent*3@*yagos.cloudapp.net* y ocurre el siguiente proceso:

(1) El agente *agent*1@*dbpedias.cloudapp.net* recibe la tarea representada mediante la expresión NautiLOD que incluye las acciones a ejecutar el mismo que representado en la Figura 4.4. Estas acciones son *putTo* y *Exec* donde el agente *agent*1@*dbpedias.cloudapp.net* debe de ejecutar para resolver la tarea. Los primeros URI´s obtenidas desde el Endpoint *dbpedia.org* es presentado mediante la Figura 4.7 que representa a todos los triples que cumplen con el predicado *dbp:hometown*.

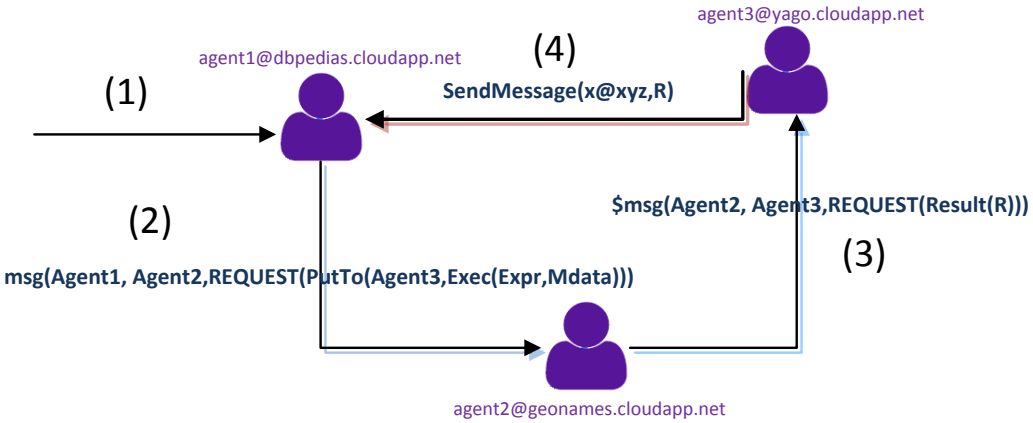

Figura 4.6: *Interacción entre agentes distribuidos*

(2) Del resultado obtenido por *agent*1@*dbpedias.cloudapp.net* se verifica si existen URI´s que pertenecen al Endpoint de *geonames.org*. Si en el caso que existiese estas URI's el agente *agent*1@*dbpedias.cloudapp.net* delega la tarea al agente *agent*2@*geonames.cloudapp.net* mediante el mensaje ACL de FIPA representado en la Figura 4.8, donde se puede observar que existen triples propios del Endpoint de *geonames* y estos mensajes son generados para cada URI obtenido del resultado.

(3) El agente *agent*2@*geonames.cloudapp.net* evalúa las expresiones NautiLOD con un rozamiento similar al agente *agent*1@*dbpedias.cloudapp.net*. Las URI´s obtenidas desde el Endpoint *geonames.org* es presentado mediante la Figura 4.9 que representa a todos los tiples que cumplen la condición $[ASK?ctx < geo : population >?pop.FILTER(?pop > 10000).]$.

(4) El agente *agent*2@*geonames.cloudapp.net* entrega, aquellas URI´s que cumplen el patrón especificado, al agente *agent*3@*yagos.cloudapp.net* mediante el mensaje ACL representado en la Figura 4.8. El resultado final de la tarea es representado por la Figura 4.10.

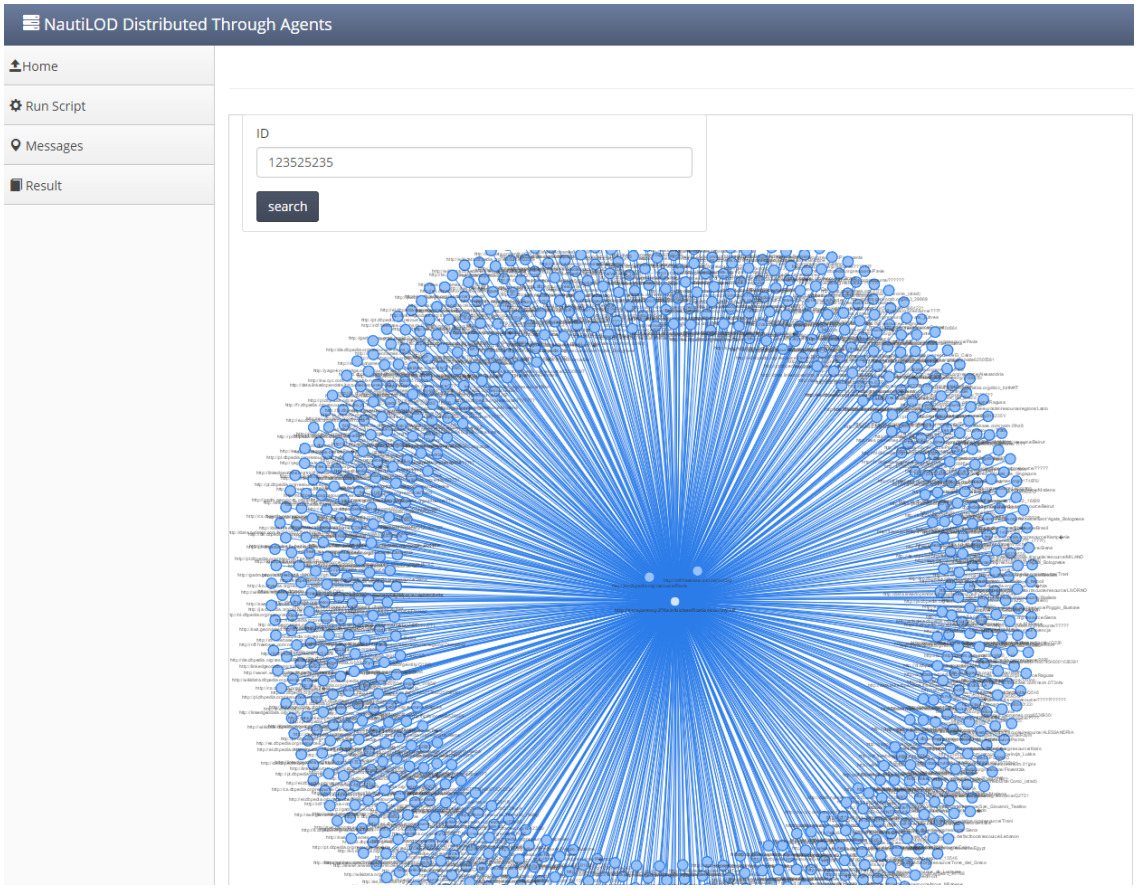

Figura 4.7: *Resultado obtenido en el Endpoint dbpedia.org*

(5) El agente *agent3@yagos.cloudapp.net* recibe todo los resultados y realiza la escritura en su base de datos para posteriormente realizar la notificación del resultado al usuario.

(6) De manera opcional, *agent3@yagos.cloudapp.net* notifica directamente al correo electrónico, el resultado obtenido.

El resultado final de la ejecución de la tarea es representado mediante la Figura 4.11, donde se puede observar como se va disminuyendo la carga de trabajo en cada Endpoint, obteniendo cada vez menos URI´s que cumplen el patrón especificado quedando finalmente el resultado de las URI´s obtenidas en el Servidor de $http://yagos.cloudapp.net/$.

```
{
  "conversationId": "geonames-123581844",
  "sender": "agent1@dbpedias.cloudapp.net",
  "receiver": "agent2@geonames.cloudapp.net",
  "replyTo": "agent3@yagos.cloudapp.net",
  "content": "::putTo(agent3@yagos.cloudapp.net,
              ::exec(http://sws.geonames.org/3165322/ -p
               <http://www.w3.org/2000/01/rdf-schema#isDefinedBy>
              [ASK {?ctx <http://www.geonames.org/ontology#population>
              ?pop. FILTER (?pop >10000).}] -f files.rdf))",
  "language": "",
  "encoding": "123525235",
  "ontology": "2",
  "protocol": "",
  "replyWith": "",
  "inReplyTo": "",
  "replyBy": "",
  "performative": "REQUEST"
 },
```


Figura 4.8: *Comunicación mediante mensajes ACL*

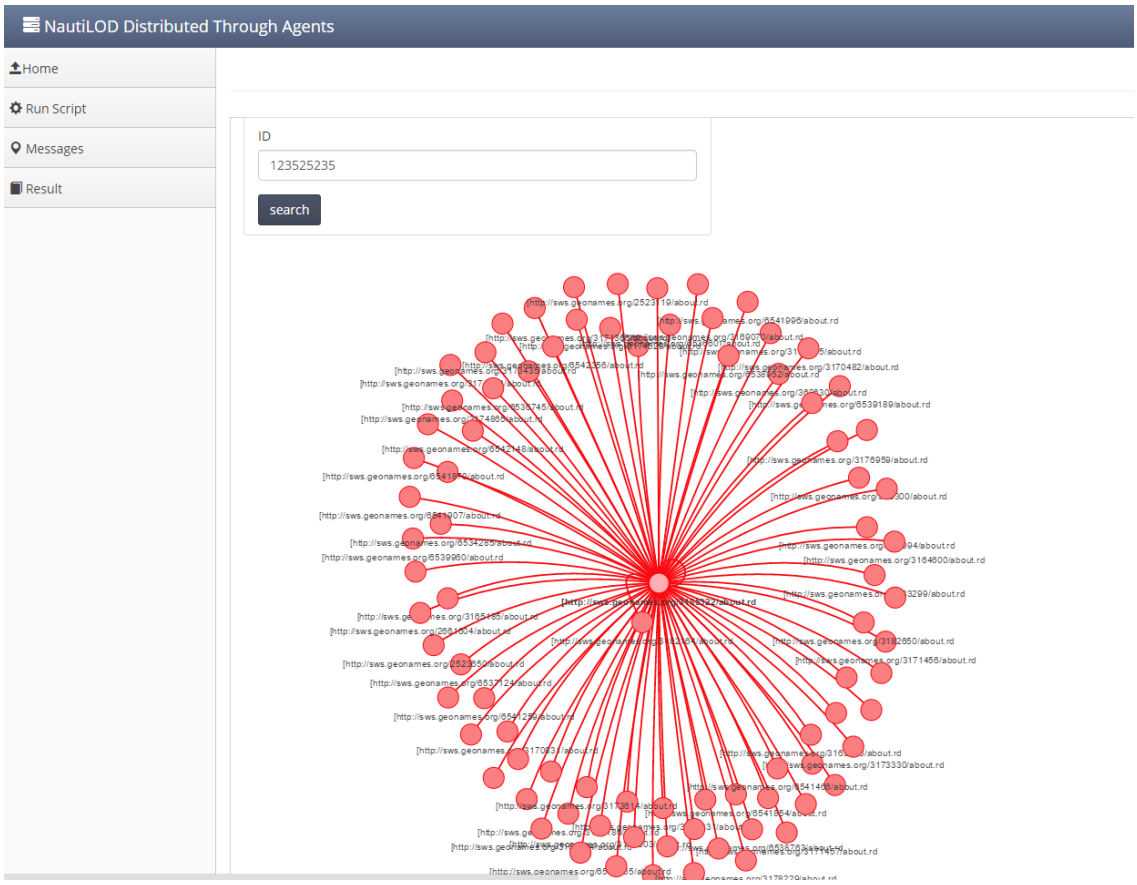

Figura 4.9: *Resultado obtenido en el Endpoint geonames.org*

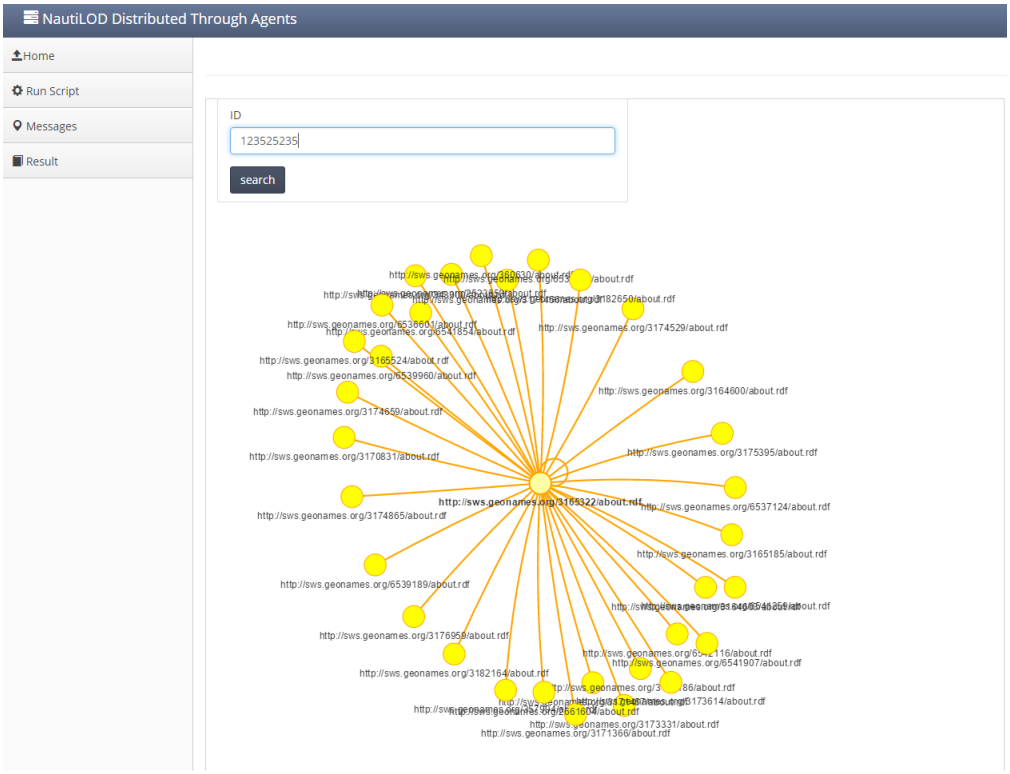

Figura 4.10: *Resultado entregado al Endpoint yago.org*

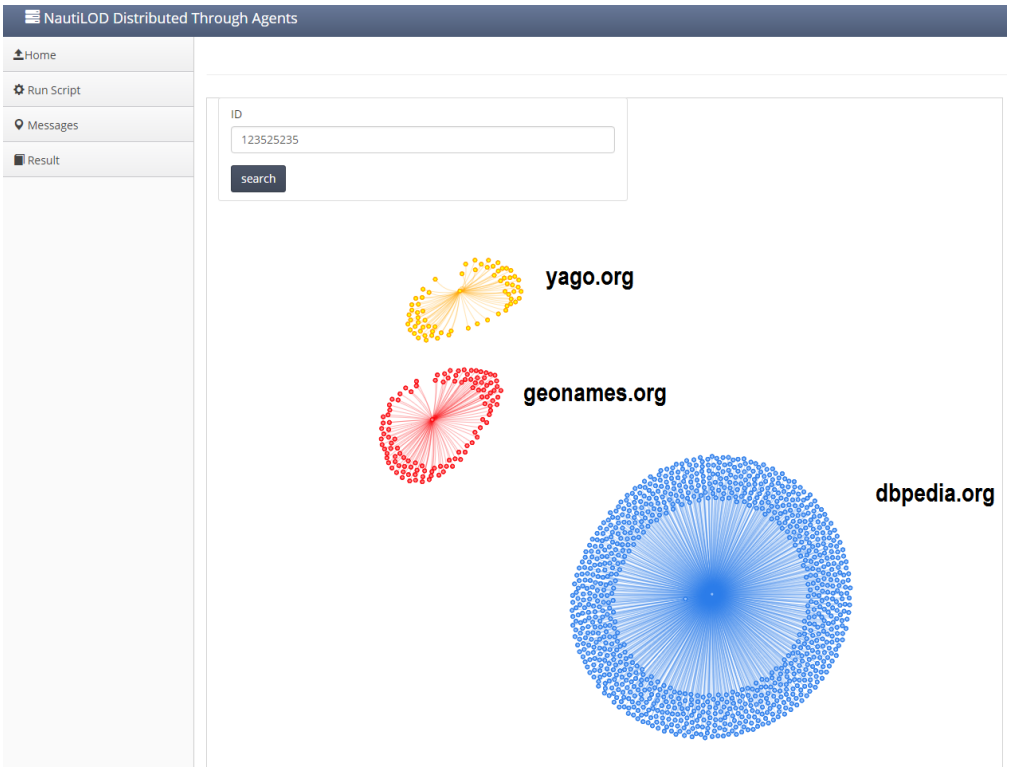

Figura 4.11: *Resultado final de NautiLod Distribuido*

Capítulo 5

Plataforma Agent Server

En este capítulo, presentamos el desarrollo de *Agent Server*, una plataforma flexible y escalable para Sistema Multi Agentes basados en el ambiente de la Web implementada completamente bajo los principios de REST donde los agentes pueden coexistir de manera persistente. En primer lugar, describimos la arquitectura de la plataforma *Agent Server*. Seguidamente nos enfocamos en el ciclo de vida de los agentes. Finalmente, describimos la gestión de la plataforma y su configuración para una eficiente ejecución del Sistema Multi Agente.

5.1. Descripción de Agente Server

El creciente uso de Internet para realizar tareas de la vida diaria hace necesario el desarrollo de software capaz de hacer frente a entornos abiertos y dinámicos. Frente a otras tecnologías usadas para desarrollar aplicaciones distribuidas en la Web, la tecnología de los Sistemas Multi Agente (SMA) se ha convertido en el pilar de una gran cantidad de aplicaciones, gracias a la incorporación de una base de conocimiento basada en acciones y tareas que pueden resolver problemas complejos. Así mismo, es esencial que un SMA cuente con características como alto rendimiento, seguridad y principalmente escalabilidad.

Según [22] los agentes se deben diseñar y aplicar en sistemas de software grandes que consisten en cientos de agentes y no solo en unos cuantos agentes, ya que para el desarrollo de estos sistemas se requieren plataformas eficientes y escalables. A pesar de los grandes avances realizados en el ámbito de los SMA utilizando el enfoque de FIPA, estos sistemas no escalan. Una de las razones por la que los SMA no escalan, es que estos sistemas no siguen los principios de la Web. La Web es un sistema completo diseñado sobre todo para una gran escalabilidad. Los sistemas basados SMA pueden ser desarrollados siguiendo diferentes técnicas al igual que la mayoría de las aplicaciones Web [5].
La World Wide Web implementa la arquitectura REST y por tanto, es altamente escalable. A fin de lograr una alta escalabilidad en el SMA, presentamos la plataforma *Agent Server*, desarrollada

completamente siguiendo los principios REST, que es una aplicación que utiliza el protocolo HTTP con sus métodos POST, GET, PUT y DELETE para pasar información a través de la URL y la información está expresada en formato JSON.

5.2. Background y trabajos relacionados

La interacción de los agentes mediante los métodos, GET, POST, PUT y DELETE del protocolo de comunicaciones HTTP, ha sido estudiando recientemente por Abdelkader y Bergeret [10, 49], que presenta REST-Agent un framework basado en la combinación de los conceptos de SMA y los principios de *Representational State Transfer* (REST) [36] definido por Fielding. Así mismo, Althagafi [5] propone un modelo de agentes que siguen los principios REST en un esfuerzo para hacer frente a los desafíos de la escalabilidad, especificando las locuciones de los agentes mediante *Actos Comunicativos* de la arquitectura FIPA [44] y los métodos GET, POST y PUT del protocolo de comunicaciones HTTP.

Una aplicación Web se puede implementar siguiendo los principios de los Servicios Web tales como SOAP y REST, enfoques completamente distintos. REST es un estilo de arquitectura para sistemas hipermedia distribuidos a gran escala, centrándose en el acceso de recurso a través de una única interfaz común. Por el contrario, SOAP es un protocolo para el intercambio de datos estructurados en la implementación de Servicios Web que expone operaciones. Se han realizado diversos estudios centrados en la comparación de SOAP y REST [21,69,70]. Ambas tecnologías, son similares respecto a las decisiones tecnológicas, sin embargo, REST ofrece más escalabilidad, interoperabilidad y rendimiento en comparacion con SOAP, mientras que SOAP ofrece más seguridad y fiabilidad.

Por otro lado, respecto a los trabajos relacionados, en los últimos años, muchos investigadores se han centrado en probar el rendimiento y principalmente la escalabilidad de las plataformas Multi Agente. Una de las principales propiedades probadas en este informe es la escalabilidad de la plataformas SMAs . Como se indica en [8], se han desarrollado una gran cantidad de plataformas SMA a través de los años y solo unos cuantos de estos sistemas siguen siendo desarrollados activamente. Sin embargo, en la actualidad es difícil encontrar alguna plataforma SMA con enfoque al ambiente de la Web. Dentro de la plataformas que más se aproximan a nuestro trabajo encontramos:

(a) JADE es el FrameWork de SMA más estable y ampliamente utilizado [9]. El sistema es compatible con FIPA [42] y proporciona una amplia gama de funcionalidades a los desarrolladores de agentes, ya sea como características integradas, o por medio de su extenso ecosistema de plug-ins. Puede ser utilizado para desplegar tanto reactividad y el razonamiento de los agentes.

La plataforma puede ser ejecutada como un conjunto de contenedores en la red de computadores. La interacción de estos contenedores se realiza mediante el protocolo RMI (Java Remote Method Invocation) y también tiene soporte para el protocolo de comunicaciones HTTP pero no está enfocado completamente al entorno de la Web.

(b) Por otro lado, Magentix [3] es una plataforma SMA que permite gestionar agentes distribuidos a través de múltiples computadoras. Está desarrollado a nivel de sistema operativo el cual proporciona altas tasas de eficiencia y escalabilidad. Sin embargo la interacción de los agentes está basada bajo el protocolo TCP y el proceso es centralizado. Es decir, es necesario tener un servidor central donde se encuentra la plataforma principal.

(c) Extensible Java EE-based Agent Framework (XJAF) [63] es una plataforma SMA construida bajo *Java Enterprise Edition (Java EE)* la que la diferencia de otras plataformas de agentes que están desarrolladas en *Java Standard Edition (Java SE)*. XJAF utiliza *Java Message Service* [1] para la gestión de la comunicación de los agentes, tecnología propia de Java EE. El principal objetivo de esta plataforma es hacer *cluster* de servidores para el balanceo de carga y tolerancia a fallos mediante agentes.

(d) *Agent Server Stage 0* [56] es una aplicación *Open Source* para agentes Web que se encuentra aún en su etapa 0 (preliminar) y está realizado con características de una API REST. Sin embargo, no está implementada la comunicación entre agentes. Por ello, en este trabajo nos concentramos en extender este trabajo desarrollando una versión más avanzada de esta aplicación a lo que denominamos *Agent Server*. *Agent Server* incorpora nuevas funcionalidades tales como: La gestión de mensajería siguiendo el Lenguaje de Comunicación de Agentes (ACL) estipulado por FIPA, delegación de tareas entre agentes, nuevas funcionalidades en el Lenguaje Scripting para el soporte para Web Semántica.

5.3. Arquitectura de Agent Server

La plataforma *Agent Server* incorpora información relevante y funciones para la gestión de los agentes desde el comportamiento hasta el ciclo de vida. En la figura 5.1 muestra la arquitectura de la plataforma *Agent Server* que incluye el Módulo del Protocolo de Transporte del Mensaje (MTP) para un procesamiento de mensajes fiable, para lo cual utiliza el Lenguaje de Comunicación de Agentes (ACL) estipulado por FIPA gestionado por los métodos GET, POST, PUT y DELETE del protocolo de comunicación HTTP. La plataforma de agentes contiene el módulo principal responsable de la gestión de la plataforma y los agentes.

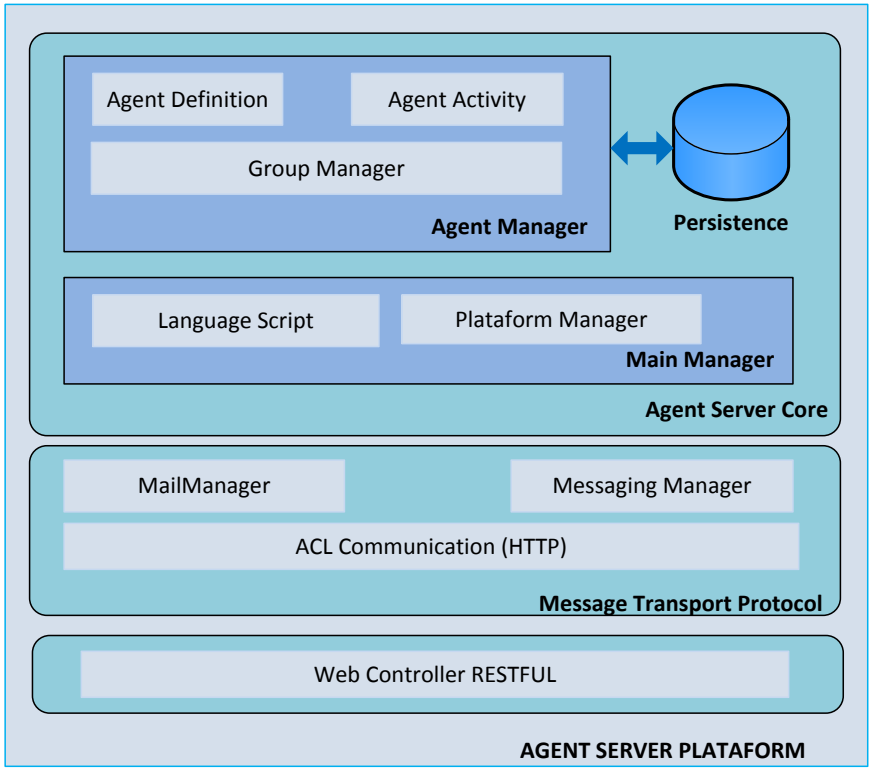

Figura 5.1: *Arquitectura de Agent Server Platform*

El módulo **Message Transport Protocol** está construido de la siguiente manera:

- *ACL Communication HTTP*. Este módulo es el encargado de gestionar la comunicación entre los agentes mediante los métodos GET, POST, PUT y DELETE.
- *MailManager*. Este módulo está construido para notificar al usuario de tareas completadas o que requieren alguna confirmación. Cuando se requiere una confirmación, o cualquier otra notificación el agente envía una URI al email del usuario.
- *Messaging Manager*. Este módulo es el encargado de gestionar el sistema de cola de mensajes, desde su recepción y envío de éstos. Los mensajes son almacenados en el módulo **Persistence** para su posterior procesamiento por los agentes.

El módulo **Agent Manager**, es el encargo de gestionar el ciclo de vida, el comportamiento y la definición de los agentes. Está construido de la siguiente manera:

- *Agent Definition*. Este módulo está construido para definir el comportamiento y el conocimiento de los agentes. Este se define como una colección de valores y propiedades que pueden ser desde simples funciones hasta expresiones o estructuras anidadas.

- *Agent Activity*. Este módulo está implementado para gestionar el ciclo de vida de los agentes y supervisa las actividades de los agentes. Así mismo gestiona las actividades de la plataforma que son el *Timer* y el número de *Thread* en ejecución.

- *Group Manager*. Este módulo esta implementado para gestionar los grupos de agentes. Permite importar la lista de agentes que residen en otros servidores y consecuentemente suscribirse a otros servidores. Este módulo nos permite alcanzar la escalabilidad de plataforma permitiendo al agente suscribirse automáticamente a grupos de agentes organizados.

El módulo **Main Manager**, módulo principal de la plataforma, encargado de gestionar el funcionamiento de la Plataforma *Agent Server*. El mismo que está construido de la siguiente manera:

- *Plataform Manager*. Este sub módulo es el encargado de inicializar y finalizar la plataforma *Agent Server*.

- *Language Script*. Este módulo es el encargado de resolver las definiciones de los agentes, desde funciones básicas hasta la extracción de datos de la Web. El lenguaje scripting se basa en Java, con una serie de simplificaciones y extensiones.

El módulo **Web Controller**. Este módulo es el encargado de gestionar la interacción de eventos por parte de los usuarios, gestiona las peticiones HTTP. La plataforma es gestionada completamente como una API RESTFUL que es soportado mediante el formato JSON.

El módulo **Persistence**, consiste en un archivo JSON que representa la Base de Datos de la plataforma *Agent Server*. Es el encargado de almacenar toda la información necesaria de los agentes, definiciones, mensajes y las propiedades necesarias para el funcionamiento de la plataforma *Agent Server*.

5.4. Funcionamiento de Agent Server

La plataforma, gestionada como una API REST, contiene componentes que permiten las interacciones entre los agentes y gestionar la ejecución de la plataforma *Agent Server*. La interacción de los componentes se muestra en la Figura 5.2 desde el inicio de la plataforma, el registro de usuarios, *Agent Definition*, agentes, las actividades del agente y hasta la gestión de los mensajes.

Cuando la plataforma *Agent Server* inicia su operación, los valores son almacenados por el modulo *Persistence* en el formato JSON, valores que pueden ser gestionados manualmente de acuerdo a las necesidades. Posteriormente se inicia el ciclo de vida de los agentes que consiste en lo siguiente:

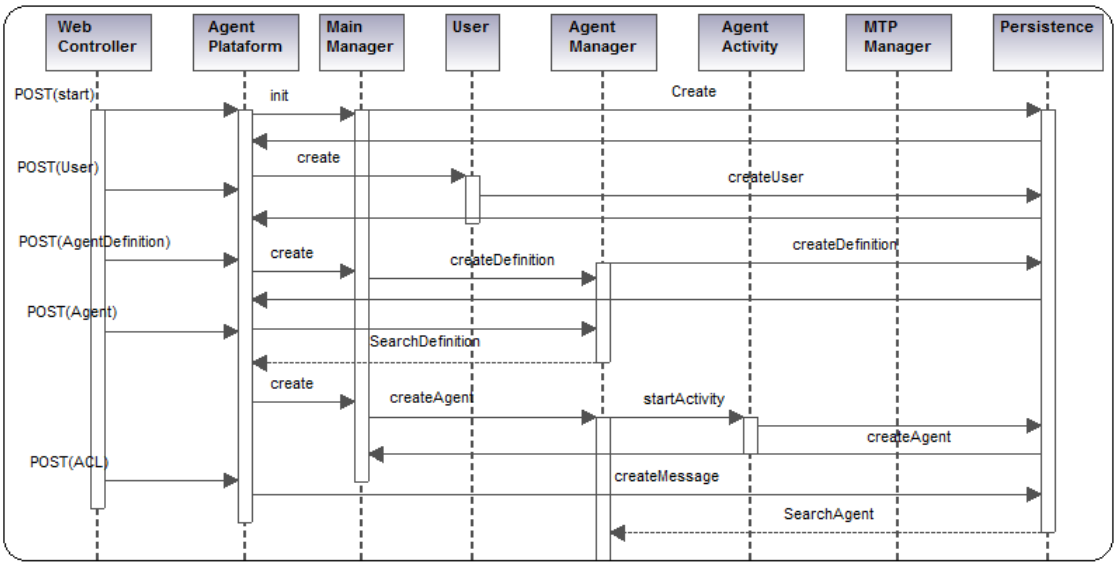

Figura 5.2: *Secuencia de ejecución de la plataforma "Agent Server"*

Para dar inicio a la operación de un agente, el usuario debe registrar sus datos en la plataforma con un identificador único. Luego define el *Agent Definition* donde se determina el comportamiento, conocimiento, tareas (acciones) y el tiempo que el agente va a demorar en realizar la actividad o cada cuanto tiempo va a realizar una actividad. El usuario crea la instancia de los agentes, a cada agente le asigna el *Agent Definition*. La plataforma inicia automáticamente los agentes tomando como referencia los valores y propiedades definidas en el *Agent Definition* de cada instancia del agente. El usuario puede activar, desactivar, eliminar, pausar y reanudar la instancia de agente en cualquier momento manualmente mediante los métodos del protocolo HTTP.

El usuario puede consultar el estado de la instancia del agente en cualquier momento; Esto incluye las entradas (inputs) y sus salidas (outputs); el estado interno (por ejemplo, campos de memoria y campos scratchpad); situación de las notificaciones de usuario; cualquier excepción que se haya producido durante la ejecución.

El usuario es responsable de responder a las notificaciones de casos de agentes. Normalmente, una notificación será un correo electrónico al usuario con enlaces a la API REST donde el usuario puede hacer clic para confirmar o hacer una selección basada en los valores de especificación de la notificación.

El usuario o una aplicación también pueden utilizar la API REST directamente para consultar y responder a las notificaciones.

La plataforma *Agent Server* reinicia automáticamente y restaura el estado de todos los agentes si se reinicia el servidor esto incluye incluso el apagado y reinicio del servidor y también si el servidor se bloquea o se reinicia.

5.5. Implementación de Agent Server

La implementación de la plataforma está enfocado en una API REST basada en el formato JSON, desarrollado con el Framework Spring MVC 4.0 de JAVA. Toda la administración de la plataforma se realiza mediante REST utilizando los métodos GET, POST, PUT y DELETE. Es decir al momento que inicia su operación, creación de usuarios, *Agent Definition*, agentes, sistema de mensajería y las respectivas consultas. A continuación se describe las principales clases y la función que desempeña dentro de la plataforma:

La clase *AgentServer*, forma parte del módulo **Main Manager**, es una de las principales clases que controla el funcionamiento de la plataforma que a su vez incorpora métodos que pueden ser gestionado manualmente por los usuarios desde el **Web Controller**, mediante los métodos de HTTP.

- *start()*, inicia la ejecución de la plataforma, puede ser gestionado por el método POST. (localhost:8080/start)
- *pause()*, pausa la ejecución de la plataforma, puede ser gestionado por el método PUT. (localhost:8080/pause)
- *getStatus()*, obtiene el estado de la plataforma, puede ser gestionado por el método GET. (localhost:8080/status)
- *restoreDefaults()*, restaura la plataforma con los valores predefinidos, puede gestionado a través del método PUT. (localhost:8080/restart)
- *shutdown()*, apaga la plataforma, puede ser gestionado a través del método PUT. (localhost:8080/stop)

Otra clase importante es *AgentActivity*, forma parte del módulo **Agent Manager**, controla la actividad de los agentes. Esta clase toma los valores y propiedades definidos en *Agent Definition*, e implementa los siguientes métodos:

- *startingActivity()*, gestiona el inicio de la actividad del agente.
- *performActivity()*, gestiona la realización de la actividad del agente.
- *gotException()*, obtiene la excepciones durante el desarrollo de la actividad del agente.
- *finishActivity()*, gestiona el término de la actividad del agente..

```
{
        "name" : "TestDefinition",
        "description" : "Test Definiton",
        "parameters": [],
        "memory" : [{
                "name": "counter",
                "type" : "int"
        }],
        "timers": [{
                "name": "count",
                "interval": "seconds(4)",
                "script": "counter++;"
        }],
        "outputs": [{
                "name": "output1",
                "type": "string",
                "compute": "\"Count is\"+counter"
        }],
        "notifications": [{
                "name": "Not1",
                "type": "notify_only",
                "manual": true
        }]
}
```

Figura 5.3: *Definición de los valores y propiedades de AgentDefinition*

En la clase *AgentDefinition* se define el comportamiento del agente. Los valores definidos pueden ser: *name, user, parameters, memory, inputs, timers, conditions, notifications, scripts, outputs*. En su mayoría estos valores son opcionales y se defines en una estructura en formato JSON es representado mediante la Figura 5.3. En esta estructura se define el comportamiento para un agente contador que va incrementando cada 4 segundos. Para ello, se define el nombre de la variable *counter* de tipo *int* con un *timer* de *seconds*(4) y el *script* permite incrementar su valor inicial y con opción para ser notificado al usuario.

La clase *AgentInstance*, en esta clase se define los datos del agente que pueden ser: *name, user, definition, parametervalues, publicoutput, enabled, state*. Con el valor *definition* se instancia al *Agent Definition*. Los métodos más relevantes que implementa son:

- *enable()*, habilita al agente para que pueda continuar su actividad. Puede ser gestionado mediante método PUT. (../agent/name/enable)
- *disable()*, deshabilita la actividad del agente. Puede ser gestionado directamente mediante método PUT. (../agent/name/enable)
- *delete()*, elimina al agente. Puede ser gestionado mediante método DELETE (../agent/name/delete)
- *getStatus()*, obtiene el estado de la actividades del agente. Puede ser gestionado mediante método GET. (../agent/name/status)
- *parseScript()*, valida los scripts definidos en *Agent Definition*.
- *evalueExpression()* evalúa los scripts definidos en *Agent Definition*. Puede ser gestionado mediante método PUT. (../name/evalue)
- *runScript()*, ejecuta los scripts definidos en *Agent Definition*. Puede ser gestionado mediante método PUT. (../namescript/runscript)
- *getEvent()*, obtiene los eventos del agente.
- *getInput()*, obtiene los inputs definidos en *Agent Definition*.
- *getMemory()*, obtiene los valores de memory definidos en *Agent Definition*.
- *getScripts()*, obtiene los scripts definidos en *Agent Definition*.
- *getOutput()*, determina los valores a mostrar definidos en *Agent Definition*.
- *getParemeters()*, obtiene los valores definidos en *Agent Definition*..

Finalmente, otra clase relevante es *ACLMessage*, forma parte del módulo **Message Transport Protocol**, dentro de los métodos más relevantes podemos considerar a: *send()*, encargado de enviar los mensajes al destino especificado mediante los métodos de HTTP; y *receive()*, encargado de leer mensajes recibidos que se encuentran almacenados en el sistema de cola de mensajes. Podemos crear un Agente que se encargue solamente de leer los mensajes o asignarle a cada Agente para que pueda leer sus mensajes. Esta clase es implementada siguiendo el Lenguaje de Comunicación de Agentes (ACL) estipulado por FIPA. Se utilizan los *Actos Comunicativos* de FIPA para establecer los protocolos de interacción de intercambio de mensajes, tal como se puede ver en la Figura 5.4. Por ello, la comunicación de los agentes será asertiva, directiva y declarativa.

```
{
  "conversationId": "geonames-123581844",
  "sender": "agent1@dbpedias.cloudapp.net",
  "receiver": "agent2@geonames.cloudapp.net",
  "replyTo": "agent3@yagos.cloudapp.net",
  "content": "::putTo(agent3@yagos.cloudapp.net,
              ::exec(http://sws.geonames.org/3165322/ -p
               <http://www.w3.org/2000/01/rdf-schema#isDefinedBy>
              [ASK {?ctx <http://www.geonames.org/ontology#population>
              ?pop. FILTER (?pop >10000).}] -f files.rdf))",
  "language": "",
  "encoding": "123525235",
  "ontology": "2",
  "protocol": "",
  "replyWith": "",
  "inReplyTo": "",
  "replyBy": "",
  "performative": "REQUEST"
},
```

Figura 5.4: *Composición del mensaje ACL estipulado por FIPA*

5.6. Aplicación de la plataforma Agent Server

La aplicación de *Agent Server* es muy amplio gracias a que todos los comportamientos de los agentes se puede especificar de manera declarativa, en una estructura del formato JSON. Para demostrar sus capacidades de *Agent Server* se ha evaluado de dos maneras:

- **Centralizada**, primero se ha creado un usuario denominado "test-user-1" al que se le ha asociado el *AgentDefinition*, definido en la Figura 5.3, lo que permite al agente cumplir el rol de agente contador que va incrementando su valor cada 4 segundos. Así mismo, en el menú "Test" del menú principal de *Agent Server* se ha implementado varios ejemplos: Web, permite extraer información especifica desde un sitio Web sea en formato HTML o RDF, verificar si un sitio Web esta activo; operaciones básicas, permite sumar, restar, generar bucles entre otros; listas, permite generar listas, mapas y entres otros.

- **Distribuida**, la validación se realizó mediante un caso de estudio descrito en el capítulo 4, denominado NautiLOD Distribuido, aplicado a una infraestructura con datos homogéneos como es Linked Data, en general se trabajó con servidores, *Endpoint SPARQL*, separados geográficamente. Donde los agentes que residen en diferentes servidores pueden comunicarse entre si para compartir información y delegar tareas entre ellos. Esta aplicación se encuentra dentro del menú "Aplications" del menú principal de *Agent Server*.

Una de las grandes ventajas que *Agent Server* ofrece es la definición del comportamiento de un agente a través de *AgentDefinition* que es completamente declarativo y se puede especificar en una estructura del formato JSON. Su uso esta enfocado a aplicaciones basadas en la Web como por

ejemplo se puede aplicar para extraer información de las redes sociales como Twitter cada cierto tiempo, verificar si una página esta activa de igual forma cada cierto tiempo, entre otras aplicaciones que podemos imaginar.

Finalmente en el menú "Tutorial" se encuentra una guía completa del uso de *Agent Server* desde la perspectiva de una API REST.

Capítulo 6

Resultados y Discusión

El principal resultado de esta investigación dice relación con el concepto de la delegación de tareas en el entorno de la Web, desarrollando agentes que sean capaces de comunicarse e intercambiar mensajes o recursos entre sí y estos estén separados geográficamente. La comunicación se realiza mediante el lenguaje de especificación para la delegación de tareas.

El lenguaje de especificación desarrollado durante la investigación puede tener un impacto significativo dentro de la comunidad de desarrolladores que rodean los temas relativos a agentes con enfoque al desarrollo de aplicaciones Web, donde sus componentes se encuentran distribuidos geográficamente y requieren delegar ciertas tareas o intercambiar información entre los participantes.

Con el desarrollo *Agent Server* se logra la escalabilidad [5] en la implementación de los agentes en el entorno de la Web donde los agentes pueden residir de manera indefinida. Su soporte para la Web semántica puede tener un impacto significativo en la comunidad científica de la Web. La combinación de los métodos del protocolo HTTP entre el Lenguaje de Comunicación de Agentes (ACL) estipulado por FIPA también podría llamar el interés en la comunidad científica enfocada al desarrollo de Sistemas MutiAgente.

Por otro lado, el desarrollo del caso de estudio "NautiLOD Distribuido" es una de las contribuciones que resuelve el problema de las consultas SPARQL distribuidas en diferentes Endpoints. Los agentes que lo conforman ayuda a obtener resultados mucho más específicos y con menos carga de trabajo.

El resultado de la Tesis se encuentra publicado en la Web y en la Figura 6.1 se presenta una esquema que concentra las direcciones tanto de *Agent Server*, *NautiLOD Distribuido* y el código fuente.

El desarrollo de la investigación ha sido orientado solamente a agentes con arquitectura reactivas. Esto se elige debido a la simplicidad del razonamiento que posee. En algún momento se pensó la implementación de agentes con arquitecturas BDI, pero estos agentes tienen mucha capacidad de

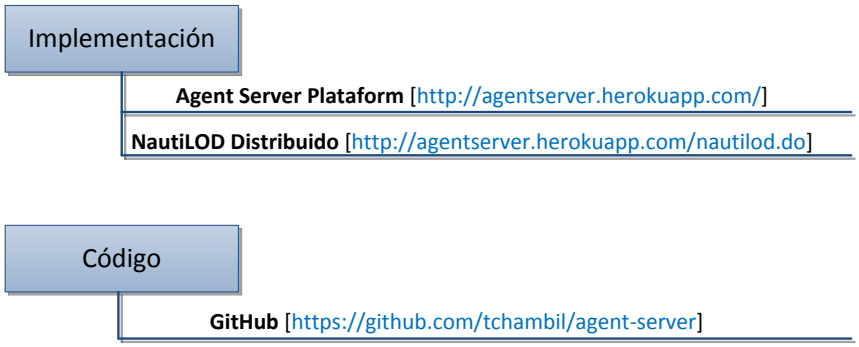

Figura 6.1: *Enlaces a sitios con resultado de la Tesis.*

razonamiento y requieren formalismos más complejos para modelar su comportamiento.

En una primera instancia se pensó trabajar solamente como sistemas distribuidos, específicamente con la plataforma JXTA [46]. Sin embargo, esta fue descartada ya que no permitía agregar algún grado de razonamiento. Se optó entonces por los Sistemas MultiAgente que, gracias a sus características permiten la interacciones de los agentes que lo conforman con algún grado de inteligencia.

Cabe mencionar que gran parte del esfuerzo dedicado al trabajo de Tesis estuvo concentrado en la implementación de "Agent Server" y "NautiLOD Distribuido"; el resto del tiempo se repartió entre el desarrollo del marco teórico y el diseño conceptual. La restauración de los datos en los Servidores Web tomó también gran parte del tiempo, ya que en un inicio solo se pensó en restaurar un cierta cantidad de datos. Sin embargo, el tamaño de los dumps de los Endpoints superaban la capacidad prevista, por ejemplo para restaurar los datos de freebase.org que pesa alrededor de 350 GB con procesador 8 Core Intel(R) Xeon(R) CPU E5-2673 v3 @ 2.40GHz y RAM 12GB RAM habían pasado cerca de dos semanas y no terminaba de restaurar, debido a ello se opta por un Servidor con mayor capacidad 16 Core Intel(R) Xeon(R) CPU E5-2673 v3 @ 2.40GHz, con 112 GB de RAM terminando de restaurar en cerca de tres días. Finalmente para que la comunidad científica pueda utilizar o testear *Agent Server* y *NautiLOD Distribuido* se crea un pequeño base de datos de grafos RDF que está incorporado en la plataforma.

Capítulo 7

Conclusiones y trabajos futuros

Este trabajo de tesis se propuso el diseño de un lenguaje de especificación para la delegación de tareas en el ambiente de la Web. Este lenguaje de especificación ha sido empleado para implementar el caso de estudio NautiLOD distribuido. Con este caso de estudio se pudo determinar las potenciales que nos ofrece un lenguaje que permita delegar tareas especificas en el entorno de la Web.

Uno de los pilares de este proceso fue el desarrollo de la plataforma *Agent Server* que permite gestionar Sistemas MultiAgente en el ambiente de la Web y una de la principales conclusiones de este desarrollo, es que para la comunicación de los agentes en el entorno de la Web es necesario que se siga la filosofía de especificaciones de los *Actos Comunicativos* de la arquitectura FIPA, siguiendo las especificaciones de la "interfaz común", establecido por REST. Tal especificación facilita la comunicación, permitiendo de este modo lograr la escalabilidad.

La validación del lenguaje de especificación sobre una infraestructura con datos homogéneos como es Linked Data fue crucial ya que gracias a su semántica los agentes son capaces de procesar su contenido razonar sobre este, combinarlo y realizar deducciones lógicas para resolver las consultas especificadas.

En lo que respecta al trabajo a futuro del lenguaje de especificación formal sería la implementación con agentes con arquitectura BDI, para así aprovechar el razonamiento más amplio que estos agentes nos ofrecen. De la misma manera, la implementación de la Semántica a través de algún formalismo, que permite especificar entidades distribuidas, como álgebra de proceso o session Type o simplemente utilizar la Lógica Temporal Action. En esta Tesis hemos mostrado a modo de ejemplo el caso de *Attribute Global Types*.

Los trabajos de desarrollo realizado durante la investigación con respecto a la plataforma *Agent Server* tiene muchas aplicaciones y permiten su continuación en algunos de los siguiente temas:

- Agregar funcionalidades para dar soporte a los agentes con arquitectura BDI.
- Agregar la funcionalidad para que el intercambio de información entre agentes se desarrolle

en formatos RDFs.

- Incorporar agentes auxiliares tipo *Sniffer Agent* del Framework JADE que permite el seguimientos de las interacciones que se producen y del intercambio de mensajes en tiempo real.
- Agregar soporte para la lectura de las bandejas de entrada de correo electrónicos POP3.
- Implementar una interfaz de usuario Web que permita gestionar los agentes desde los dispositivos móviles sea Android u otros.

Apéndices

Acto comunicativo FIPA y su enfoque en RESTFul

FIPA ACT	**DESCRIPCIÓN**	**MÉTODO RESTFul**
Accept-proposal	Aceptar una propuesta recibida previamente	POST/PUT
Agree	Estar de acuerdo en realizar alguna acción, posiblemente en el futuro	POST/PUT
Cancel	Cancelar alguna acción pedida previamente	DELETE
Cfp	Solicitar propuestas para realizar una acción	GET/POST
Confirm	Informar a un receptor que una proposición es cierta	POST
Disconfirm	Informar a un receptor que una proposición es falsa	POST
Failure	Informar a otro agente que se intentó una acción pero falló	POST
Inform	Informar a un receptor que una proposición es cierta	POST
Inform-if	El gente que recibe la acción cree que la sentencia es verdadera informará de manera afirmativa	POST
Inform-ref	El emisor informa al receptor de un objeto que cree que corresponde a un descriptor, por ejemplo su nombre.	GET/POST
Not-understood	Informar a un receptor que el emisor no entendió el mensaje	GET/POST
Propagate	El receptor trata el mensaje como si fuese dirigido a él, y debe identificar los agentes en este descriptor y enviarles el mensaje a ellos	GET/POST
Propose	Enviar una propuesta para realizar una cierta acción	POST
Query-if	preguntarle a otro agente si una determinada proposición es cierta	POST
Query-ref	Preguntar a otro agente por el objeto referenciado en una expresión	POST
Refuse	Rechazar realizar una acción	PUT
Reject-proposal	Rechazar una propuesta durante una negociación	PUT
Request	Solicitar a un receptor que realice alguna acción	POST
Subscribe	Una intención persistente de notificar al emisor de un determinado valor, y volver a notificarle cada vez que dicho valor cambie	POST

Cuadro 7.1: Especificaciones del acto comunicativo FIPA y su enfoque aplicado en RESTFul

Bibliografía

[1] Java message service homepage. `http://www.oracle.com/technetwork/java/`, [retrieved 25-05-2015].

[2] F. 2004. FIPA Agent Management Specification. Standard Component SC00023K, Foundation for Intelligent Physical Agents (FIPA),Geneve, Switzerland. http://www.fipa.org/specs/fipa00023/SC00023K.pdf.

[3] J. M. Alberola, J. M. Such, V. Botti, A. Espinosa, and A. García-Fornes. A scalable multiagent platform for large systems. *Computer Science and Information Systems*, 10(1):51–77, 2013.

[4] J. M. Alberola, J. M. Such, A. Garcia-Fornes, A. Espinosa, and V. Botti. A performance evaluation of three multiagent platforms. *Artificial Intelligence Review*, 34(2):145–176, 2010.

[5] A. H. H. Althagafi. Designing a framework for restful multi-agent systems. Master's thesis, Department of Computer Science, University of Saskatchewan, 2012.

[6] D. Ancona, M. Barbieri, and V. Mascardi. Global types for dynamic checking of protocol conformance of multi-agent systems. In *ICTCS*, volume 2012, pages 39–43, 2012.

[7] J. L. Austin. *How to do things with words*, volume 367. Oxford university press, 1975.

[8] C. Bădică, Z. Budimac, H.-D. Burkhard, and M. Ivanovic. Software agents: Languages, tools, platforms. *Computer Science and Information Systems*, 8(2):255–298, 2011.

[9] F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing multi-agent systems with JADE*, volume 7. John Wiley & Sons, 2007.

[10] M. Bergeret and A. Gouaïch. Rest-a and intercycle messages. In *Agent and Multi-Agent Systems: Technologies and Applications*, pages 82–91. Springer, 2010.

[11] T. Berners-Lee, J. Hendler, O. Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[12] G. Boella and L. van der Torre. Norm governed multiagent systems: The delegation of control to autonomous agents. In *Intelligent Agent Technology, 2003. IAT 2003. IEEE/WIC International Conference on*, pages 329–335. IEEE, 2003.

[13] G. Boella and L. Van Der Torre. δ: The social delegation cycle. In *Deontic Logic in Computer Science*, pages 29–42. Springer, 2004.

[14] J. Bradshow. An introduction to software agents. *Software Agents, JM Bradshow, Cambridge*, 1997.

[15] R. Brooks et al. A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of*, 2(1):14–23, 1986.

[16] R. A. Brooks. Elephants don't play chess. *Robotics and autonomous systems*, 6(1):3–15, 1990.

[17] C. Burnett, T. J. Norman, and K. Sycara. Bootstrapping trust evaluations through stereotypes. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 241–248. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[18] C. Castelfranchi. Modelling social action for ai agents. *Artificial Intelligence*, 103(1):157–182, 1998.

[19] C. Castelfranchi and R. Falcone. Principles of trust for mas: Cognitive anatomy, social importance, and quantification. In *Multi Agent Systems, 1998. Proceedings. International Conference on*, pages 72–79. IEEE, 1998.

[20] C. Castelfranchi and R. Falcone. Towards a theory of delegation for agent-based systems. *Robotics and Autonomous Systems*, 24(3):141–157, 1998.

[21] P. A. Castillo, J. L. Bernier, M. G. Arenas, J. Merelo, and P. García-Sánchez. Soap vs rest: Comparing a master-slave ga implementation. *arXiv preprint arXiv:1105.4978*, 2011.

[22] K. Chmiel, D. Tomiak, M. Gawinecki, P. Karczmarek, M. Szymczak, and M. Paprzycki. Testing the efficiency of jade agent platform. In *Parallel and Distributed Computing, 2004. Third International Symposium on/Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks, 2004. Third International Workshop on*, pages 49–56. IEEE, 2004.

[23] K. Decker, A. Pannu, K. Sycara, and M. Williamson. Designing behaviors for information agents. In *Proceedings of the first international conference on Autonomous agents*, pages 404–412. ACM, 1997.

[24] P. Doherty, F. Heintz, and D. Landén. A delegation-based architecture for collaborative robotics. In *Agent-Oriented Software Engineering XI*, pages 205–247. Springer, 2011.

[25] P. Doherty, F. Heintz, and D. Landén. A distributed task specification language for mixed-initiative delegation. In *Principles and Practice of Multi-Agent Systems*, pages 42–57. Springer, 2012.

[26] P. Doherty and J.-J. C. Meyer. Towards a delegation framework for aerial robotic mission scenarios. In *Cooperative Information Agents XI*, pages 5–26. Springer, 2007.

[27] C. N. Duarte, G. R. Martel, C. Buzzell, D. Crimmins, R. Komerska, S. Mupparapu, S. Chappell, D. R. Blidberg, and R. Nitzel. A common control language to support multiple cooperating auvs. In *Proceedings of the 14th International Symposium on Unmanned Untethered Submersible Technology*, pages 1–9, 2005.

[28] C. N. Duarte and B. B. Werger. Defining a common control language for multiple autonomous vehicle operation. In *OCEANS 2000 MTS/IEEE Conference and Exhibition*, volume 3, pages 1861–1867. IEEE, 2000.

[29] E. H. Durfee and T. A. Montgomery. Mice: A flexible testbed for intelligent coordination experiments. *Ann Arbor*, 1001:48103, 1989.

[30] E. H. Durfee and J. S. Rosenschein. Distributed problem solving and multi-agent systems: Comparisons and examples. *Ann Arbor*, 1001(48109):29, 1994.

[31] C. D. Emele. *Informing Dialogue Strategy through Argumentation-Derived Evidence*. PhD thesis, University of Aberdeen, 2011.

[32] R. Falcone and C. Castelfranchi. Levels of delegation and levels of help for agents with adjustable autonomy. In *Proceedings of AAAI Spring symposium on agents with adjustable autonomy*, pages 25–32, 1999.

[33] R. Falcone and C. Castelfranchi. The human in the loop of a delegated agent: The theory of adjustable social autonomy. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 31(5):406–418, 2001.

[34] R. Falcone, G. Pezzulo, C. Castelfranchi, and G. Calvi. Why a cognitive trustier performs better: Simulating trust-based contract nets. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1394–1395. IEEE Computer Society, 2004.

[35] J. Ferber and A. Drogoul. Using reactive multi-agent systems in simulation and problem solving. *Distributed artificial intelligence: Theory and praxis*, 5:53–80, 1992.

[36] R. Fielding. Representational state transfer. *Architectural Styles and the Design of Netowork-based Software Architecture*, pages 76–85, 2000.

[37] T. Finin, Y. Labrou, and J. Mayfield. Kqlm as an agent communication language. software agents. *MIT Press*, 29:1–316, 1997.

[38] V. Fionda, C. Gutierrez, and G. Pirró. Semantically-driven recursive navigation and retrieval of data sources in the web of data, 2011.

[39] V. Fionda, C. Gutierrez, and G. Pirró. Semantic navigation on the web of data: specification of routes, web fragments and actions. In *Proceedings of the 21st international conference on World Wide Web*, pages 281–290. ACM, 2012.

[40] V. Fionda, C. Gutierrez, and G. Pirro. The swget portal: Navigating and acting on the web of linked data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 26:29–35, 2014.

[41] V. Fionda, G. Pirrò, and C. Gutierrez. N auti lod: A formal language for the web of data graph. *ACM Transactions on the Web (TWEB)*, 9(1):5, 2015.

[42] FIPA. `http://www.fipa.org`, [retrieved 25-05-2015].

[43] A. Fipa. Fipa acl message structure specification. *Foundation for Intelligent Physical Agents, http://www. fipa. org/specs/fipa00061/SC00061G. html (30.6. 2004)*, 2002.

[44] T. FIPA. Fipa communicative act library specification. *Foundation for Intelligent Physical Agents, http://www. fipa. org/specs/fipa00037/SC00037J. html (30.6. 2004)*, 2008.

[45] S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Intelligent agents III agent theories, architectures, and languages*, pages 21–35. Springer, 1997.

[46] L. Gong. Jxta: A network programming environment. *Internet Computing, IEEE*, 5(3):88–95, 2001.

[47] S. Gottifredi and A. J. García. Análisis de lenguajes para especificación e implementación de agentes. In *VIII Workshop de Investigadores en Ciencias de la Computación*, 2006.

[48] S. Gottifredi and A. J. García. Análisis lenguajes de especificación de agente en robótica móvil. In *IX Workshop de Investigadores en Ciencias de la Computación*, 2007.

[49] A. Gouaïch and M. Bergeret. Rest-a: An agent virtual machine based on rest framework. In *Advances in Practical Applications of Agents and Multiagent Systems*, pages 103–112. Springer, 2010.

[50] A. Haddadi. *Communication and cooperation in agent systems: a pragmatic theory*, volume 1056. Springer Science & Business Media, 1996.

[51] P. Hayes and C. Menzel. A semantics for the knowledge interchange format. In *IJCAI 2001 Workshop on the IEEE Standard Upper Ontology*, volume 1, page 145, 2001.

[52] J. A. Hendler, A. Tate, and M. Drummond. Ai planning: Systems and techniques. *AI magazine*, 11(2):61, 1990.

[53] M. N. Huhns and M. P. Singh. *Readings in agents*. Morgan Kaufmann, 1998.

[54] M. N. Huhns and L. M. Stephens. *Multiagent Systems and Societies of Agents*. MIT Press Cambridge, 1999.

[55] N. R. Jennings. Specification and implementation of a belief-desire-joint-intention architecture for collaborative problem solving. *International Journal of Intelligent and Cooperative Information Systems*, 2(03):289–318, 1993.

[56] J. Krupansky. Agent server stage 0. `http://agtivity.blogspot.com/`, 2012 [accedido 25-05-2015].

[57] Y. Labrou and T. Finin. *Semantics for an agent communication language*. Springer, 1998.

[58] V. R. Lesser. Cooperative multiagent systems: A personal view of the state of the art. *Knowledge and Data Engineering, IEEE Transactions on*, 11(1):133–142, 1999.

[59] R. Leszczyna. Evaluation of agent platforms. In *Performance, Computing, and Communications, 2004 IEEE International Conference on, pages*, pages 857–864, 2004.

[60] J. Madejski. Agents as building blocks of responsibility-based manufacturing systems. *Journal of Materials Processing Technology*, 106(1):219–222, 2000.

[61] V. Mascardi and D. Ancona. Attribute global types for dynamic checking of protocols in logic-based multiagent systems. *TPLP*, 13(4-5-Online-Supplement), 2013.

[62] V. Mascardi, D. Briola, and D. Ancona. On the expressiveness of attribute global types: The formalization of a real multiagent system protocol. In *AI* IA 2013: Advances in Artificial Intelligence*, pages 300–311. Springer, 2013.

[63] D. Mitrović, M. Ivanović, M. Vidaković, and Z. Budimac. Extensible java ee-based agent framework in clustered environments. In *Multiagent System Technologies*, pages 202–215. Springer, 2014.

[64] C. B. T. J. Norman and K. Sycara. Trust decision-making in multi-agent systems. *In Proceedings of the 22nd International Joint Conference on Artificial Intelligence.*, 2011.

[65] T. J. Norman and C. Reed. A model of delegation for multi-agent systems. In *Foundations and Applications of Multi-Agent Systems*, pages 185–204. Springer, 2002.

[66] H. S. Nwana. Software agents: An overview. *The knowledge engineering review*, 11(03):205–244, 1996.

[67] J. J. Odell, H. V. D. Parunak, and B. Bauer. Representing agent interaction protocols in uml. In *Agent-oriented software engineering*, pages 121–140. Springer, 2001.

[68] G. M. O'Hare and N. Jennings. *Foundations of distributed artificial intelligence*, volume 9. John Wiley & Sons, 1996.

[69] C. Pautasso, O. Zimmermann, and F. Leymann. Restful web services vs. big'web services: making the right architectural decision. In *Proceedings of the 17th international conference on World Wide Web*, pages 805–814. ACM, 2008.

[70] P. K. Potti. On the design of web services: Soap vs. rest. Master's thesis, Computer and Information Sciences (MS) - College of Computing, Engineering Construction.

[71] A. S. Rao and M. P. Georgeff. Modeling rational agents within a bdi-architecture. *KR*, 91:473–484, 1991.

[72] S. Russell, P. Norvig, and A. Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25, 1995.

[73] F. SC00027H. Fipa query interaction protocol specification. *Foundation for Intelligent Physical Agents*, 2002.

[74] Y. Shoham. An overview of agent-oriented programming. *Software agents*, 4, 1997.

[75] R. Simmons and D. Apfelbaum. A task description language for robot control. In *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on*, volume 3, pages 1931–1937. IEEE, 1998.

[76] R. Strausz. Delegation of monitoring in a principal-agent relationship. *The Review of Economic Studies*, 64(3):337–357, 1997.

[77] W. van der Hoek, B. van Linder, and J.-J. C. Meyer. *An integrated modal approach to rational agents*. Springer, 1999.

[78] R. WESSON, F. HAYES-ROTH, J. W. BURGE, and C. STASZ. Network structures for distributed situation assessment. *Readings in Distributed Artificial Intelligence*, page 71, 2014.

[79] M. Wooldridge and N. R. Jennings. Agent theories, architectures, and languages: a survey. In *Intelligent agents*, pages 1–39. Springer, 1995.

[80] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The knowledge engineering review*, 10(02):115–152, 1995.

[81] F. XC00026. Fipa request interaction protocol specification. *Foundation for Intelligent Physical Agents*, 2000.