UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

ESCUELA DE CIENCIAS DE LA COMPUTACIÓN

INGENIERÍA DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

“Ontología para determinar situaciones de inseguridad. Nivel de comportamiento humano”

Tesis previa a la obtención del
Título de Grado de Ingeniero en Sistemas
Informáticos y Computación

Tesista:
Sandra Elizabeth Barahona Rojas

Director:
Ing. Héctor F. Gómez A.

Codirector:
Ing. Greyson Alberca P.

2009

Ing. Héctor Gómez

DIRECTOR DE TESIS

CERTIFICA

Que la Sra. **Sandra Elizabeth Barahona Rojas,** autora de la tesis Ontología para determinar situaciones de inseguridad. Nivel de comportamiento humano, ha cumplido con los requisitos estipulados en el Reglamento General de la Universidad Técnica Particular de Loja, la misma que ha sido coordinada y revisada durante todo el proceso de desarrollo desde su inicio hasta la culminación, por lo cual autorizo su presentación.

Loja, 25 de mayo del 2009.

Ing. Héctor F. Gómez A.

SISTEMAS DE INFORMACIÓN GEOGRÁFICA

CESIÓN DE DERECHOS

Yo, **Sandra Elizabeth Barahona Rojas,** declaro conocer y aceptar la disposición del Art. 67 del Estatuto Orgánico de la Universidad Técnica Particular de Loja que en su parte pertinente textualmente dice: “Forma parte del patrimonio de la Universidad la propiedad intelectual de investigaciones, trabajos científicos o técnicos y tesis de grado que se realicen a través o con el apoyo financiero, académico o institucional (operativo) de la Universidad”

Sandra Elizabeth Barahona Rojas

AUTORIA

Las ideas, opiniones, conclusiones, recomendaciones y más contenidos expuestos en el presente informe de tesis son de absoluta responsabilidad del autor.

Sandra Elizabeth Barahona Rojas

DEDICATORIA

Quiero dedicar este trabajo para el amor de mi vida, mi hijo, mi flaco bello, porque es lo mejor que me ha pasado y es sin duda la fuerza que me impulsa a seguir adelante.

A mi esposo que aunque ahora se encuentra lejos, siempre he recibido su apoyo y cariño incondicional.

A mi mami, por su amor y ayuda en todo momento, que ha sido madre y padre para mí, ha sabido formarme y hacer de mi una mejor persona inculcándome valores, principios, perseverancia y empeño, para así culminar mis metas.

A mis hermanos por confiar en mí y demostrarme día a día que todos somos diferentes pero muy importantes.

A mi pequeña sobrina por darnos alegría cada día e iluminar nuestras vidas.

Y de manera especial a Dios, ya que sin él, no tuviera a todas estas personas a mi lado, que me aman mucho.

Sandra Elizabeth Barahona Rojas

AGRADECIMIENTOS

Quiero agradecer primero a Dios por llenar mi vida de bendiciones, permitirme llegar hasta este momento tan importante y lograr una meta más.

A la Universidad Técnica Particular de Loja por poner a disposición los medios y recursos necesarios para mi formación y llegar a ser una persona capaz de servir a la sociedad.

A mi director Ing. Hector Gómez y codirector Ing. Greyson Alberca, quienes con sus conocimientos, orientación y ayuda desinteresada han sido fundamentales para la culminación de este trabajo.

A las personas del departamento de Sistemas de Información Geográfica de la Universidad por la ayuda prestada en el desarrollo del mismo.

A mi mami, hermanos y sobrina por darme la estabilidad emocional y sentimental para poder llegar hasta este logro, que definitivamente no hubiese podido ser realidad sin ustedes. Mamita, serás siempre mi inspiración para alcanzar mis metas, por enseñarme que todo se aprende y que todo esfuerzo es al final recompensa. Tu esfuerzo, se convirtió en tu triunfo y el mío.

A mi hijo, a mi pedacito de cielo que hace 12 años bajó a mi lado para hacerme la mujer más feliz del mundo, gracias por ser parte de mi vida. A mi esposo, que desde la distancia sabe apoyarme en las buenas y en las malas.

A todos mis amigos que me ayudaron a crecer y madurar como persona apoyándome en toda circunstancia.

ÍNDICE DE CONTENIDOS

CONTENIDO **PÁG.**

CERTIFICA ii

CESIÓN DE DERECHOS iii

AUTORIA iv

DEDICATORIA v

AGRADECIMIENTOS vi

ÍNDICE DE CONTENIDOS vii

ÍNDICE DE FIGURAS x

ÍNDICE DE TABLAS xii

PERFIL DEL ANTEPROYECTO DE TESIS 1

RESUMEN 3

INTRODUCCIÓN 5

MÓDULO 1 7

CAPÍTULO I: MARCO TEÓRICO 8

1.1 INTRODUCCIÓN 9

1.2 ONTOLOGÍA 9

1.3 CARETAKER 11

1.4 PROTÉGÉ 15

1.5 JADE 17

MÓDULO 2 20

CAPÍTULO II: METODOLOGÍA PARA CONSTRUIR ONTOLOGÍAS 21

2.1 INTRODUCCIÓN 22

2.2 METODOLOGÍAS 22

2.3 METODOLOGÍA A UTILIZAR 28

2.4 PERSONALIZACIÓN DE CARETAKER 29

2.4.1 COMPONENTES DE CARETAKER 29

2.5 PERSONALIZACIÓN DE LA ONTOLOGÍA PROPUESTA POR JADE 34

2.6 OBTENCIÓN DE LA ONTOLOGÍA FINAL 36

2.6.1 ESPECIFICACIÓN 36

2.6.2 CONCEPTUALIZACIÓN 36

2.6.3 FORMALIZACION, IMPLEMENTACION Y MANTENIMIENTO 48

2.6.4 DIAGRAMA PARA GENERAR ALARMAS 49

2.7 PERSONALIZACIÓN Y VALIDACIÓN DE LA ONTOLOGÍA 52

2.7.1 ONTOLOGÍA DE BOMBEROS INICIAL 53

2.7.2 MODIFICACIONES 54

2.7.3 ONTOLOGÍA FINAL 57

CAPÍTULO III: DISCUSIÓN 60

3.1 INTRODUCCIÓN 61

3.2 ONTOLOGÍA 61

3.3 METODOLOGÍA 61

3.4 RESULTADOS 62

3.5 FUTURAS INVESTIGACIONES 62

CONCLUSIONES 64

RECOMENDACIONES 66

BIBLIOGRAFÍA 67

ANEXOS 70

ANEXO A. ESQUEMA GENERAL DE LA ONTOLOGÍA CARETAKER 71

ANEXO B. DESARROLLO DE CLASES DE UNA ONTOLOGÍA USANDO PROTÉGÉ Y BEANGENERATOR 75

ANEXO C. INSTALACIÓN DE JADE 87

ANEXO D. EJECUCIÓN DE JADE, PARA QUE INTERACTÚEN LOS AGENTES CON LA ONTOLOGÍA FINAL. 90

ÍNDICE DE FIGURAS

FIGURAS **PÁG.**

Figura 1: Objetos Físicos de Interés de la Ontología CARETAKER 13

Figura 2: Objetos Contextuales de la Ontología CARETAKER 13

Figura 3: Eventos de Video de la Ontología CARETAKER 14

Figura 4: Jerarquía de CARETAKER, conceptos principales: Objetos Físicos y Eventos de Video. 15

Figura 5: Plataforma de Protégé 16

Figura 6: Soporte de Jade para lenguajes de contenido y ontologías. [Bellifemine2007] 18

Figura 7: Proceso de Desarrollo y Ciclo de Vida de METHONTOLOGY, traducido de [Gómez2003] 23

Figura 8: Procesos de la Metodología On-to-knowledge, traducido de [Gómes2003] 26

Figura 9: Diagrama General de CARETAKER 31

Figura 10: Incrementación de Objetos Físicos de Interés en CARETAKER, en la clase Person 32

Figura 11: Incrementación de Objetos Físicos de Interés en CARETAKER, en la clase Carro 32

Figura 12: Incrementación de Objetos Contextuales en CARETAKER, en la clase Zone 33

Figura 13: Propiedades creadas (viene_en_un, esta_en_zona_prohibida y tiene_que_ser_registrado) con Protégé, modificando a CARETAKER 33

Figura 14: Reglas creadas en Protégé modificando a la ontología CARETAKER 34

Figura 15: Diagrama inicial de la Ontología Bomberos, propuesta por Jade. 35

Figura 16: Diagrama final de la Ontología. 40

Figura 17: Diagrama conceptual instanciado en Protégé. 48

Figura 18: Diagrama de la interacción de los agentes con la ontología final. 49

Figura 19: Diagrama de casos de uso, demostrando la ejecución de los agentes. 50

Figura 20: Diagrama de casos de uso de la comunicación entre los agentes CentrslBomberos y Alarmado. 51

Figura 21: Esquema para un Sistema de Vigilancia en tiempo real. 63

ÍNDICE DE TABLAS

TABLA **PÁG.**

Tabla 1: Cuadro comparativo de las Metodologías: Methontology y On-to-Knowledge..... 28

Tabla 2: Secuencia para ejecución de los agentes CentralBomberos y Alarmado 51

Tabla 3: Secuencia de la comunicación entre los agentes CentralBomberos y Alarmado..... 52

Tabla 4: Resultados de la personalización y evaluación. 59

MODULO 1

PERFIL DEL ANTEPROYECTO DE TESIS

Título del proyecto:	**ONTOLOGIA PARA DETERMINAR SITUACIONES DE INSEGURIDAD. NIVEL DE COMPORTAMIENTO HUMANO**	
Duración: Propuesto por: Héctor F Gómez A	6 MESES Equipo:	
	Docente Investigador:	Héctor F Gómez A
	Tesista:	Sandra Elizabeth Barahona Rojas
	Línea de Investigación:	Clasificación de patrones en imágenes
Perfil Requerido del Tesista	- Interés matemático y lógico avanzado. - Nivel de ingles aceptable. - Conocimientos básicos de elaboración de proyectos. - Conceptos básicos de redes neuronales	

Propósito / Descripción

Identificar mediante ontologías situaciones de inseguridad en base al comportamiento humano a partir del resultado de procesamiento de un sistema de visión artificial.

Componentes:

Para le elaboración se ha dividido el desarrollo del proyecto en dos módulos

MODULO 1

Análisis de la situación actual seguridad vs inseguridad

MODULO 2

Determinación de objetos y elementos de inseguridad clasificados anteriormente.

Generación de la ontología.

Pruebas del desarrollo.

Estrategia o Metodología de desarrollo (Opcional)

Método inductivo

Resultados esperados

MODULO 1

Determinar situaciones de inseguridad en un entorno.

MODULO 2

Ontología que permita determinar las situaciones de inseguridad dadas por la investigación preliminar de la tesis.

Cronograma

Modulo 1: 1 mes
Modulo 2: 5 meses

RESUMEN

El presente trabajo tiene como objetivo identificar situaciones de inseguridad en base al comportamiento humano por medio de imágenes de video tomadas por cámaras de seguridad, usando ontologías.

Analizando las diferentes investigaciones en donde trabajan con ontologías se tomo como base la ontología de CARETAKER por el análisis de personas y las actividades que esta realiza, enfocándonos al objetivo del presente trabajo.

Una vez personalizada la ontología CARETAKER, se observó que es muy extensa, tiene muchos elementos que no son necesarios para el presente proyecto, lo que se optó por la personalización de otra ontología propuesta por JADE donde se caracterizó a una persona y objeto, elementos necesarios para poder reconocer si una persona ingresa a una zona prohibida o si un objeto fue removido o colocado en cierto lugar.

La ontología final consiste en tres partes: conceptos (persona y objeto), predicados (PersonaRevisada y ObjetoRevisado) y acciones (RevisarPersona y RevisarObjeto), esto con la ayuda del editor de ontologías Protégé.

Claro, la ontología solo caracteriza un entorno específico, por lo que mediante Jade (Java Agent Development Framework) se modificó a dos agentes, Alarmado y CentralBomberos (que se tomó de un ejemplo del manual de programación de Jade), los mismos que interactúan con la ontología antes descrita, y ésta toma decisiones.

Básicamente los agentes, están leyendo un archivo de texto (internamente tiene un objeto o persona) y en base a esta entrada, la ontología toma decisiones, como generar una alarma en la que se pide revisar lo que sucede en una zona.

Las alarmas generadas son: una persona está en zona prohibida, por lo tanto se debe revisar y un objeto fue detectado en cierto lugar, por lo tanto se tiene que revisar.

INTRODUCCIÓN

En la última década se ha marcado una alta tendencia al desarrollo de sistemas implementados en base a ontologías.

Uno de los retos en los que se enfrentan los investigadores actualmente es la necesidad de expresar el conocimiento humano en un lenguaje que pueda ser procesado por la máquina, es así que desde el nacimiento de la inteligencia artificial a finales de los años 50, varias investigaciones se han enfocado en la solución de esta y gracias a la evolución del pensamiento lógico, se abren nuevas líneas de investigación elaborando grandes bases del conocimiento en reglas lógicas capaces de ser procesada por cualquier agente software.

Debido a que la comunicación a nivel sintáctico (la forma en que se combinan las palabras) entre el humano y la máquina, aún no está completamente resuelta, se sigue investigando en cómo dar solución a este desafío, el cual consiste en definir bien la semántica (significado), detallando los conceptos y sus relaciones, propiedades y operaciones; todo en forma estructurada.

Es por esto que en el presente trabajo se investiga como elaborar una ontología que pueda determinar situaciones de inseguridad a partir del resultado de procesamiento de un sistema de visión artificial.

Iniciando con la investigación, primeramente se estudió la ontología CARETAKER, con la finalidad de ponerla en funcionamiento en un entorno más pequeño, se le realizó varios cambios, pero al final no se pudo lograr este objetivo, sin embargo se aprende muchísimo el cómo desarrollar una ontología. Además se investigó los agentes, para que puedan trabajar con las ontologías y CARETAKER no permite pasar sus clases a clases java, que es lo que normalmente utilizan los agentes. Por esta razón se trabajó con otra ontología, CentralBomberos la misma que está estructurada mediante tres partes, como son: conceptos, predicados y acciones, y sobre todo estos elementos son clases java. Todo esto se lo puede observar en el

capítulo I, iniciando con conceptos de ontologías y finalizando con algunas características de Protégé, que es lo que se utilizó para poder hacer los cambios en CARETAKER y JADE que se utilizó en la segunda ontología la misma que interactúa con agentes.

En el capítulo II, inicialmente se detalla las dos metodologías más importantes estudiadas por la Universidad Politécnica de Madrid, como son: Methontology y On-to-knowlegde, y se explica las diferencia de estas y la que se optó para el presente trabajo. Luego con la metodología escogida se detalla cada cambio hecho en las ontologías propuestas, hasta llegar a la ontología final.

Y en el Capítulo III, se realiza la discusión de los procesos seguidos para obtener la ontología final y la recomendación para futuras investigaciones.

MÓDULO 1

CAPÍTULO I

MARCO TEÓRICO

1.1 INTRODUCCIÓN

En este capítulo, se realiza una breve introducción sobre ontologías, se detalla las características de la ontología CARETAKER, la misma que está basado en una estructura de árbol, luego se describirá algunas utilidades del editor Protégé que permite trabajar con ontologías y se habla de la plataforma Jade y agentes.

1.2 ONTOLOGÍA

Hace muchos siglos en el campo de la filosofía se define a la ontología como “el estudio del ser”, parte de la metafísica que trata del ser en general y sus propiedades trascendentales [RAE].

Pero desde el punto de vista informático: “Las ontologías son teorías que especifican un vocabulario relativo a un cierto dominio. Este vocabulario define entidades, clases, propiedades, predicados y funciones y, las relaciones entre estos componentes, en donde toman un papel clave en la resolución de interoperabilidad semántica entre sistemas de información y su uso” [Redondo2009], porque se busca catalogar las palabras por su significado y no mediante palabras clave.

Actualmente ha dado una gran importancia en el estudio de la Web Semántica, donde prima la idea de transformar la red, de un espacio de información a un espacio del conocimiento.

Las ontologías son muy usadas por los expertos de una aplicación y dominio específico y se utilizan para entender los sistemas de una forma autónoma [Corvee2006]. Se construyen estos sistemas para entender de una forma óptima en base a expertos y a usuarios los términos en los cuales se describen las actividades dentro de un modelo. Además, la ontología es usada para evaluar una

escena que describe los sistemas, para entender exactamente qué tipo de eventos en un sistema en particular pueden ser reconocidos y para los desarrolladores, decidir que compartir y que actividades re-utilizar en modelos dedicados al reconocimiento de eventos específicos.

Las definiciones de la ontología para seguridad siempre tienen que darse en base a un lugar o un espacio apropiado, es decir los elementos de seguridad solo pueden darse sobre elementos físicos tal como se puede rescatar en [Zoe2006], en donde se habla de la construcción de una ontología que le permite a un robot poder guiarse con facilidad dentro de un edificio y sobre todo tomar sus propias decisiones en base a las características que presenta la misma, luego esto en términos de seguridad de hecho nos permite identificar los objetos físicos móviles y no móviles presentes en una escena.

La utilización de agentes, es de mucha ayuda, porque permiten simular una conversación, como si lo hicieran dos personas, y se están comunicando entre sí e interactúan con las ontologías de tal manera que esta pueda tomar decisiones, en el presente proyecto se trabajará con la plataforma Jade (Java Agent DEvelopment Framework), en donde se desarrollan aplicaciones con agentes.

Jade [JadeOntologías2007] define a la ontología, como “una entidad computacional que se compone de un conjunto de información que es útil en el dominio del problema en el que se está tratando”, y la estructura mediante conceptos, predicados y acciones, mediante clases java. De igual manera, en la sección 1.4 se detalla algunas características de Jade.

1.3 CARETAKER

La ontología CARETAKER está integrada en el proyecto con el mismo nombre (**C**ontent **A**nalysis and **Re**trieval **T**echnologies to **A**pply **K**nowledge **E**xtraction to massive **R**ecording) [Caretaker2006], el cual inició el primero de marzo del 2006 y finalizó el 30 de septiembre del 2008, tiene por objeto estudiar, desarrollar y evaluar el conocimiento multimedia basado en el análisis del contenido, la extracción de componentes, gestión automática de metadatos y sub-sistemas, el diagnóstico y el apoyo a la decisión.

Además, CARETAKER fue desarrollada por INRIA **(**Instituto Nacional de Investigación de Informática y Automática), probada en el metro de Roma y Turín y la extracción del conocimiento estructurado lo obtiene de las grandes colecciones multimedia, grabadas a través de las redes de cámaras y micrófonos desplegados en sitios reales.

Ésta ontología, está diseñada mediante una estructura de árbol, tiene dos clases principales, como son: Objetos Físicos y Eventos de Video y sus respectivas subclases, las cuales están detalladas en Anexo A. Dependiendo de lo que se va a modelar se puede hacer cambios a esta.

Construir una ontología para ser usada como referencia por aplicaciones de video es particularmente difícil porque muchos desarrolladores y expertos de aplicaciones de dominio tienen varias formas de describir el comportamiento humano. Estos términos permiten escoger el nombre para los conceptos en las ontologías y permiten día a día determinar ambigüedades en los mismos.

A continuación se detallará algunos términos que se utilizan en el análisis de escenas, según los resultados obtenidos al utilizar CARETAKER [Caretaker2006b]:

- **Escena**: Es el lugar observado por una o varias cámaras, es el espacio donde ocurren los eventos.

- **Objetos Físicos:** Son los elementos que conforman una escena. Existen dos tipos de objetos físicos: Un objeto físico de interés y un objeto contextual.

    - **Objetos físicos de interés**: Es un elemento de una escena cuyo movimiento no se puede predecir (personas, grupos de personas, muchedumbre, vehículos). Dependiendo de las características en dos dimensiones o tres dimensiones se etiquetan por clases semánticas, ancho y largo, postura, una trayectoria, una dirección, una velocidad, el tiempo inicial de grabación, una referencia a la cámara ubicada en la escena que mejor capta el objeto (en el caso de multicamaras) y un identificador. Los objetos físicos de interés que detalla CARETAKER son: Animal, Multitud, Grupo, Persona, Objeto Movible y Vehículo, dentro de este último: Avión, Automóvil, Tren y Camión, como se observa en la figura 1.

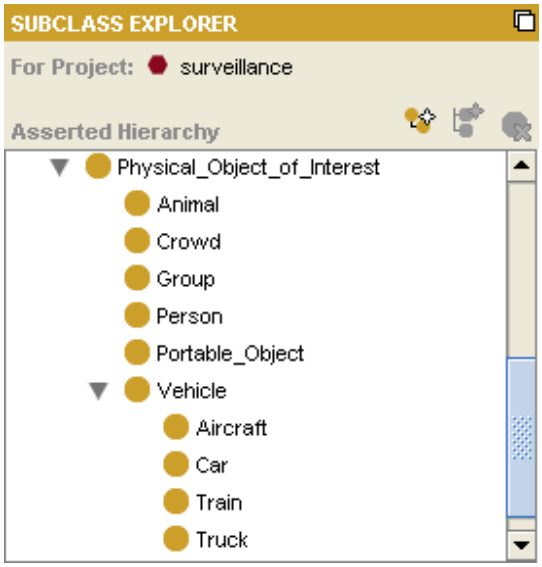


Figura 1: Objetos Físicos de Interés de la Ontología CARETAKER

- **Objeto Contextual**: Es un objeto físico que está relacionado con una escena, es estático, carece de movimiento o su movimiento se lo puede predecir, pueden ser: los movimientos de una puerta, un elevador, una silla y lugares previstos. En CARETAKER detalla como objetos contextuales a: Equipo, Pared y Zona, los mismos que se pueden observar en la figura 2.

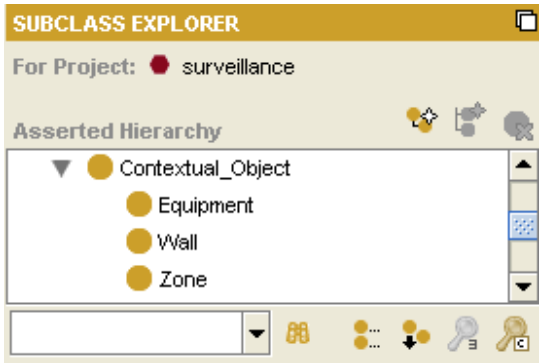


Figura 2: Objetos Contextuales de la Ontología CARETAKER

De igual manera en el análisis de videos se utilizan los siguientes términos:

- **Secuencia de video**: Son secuencias de imágenes de video tomadas por una video cámara.
- **Eventos de video**: Es una acción, evento o actividad que sucede en una escena y es observado por cámaras. Los eventos de video de interés, pueden estar predefinidos por cualquiera de los usuarios o por el aprendizaje del sistema. Los eventos de video son caracterizados por objetos de interés (incluyendo objetos contextuales y zonas de interés). Ejemplos de eventos son "detección de un vehículo en una zona", "detección de equipaje abandonado", "una reunión entre dos personas", etc.

En CARETAKER utiliza una clase para detallar los Eventos de Video, con cuatro subclases como son: Eventos Compuestos, Estados Compuestos, Eventos Primitivos y Estados Primitivos, como se ve en la figura 3, todo el esquema general se puede observar en el Anexo A.

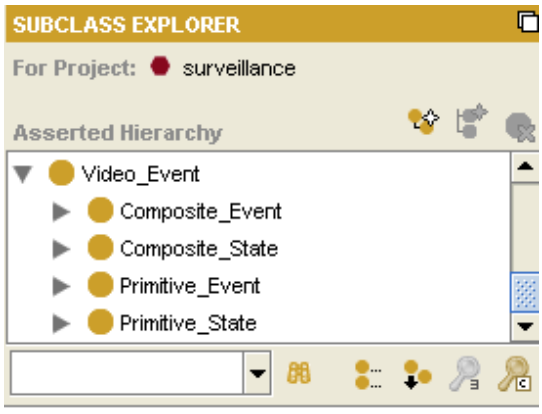


Figura 3: Eventos de Video de la Ontología CARETAKER

Como se dijo inicialmente, existen dos tipos de conceptos principales (clases) que están representados en CARETAKER: objetos físicos de las escenas observadas y los eventos de video/audio que ocurren en la escena. Como lo muestra la figura 4.

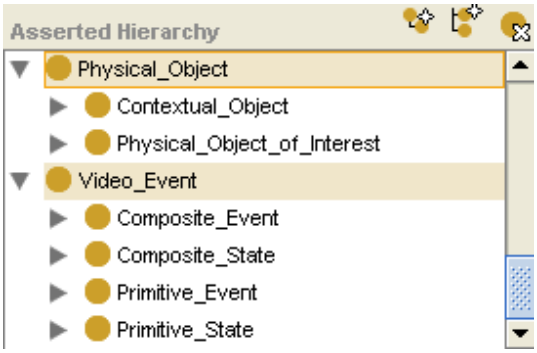


Figura 4: Jerarquía de CARETAKER, conceptos principales: Objetos Físicos y Eventos de Video.

Además, existen dos tipos de eventos en la ontología: El primero (ontología de usuario) que determina las direcciones de los escenarios reales para los usuarios finales. Estos escenarios de interés corresponden usualmente a los eventos compuestos. El segundo (visual, audio) describe los eventos primitivos que pueden ser detectados por tecnologías de audio video y este link activa el escenario de interés para los comportamientos efectivos detectables mediante tecnologías de audio y video.

1.4 PROTÉGÉ

El editor de ontologías Protégé, permite pasar clases ontológicas a clases java, la cual es la razón principal por la que se optó por esta herramienta. Esto se lo hace con el Plugin BeanGenerator, como se explica en el Anexo B, que mediante un ejemplo se detalla la utilización de este plugin.

A continuación de detallará brevemente las características del editor Protégé.

Protégé es una herramienta para el desarrollo de Ontologías y Sistemas basados en el conocimiento creada en la Universidad de Stanford, está desarrollada en JAVA y puede funcionar perfectamente bajo WINDOWS [IntroProtégé]. La herramienta Protégé emplea una interfaz de usuario que facilita la creación de una estructura de *frames* con clases, slots e instancias de una forma integrada. Como se observa en la figura 5.

Figura 5: Plataforma de Protégé

Protégé es libre, de plataforma Open-Source, implementa estructuras para modelar el conocimiento y acciones que apoyan a la creación, visualización y manipulación de ontologías en varios formatos, además se puede extenderse

mediante la arquitectura de plugins para construir aplicaciones basadas en el conocimiento [Protégé].

Otro de los plugins utilizados en Jambalaya, que permite visualizar el modelo conceptual de las ontologías. Este, viene ya en el instalador de Protege, pero también se lo puede obtener en [Jambalaya] y se carga al editor Protégé de la misma manera como se hizo con el plugin BeanGenerator, descrito en el Anexo B.

1.5 JADE

JADE (Java Agent DEvelopment Framework), es una plataforma de desarrollo de aplicaciones multi-agente conforme a los estándares de FIPA, ha sido implementado completamente en Java como código abierto [Jade].

FIPA (Fundation for Intelligent Physical Agent), es una organización sin fines de lucro, enfocada a la producción de estándares para la interoperabilidad de diferentes agentes software, se creó en 1996 para producir estándares para agentes heterogéneos y sociales y para sistemas basados en agentes. Es parte de la IEEE, aceptada el 8 de junio del 2005, para promover normas para nuevas aplicaciones basada en agentes [Fipa]. Su objetivo principal es la promoción de tecnologías y especificaciones de interoperabilidad que permitan el trabajo interno de sistemas de agentes inteligentes dentro del comercio y la industria.

Un agente, simplemente es un sistema computacional con la capacidad de tomar acciones autónomas en un medio, para así cumplir sus objetivos [AgentesJADE].

El soporte Jade para ontologías incluye las clases para trabajar con estas y con los lenguajes de contenido, entendiéndose a estos como la representación interna de los mensajes y las ontologías son la semántica de los mensajes que se intercambian y su chequeo.

Es decir, mediante el uso de ontologías se incorpora contenido semántico, y no sólo datos. Pero teniendo en cuenta que estas se definen en base a objetos de Java. En la figura 6, se puede observar el soporte que proporciona Jade para las ontologías.

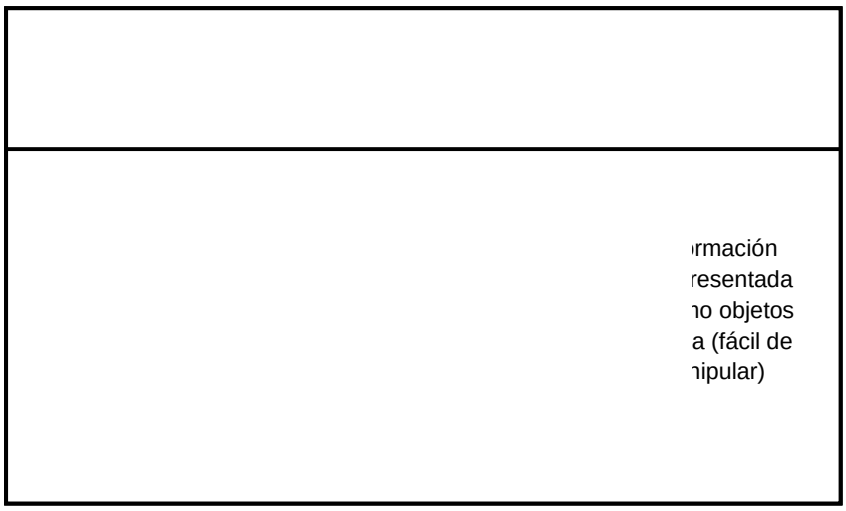


Figura 6: Soporte de Jade para lenguajes de contenido y ontologías. [Bellifemine2007]

Una ontología en Jade, está representada mediante tres tipos de elementos [JadeOntologías2007]: conceptos, predicados y acciones. Y al desarrollar los agentes en Jade, estos reconocen a la ontología, si está representada mediante estos, se crea una clase por cada uno y por último una clase general que representa la ontología en sí, la cual es usada en la comunicación. La clase general contiene toda la estructura de la ontología indicando cada uno de los nombres de los conceptos, acciones y predicados con sus respectivos atributos, e incluyendo la referencia hacia la clase con las cuales son implementados.

A continuación se explicará que significa cada uno de estos elementos [JadeOntologías2007]:

Conceptos: Son expresiones que representan objetos mediante una estructura con varios atributos. Por ejemplo: persona, vehículo, teléfono, objeto, etc.

Predicados: Son expresiones que relacionan a los conceptos, para decir algo. Es decir, lo que se obtiene después de haberse realizado una acción. Por ejemplo: zona revisada, persona revisada, etc.

Acciones: Son las expresiones que indican acciones que pueden realizar los agentes. Por ejemplo: comprar, vender, revisar persona, etc.

MÓDULO 2

CAPÍTULO II

METODOLOGÍA PARA CONSTRUIR ONTOLOGÍAS

2.1 INTRODUCCIÓN

En el presente Capítulo, primeramente se describirá brevemente dos de las metodologías más importantes para desarrollar ontologías, como son: Methontology y On-to-knowlwgde; y se realizará los procesos necesarios hasta llegar a la ontología final.

2.2 METODOLOGÍAS

Según el estudio realizado por la Facultad de Informática de Universidad Politécnica de Madrid, Asunción Gómez, Mariano Fernández y Oscar Corcho, en mayo del 2003, [Gómez 2003], sobre la Ingeniería Ontológica, se observa que existen varias metodologías y métodos que se pueden utilizar para la construcción de ontologías y las que mejores resultados han obtenido son: Methontology y On-To-Knowledge, que a continuación se detallará:

A. *Methontology*: Fue propuesta por la Fundación para los Agentes Físicos Inteligentes (FIPA) [Fipa], que provee normas y especificaciones para la interacción de agentes y sistemas basados en agentes.

Esta metodología habilita la construcción de ontologías a nivel de conocimiento, incluye: la identificación del proceso de desarrollo, un ciclo de vida basado en prototipos y técnicas para llevar a cabo cada actividad en dirección, desarrollo y soporte.

El proceso general de METHONTOLOGY se lo puede observar en la figura 7, donde las actividades de desarrollo son: Especificación, Conceptualización, Formalización, Implementación y mantenimiento. Y en cada una de estas, hay actividades de soporte como son: Adquisición del conocimiento, Integración, Evaluación, Documentación y Gestión de configuración. Además propone la

construcción de ontologías mediante prototipos porque permite la adición, mientras se está cambiando o quitando condiciones en cada versión (prototipo).

Para cada prototipo, METHONTOLOGY propone iniciar con la identificación de tareas a realizar, su disposición, el tiempo y los recursos necesarios para su finalización. Después de eso, la ontología y la especificación se inicia al mismo tiempo y en el transcurso de estas actividades, se realiza la gestión (control y aseguramiento de la calidad) y procesos de apoyo. La gestión y apoyo se realizan en paralelo con las actividades de desarrollo durante todo el ciclo de vida de la ontología.

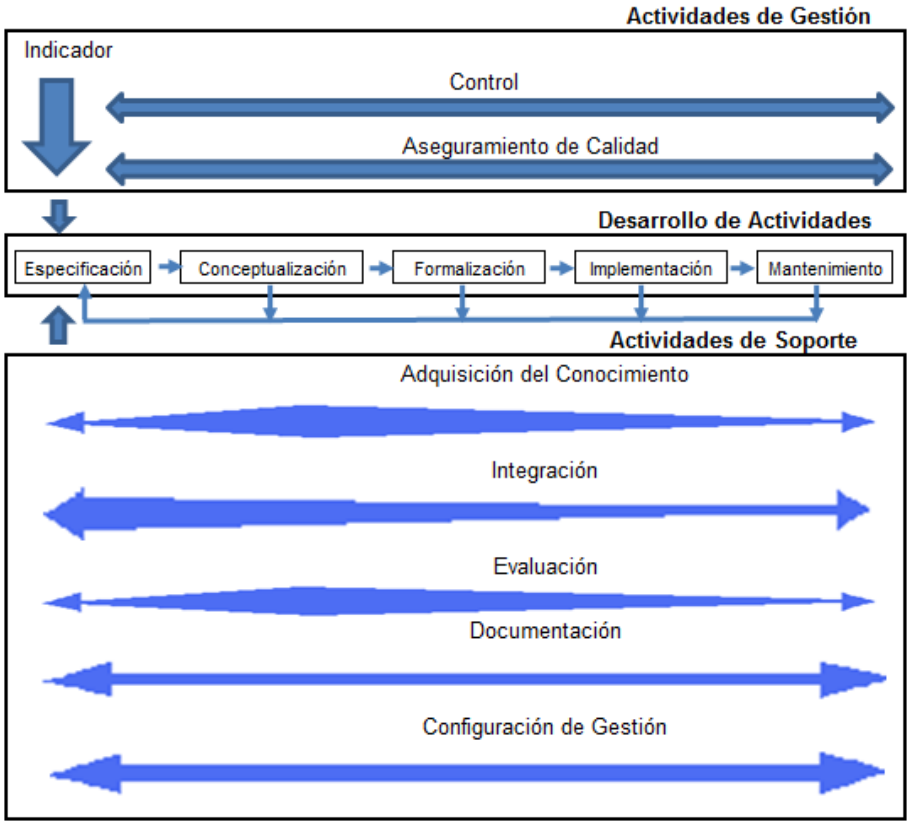


Figura 7: Proceso de Desarrollo y Ciclo de Vida de METHONTOLOGY, traducido de [Gómez2003]

Una vez especificado el primer prototipo, el modelo conceptual se construye dentro de la actividad de conceptualización de la ontología. Entonces se llevan a cabo la formalización e implementación. Si se delimitara más el proyecto, es aquí donde se puede volver a cualquiera de las actividades anteriores para hacer modificaciones o refinamientos.

Por consiguiente, la formalización no es una actividad obligatoria en METHONTOLOGY. Las actividades de desarrollo y de apoyo, se llevan a cabo simultáneamente. Pero una de las importantes es la evaluación, en donde se detalla los cambios en cada prototipo y sus resultados.

Las actividades de apoyo: adquisición de conocimiento, integración y la evaluación son importantes durante la conceptualización de la ontología, y disminuye durante la formalización e implementación.

METHONTOLOGY utiliza el método de re-ingeniería que consiste en implementar una ontología inicial, evaluarla, hacer cambios y volver a evaluar, de tal manera que se llegue a lo deseado.

A continuación se explicará brevemente cada una de las actividades que se realiza con METHONTOLOGY.

Actividades de desarrollo:

Especificación, se determina el objetivo a seguir.

Conceptualización, en esta actividad se obtiene el esquema inicial, en el cual se van a realizar las modificaciones, es decir, el prototipo inicial (modelo conceptual).

Formalización, aquí en cambio, en base al esquema inicial, se debe observar si cumple con el objetivo propuesto, es ahí donde se delimita a este o se replantea dependiendo de lo investigado.

Implementación, se realiza las modificaciones hasta llegar a las delimitaciones anteriormente propuestas.

Mantenimiento, se verifica las modificaciones realizadas.

Actividades de Soporte:

Adquisición del conocimiento, se obtiene la información necesaria para realiza una actividad de desarrollo.

Integración, se une la documentación obtenida y se integra al prototipo creando nuevas modificaciones.

Evaluación, es una de las actividades de mucha importancia, porque se evalúa los cambios realizados cumpliendo con los criterios propuestos para cada prototipo.

Documentación, como su nombre lo dice, se tiene constancia de los cambios realizados.

Configuración de Gestión, en esta actividad se observa el funcionamiento de la ontología y su resultado.

B. On-To-Knowlegde: Consiste en utilizar ontologías que se encuentren disponibles vía electrónica, para mejorar la calidad del conocimiento en organizaciones grandes y distribuidas, además se utiliza herramientas inteligentes para el acceso de grandes volúmenes de entrevistas semi-estructuradas y fuentes de información textual en entornos basados en internet. Las ontologías construidas por esta metodología servirán para utilizarlas en otras aplicaciones, por lo que son altamente dependientes de la

misma. Además propone la reducción de esfuerzos para desarrollar ontologías.

En la figura 8, se muestra los procesos que cumple esta metodología, la misma que consta de cinco procesos principales.

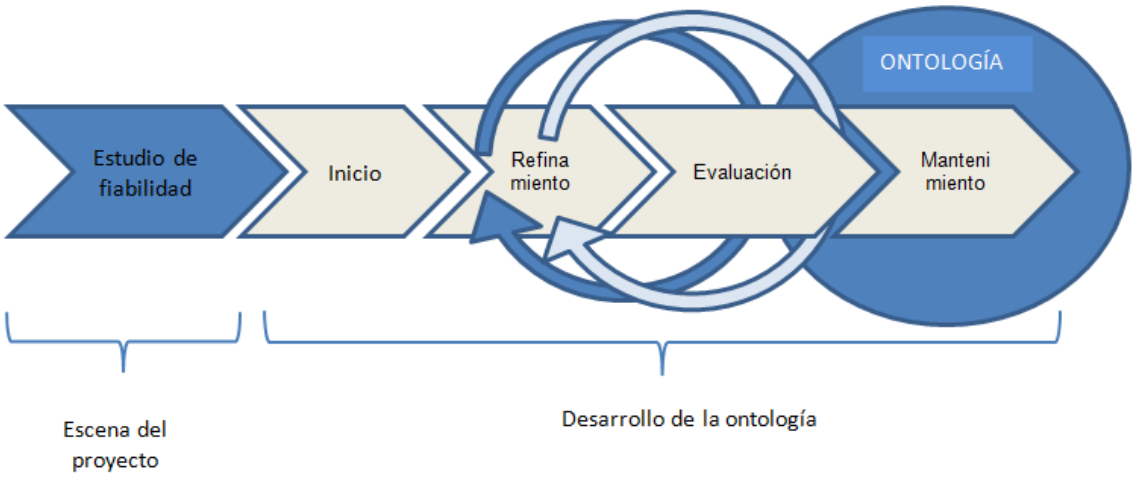


Figura 8: Procesos de la Metodología On-to-knowledge, traducido de [Gómes2003]

Estudio de la Fiabilidad, esta se lleva a cabo antes del desarrollo de la ontología y sirve como base para el proceso de lanzamiento.

Inicio, los resultados de este proceso especifican los requerimientos de la ontología, es decir, el dominio, los objetivos, los posibles usuarios, los casos de uso y el apoyo de aplicaciones de la ontología. Aquí se debe buscar ontologías ya desarrolladas para poder reutilizarlas.

Refinamiento, el objetivo es producir una aplicación madura y orientada a los objetivos de la ontología, de acuerdo a las especificaciones del proceso anterior. Tiene dos actividades: Obtención del conocimiento con expertos en el dominio y la formalización de este, es decir, seleccionar el lenguaje de la ontología de acuerdo a las necesidades iniciales.

Evaluación, sirve como prueba de la utilidad de la ontología, aquí se realizan dos actividades: Comprobación de los requisitos y ensayo de la ontología en el medio ambiente. Este proceso está estrechamente vinculado con el refinamiento, porque son necesarios varios ciclos hasta llegar a los objetivos iniciales.

Mantenimiento*,* en On-to-knowledge propone realizar mantenimiento de las ontologías como parte del software del sistema, está adaptada para crear empresas virtuales a fin de organizar memorias corporativas y proporcionar a los clientes, información adecuada sobre los productos o servicios que se desee recibir mantenimiento.

2.3 METODOLOGÍA A UTILIZAR

A continuación en la tabla 1, se detalla un cuadro comparativo de las metodologías analizadas.

Características	Methontology	On-to-Knowledge
Ciclo de vida propuesto	Evolución por prototipos y reingeniería.	Progresiva y cíclica. Evolución por prototipos
Estrategias con respecto a la aplicación	Aplicación independiente	Aplicación independiente
Estrategias para identificar conceptos	Middle-out (conceptos bien detallados)	Top-Down (menos estabilidad, mas trabajo) Bottom-up (difícil relación entre conceptos, inconsistencias) Middle-out (conceptos bien detallados)
Uso del código fuente de la ontología	Depende de los recursos disponibles	Depende de los recursos disponibles
Herramientas que dan soporte	ODE WebODE OntoEdit Protégé-2000	OntoEdit con sus plugins
Aceptación por organizaciones Externas	Recomendado por FIPA	VU Amsterdam, CognIT, Aidministrator, SwissLife, EnerSearch BT, Ontoprise GmbH, DFKI Kaiserslautern, Fraunhofer Institute for Integrated Publication and Information Systems, FIZ
Procesos	Actividades de desarrollo: - Especificación - Conceptualización - Formalización - Implementación - Mantenimiento Actividades de apoyo: - Adquisición del conocimiento - Integración - Evaluación - Documentación - Confifuración de gestión	- Estudio de fiabilidad - Inicio - Refinamiento - Evaluación - Mantenimiento

Tabla 1: Cuadro comparativo de las Metodologías: Methontology y On-to-Knowledge

No hay una metodología que sea la correcta para diseñar ontologías, solo existen alternativas, entre ellas tenemos a dos metodologías: Methontology y On-to-Knowledge. Para el presente trabajo se optó por utilizar la primera, porque además de estar basado en estándares por ser apoyado por FIPA, permite realizar

el método de reingeniería, es decir, tomar ontologías ya estudiadas para poder personalizarlas.

Dependiendo de las limitaciones que se vayan presentando en el transcurso del desarrollo de la ontología, Methontology permite ir determinando el alcance del proyecto hasta lograr a obtener la ontología deseada y evaluada.

2.4 PERSONALIZACIÓN DE CARETAKER

2.4.1 COMPONENTES DE CARETAKER

Como de explicó en el capítulo anterior, la ontología Caretaker esta compuestas por dos clases principales: Objetos físicos y Eventos de Video.

Dentro de los objetos físicos, caracteriza a: zonas, paredes, muebles fijos, animales, multitudes, grupos, personas, objetos movibles, vehículos (avión, carro, tren y camión).

En los eventos de video, que son las posibles actividades que se pueden dar al observar un video, como:

- Eventos de audio: eventos producidos cerca de un grupo agitado, actos de vandalismo en contra de una ventana.
- Eventos de monitoreo: compra de billete de entrada, monitoreo de la pista de los aeropuertos (arribo de los aviones, arribo de la tripulación, equipaje de carga, funcionamiento de remolque, descarga del equipaje), actos de vandalismo al comprar el billete de entrada.
- Eventos de vigilancia:
    - Eventos de vigilancia bancaria: clientes haciendo cola en el mostrador; cliente se acerca al mostrador y se va; clientes en

espera; ataque de una, dos o tres personas, ya sean solas o abriendo una puerta, frente o lejos de la ventanilla.

- Eventos en multitudes: pánico, división, mescla, dirección, congestionamiento, rápido aumento de multitud.
- Eventos genéricos: acceso a zonas restringidas, escaleras mecánicas, bloqueo de zonas, ingreso a zonas restringidas.
- Eventos en grupos: ataques, permanencia de un grupo en una zona, grupo se detuvo en una zona, vandalismo en contra de la máquina de billetes con dos personas.
- Eventos en personas: tratar a una persona, seguir a una persona, grafitis, persona saltando un asiento o barrera, robo a una o varias personas, vendiendo, vandalismo en contra de la máquina de billetes con una persona, vandalismo.
- Eventos de vigilancia en un tren: ataques, individuo sentado, vandalismo cerca de una ventana, robo con violencia.

- Eventos de estados: Pelea, permanecer en, estar lejos de, permanecer dentro de la zona de, permanecer fuera de la zona de, en espera.
- Eventos que se pueden dar con los objetos físicos: empieza a moverse, empieza a correr, detiene, se mueve la distancia de, se acerca a, cambios en la zona, introducir a la zona, sacar de la zona, se mueve la distancia desde, se acerca desde.
- Eventos de audio para estados: persona cerca a vidrios rotos, persona cerca a un evento, persona cerca a la llegada del tren.
- Eventos producidos en un objeto de interés:

- En un grupo: Variación del ancho, división rápida.
- En una persona: de pie, sentada, acostada, corriendo, hablando.
- Genéricos: movimiento, incrementa la velocidad, detenido, variación en la trayectoria.

- Eventos producidos en un muebles fijos: cerca de, lejos de.
- Eventos producidos en una zona: dentro de la zona, fuera de la zona.
- Eventos producidos en dos objetos de interés: cerca de la persona, después de la persona, sobrepasando la persona.

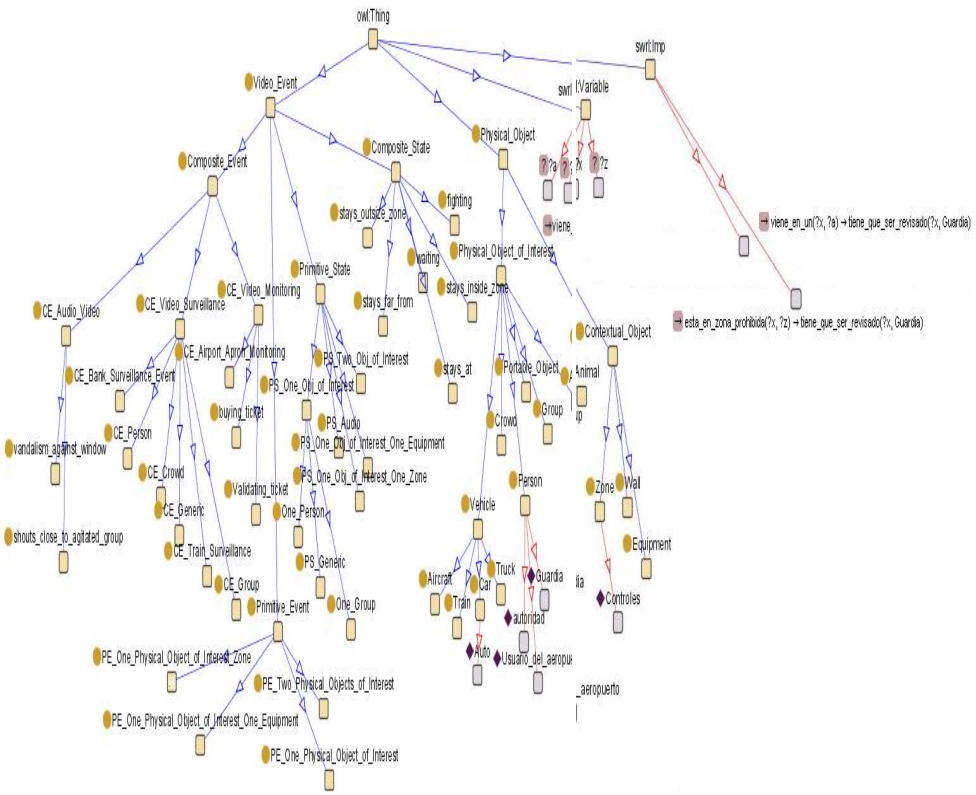


Figura 9: Diagrama General de CARETAKER

En la figura 9, se muestra el diagrama de CARETAKER y las modificaciones que se le realizó a ésta, que se detalla a continuación:

Objetos físicos de interés: Se instanció, en persona: guardia, autoridad y usuario de aeropuerto. Como se ve en la figura 10 y en la figura 11 se observa el incremento de auto en la clase carro, que está bajo vehículo.

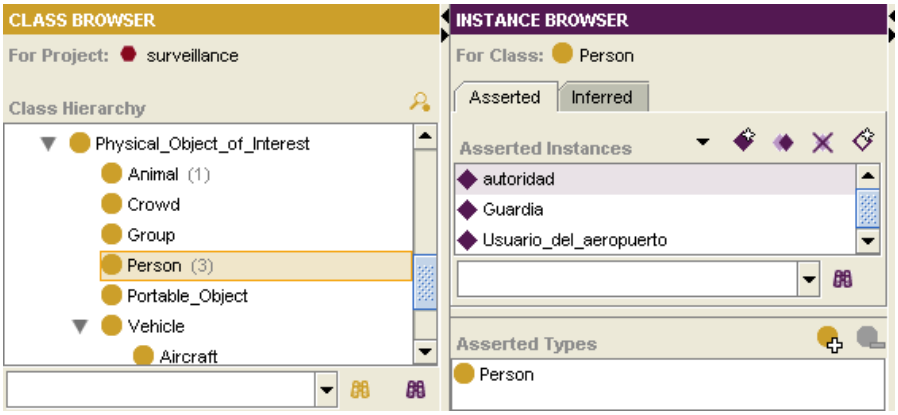


Figura 10: Incrementación de Objetos Físicos de Interés en CARETAKER, en la clase Person

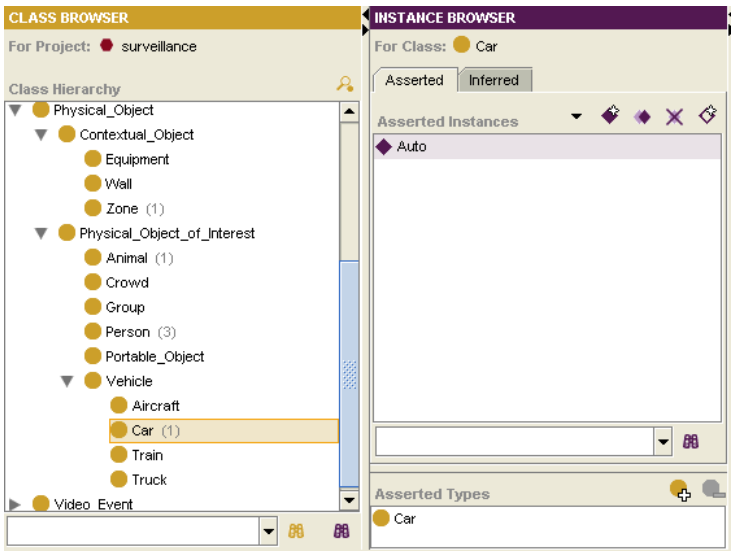


Figura 11: Incrementación de Objetos Físicos de Interés en CARETAKER, en la clase Carro

Objetos Contextuales: Se incremento en la clase Zona a Instancia, como se muestra en la figura 12.

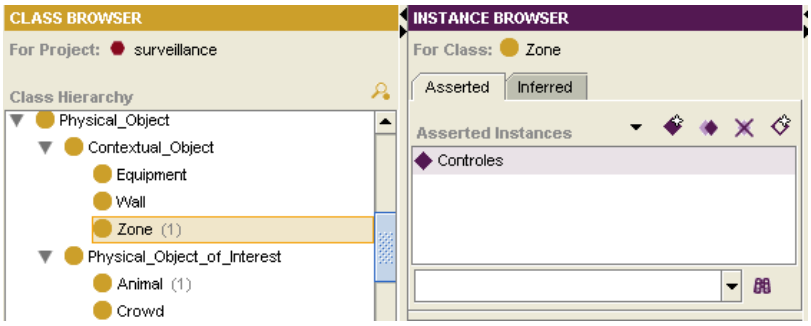


Figura 12: Incrementación de Objetos Contextuales en CARETAKER, en la clase Zone

Además se incrementó las propiedades: viene_en_un, esta_en_zona_prohibida y tiene_que_ser_revisado, como se muestra en la figura 13.

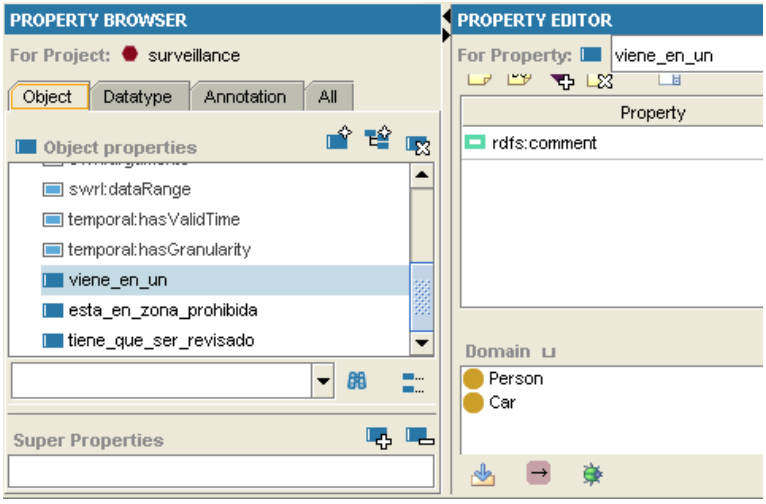


Figura 13: Propiedades creadas (viene_en_un, esta_en_zona_prohibida y tiene_que_ser_registrado) con Protégé, modificando a CARETAKER

Por último en CARETAKER se creó dos reglas, como se observa en la figura 14:

- ***Revisión**:* Consiste en preguntar si es que una persona viene en un vehículo entonces tiene que ser revisado por otra persona en este caso un guardia, en Protégé se la representa así: viene_en_un(?x, ?a) → tiene_que_ser_revisado(?x, Guardia).

- ***Zonas_Prohibidas**,* en donde se analiza dependiendo de la persona, si está o no en zona prohibida, entonces igual se tiene que revisar a esta persona, representándose así: esta_en_zona_prohibida(?x, ?z) → tiene_que_ser_revisado(?x, Guardia)

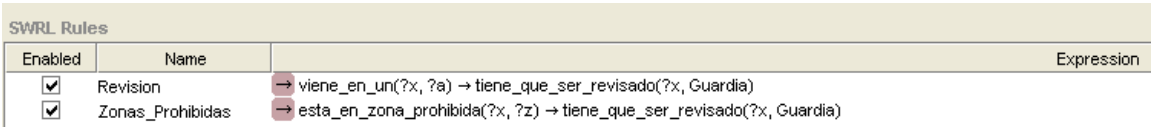


Figura 14: Reglas creadas en Protégé modificando a la ontología CARETAKER

Como la estructura de la ontología es muy grande para ponerla en funcionamiento en un escenario pequeño, se optó por la búsqueda de otra ontología más básica y poder representar a lo que se desea llegar, como es, generar alarmas en las cuales se pueda revisar a una persona cuando ésta esté en una zona prohibida o revisar a un objeto cuando este haya sido detectado.

2.5 PERSONALIZACIÓN DE LA ONTOLOGÍA PROPUESTA POR JADE

Jade propone una ontología para interactuar con agentes, estructurada en tres elementos, conceptos, predicados y acciones, esta esquematización ya se da en otro nivel, CARETAKER deja como enseñanza la forma de representar a una

ontología, en cambio en Jade se buscará utilizar una ontología para que interactúe con los agentes.

La ontología inicial propuesta por Jade, está relacionada con una central de bomberos, como se puede observar en la figura 15.

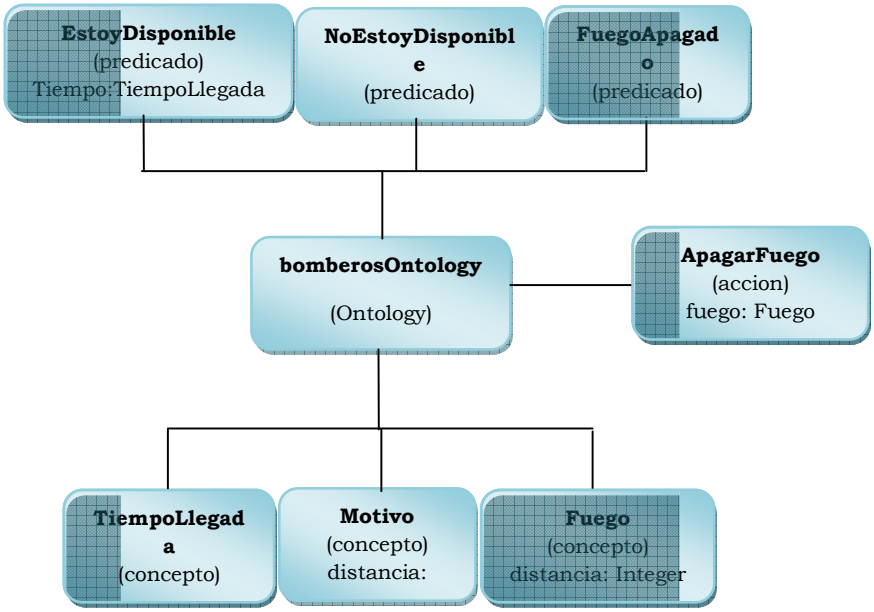


Figura 15: Diagrama inicial de la Ontología Bomberos, propuesta por Jade.

De igual manera que en CARETAKER, se puede obtener el código fuente y realizar cualquier modificación hasta llegar a lo que se desea.

Comparando las dos ontologías, se decidió personalizar a la ontología de la Central de Bomberos, propuesta por Jade, ya que se la puede modelar más fácilmente y sobre todo al trabajar con Jade se modificarán los agentes que interactúan con esta ontología.

Y con la metodología METHONTOLOGY se realizan los pasos necesarios hasta llegar a la ontología final, el proceso se lo detallará a continuación.

2.6 OBTENCIÓN DE LA ONTOLOGÍA FINAL

2.6.1 ESPECIFICACIÓN

El objetivo principal del proyecto de tesis, es identificar mediante ontologías, situaciones de inseguridad en base al comportamiento humano a partir del resultado de procesamiento de un sistema de visión artificial.

Inicialmente se trabajó personalizando a CARETAKER, como se vio en la sección 3.2, la cual es capaz de evaluar automáticamente escenas, identificando a personas, grupo de personas, pero su estructura es bastante completa, para poder realizar cambios en ella y acoplarlos.

Entonces, se analizó la ontología propuesta por Jade que permite también personalizar agentes para que interactúen con estas. Internamente se utilizará el método de re-ingeniería en donde con base a un esquema inicial, se realizan cambios y se obtiene el esquema final de la ontología, claro está que, cada cambio que se haga, se evaluará a fin de cumplir con el objetivo principal y si no se cumple se volverá a hacer los cambios.

2.6.2 CONCEPTUALIZACIÓN

En esta actividad, se determinar el esquema inicial en el cual se empezará a reutilizar. Y es ésta otra de las actividades necesarias para la delimitación del proyecto.

Al hablar de comportamiento humano, se tienen que tomar en cuenta muchos aspectos que inclusive para la visión humana, son complejos, por ejemplo:

- “persona caminando”, se tiene que analizar el rostro de la misma para ver si es sospechosa o no (nervioso, feliz, triste, etc.), ahora,

dependiendo del lugar donde se encuentra, se podría decir que está en una zona prohibida o no y se necesita que se analice esa situación.

- "persona tomando un objeto", aquí se analizaría que tipo de objeto tiene en la mano, un vaso, un arma, una cartera, etc. porque, si esta en un banco, y la persona que se observa no es guardia y tiene un arma en la mano, sería una señal de peligro en ese lugar.
- "objeto olvidado", cuando una persona deja un objeto en un lugar y se retira, aquí en cambio, no se puede saber si fue dejado o no apropósito es objeto, porque si nó sería un "objeto abandonado", en el caso de que dejaran abandonada una bomba dentro de una mochila en un aeropuerto, se generará un peligro, etc.

La idea es que una máquina, pueda hacer estas inferencias de forma automática, de tal manera que se tenga un observador las 24 horas del día, con la capacidad de generar una alarma, cuando ocurran estos acontecimientos.

Es por esto que se el alcance del tema de tesis es que se caracterizará a una persona y objeto, elementos suficientes para el propósito general, en el cual se generará dos alarmas: Revisar a la persona que fue observada en cierto lugar, deduciendo la ontología, que está en una zona prohibida y de igual manera con un objeto.

Como ya se dijo anteriormente, CARETAKER no permite pasar sus clases ontológicas a clases java, entonces se optó por personalizar la ontología propuesta por Jade como se observa en la figura 14. Esta ontología fue creada para generar alarmas en una Central de Bomberos, en la cual se pide apagar el fuego detectado, estructurada mediante clases java, la misma que contiene 3 conceptos, 3 predicados y una acción, como se detalla a continuación:

Conceptos:

- ✓ **Fuego**, que contiene un atributo de tipo Integer que guarda la distancia a la que está el fuego.
- ✓ **Motivo**, que contiene un atributo de tipo String donde se especifica el motivo.
- ✓ **TiempoLlegada**, que contiene un atributo de tipo Integer que representa el tiempo en minutos.

Predicados:

- ✓ **EstoyDisponible**, que indica el hecho de estar disponible y que contiene el concepto Tiempo que indica el tiempo que tardará en llegar.
- ✓ **NoEstoyDisponible**, indica el hecho de no estar disponible y contiene el concepto Motivo donde se indica el motivo por el cual no se puede apagar el fuego.
- ✓ **FuegoApagado**, que indica el hecho de haber apagado el fuego.

Acción:

- ✓ **ApagarFuego**, que solicita al agente que vaya a apagar un determinado fuego

Una vez personalizada la ontología anterior, se obtiene la nueva como se puede observar en la figura 16, la misma que genera dos alarmas: persona

detectada en zona prohibida por lo tanto se debe revisar y objeto detectado se debe revisar, con las siguientes características:

Conceptos:

- ✓ **Persona**, que contiene un atributo de tipo string donde se ingresa a persona detectada.
- ✓ **Objeto**, que contiene un atributo de tipo string donde se ingresa a objeto detectado.

Predicados:

- ✓ **PersonaRevisada,** contiene el concepto Persona e indica que la persona detectada ya fue revisada.
- ✓ **ObjetoRevisado,** contiene el concepto Objeto e indica que el objeto identificado fue revisado.

Acción:

- ✓ **RevisarPersona,** que solicita al agente que vaya a revisar a la persona detectada
- ✓ **RevisarObjeto,** de igual manera solicita al agente que vaya a revisar si el objeto fue revisado.

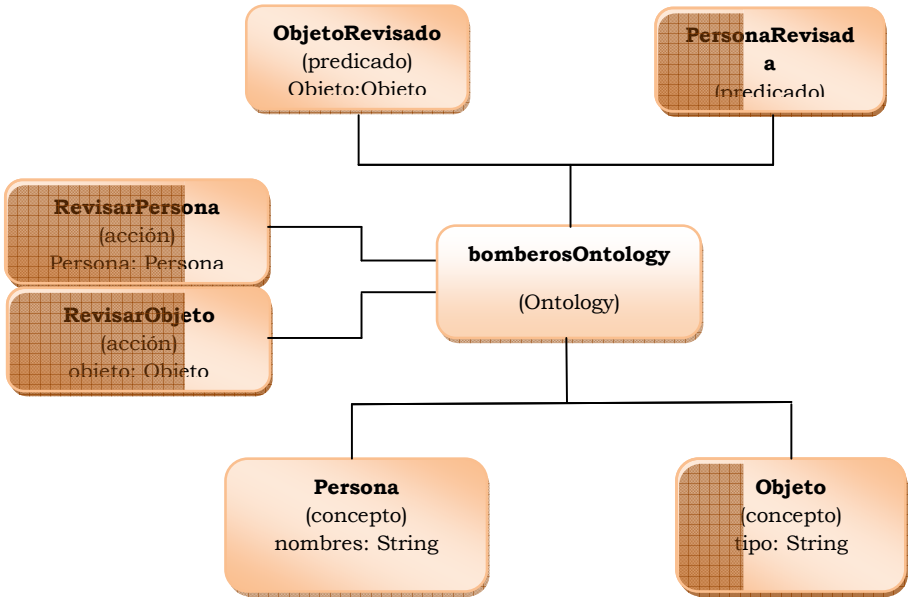


Figura 16: Diagrama final de la Ontología.

Ahora, se detalla los cambios realizados.

Primeramente, con este esquema, las clases para cada elemento de la ontología debe ser creada una clase java apropiada, es decir, dependiendo de si es un concepto, predicado o acción, cada uno de estos se instancia de las clases *jade.content.Concept, jade.content.Predicate y jade.content.AgentAction* respectivamente. En el Anexo C se puede observar algunas características de la instalación de Jade.

Dependiendo de tamaño de la ontología a desarrollar, se puede programar en un texto plano si es pequeña, o si es muy extensa, se lo puede hacer mediante Protégé, que lo hace gráficamente y al final genera el código en clases java. Para mayor detalle véase el Anexo B.

Para cumplir con el objetivo antes mencionado, al esquema inicia, se le incrementó: dos conceptos Persona y Objeto, dos predicados

PersonaRevisada y Objeto Revisado y dos acciones RevisarPersona y RevisarObjeto, como se muestra:

Conceptos: Persona y Objeto.

Persona.java

```
package bomberosOntology;

import jade.content.Concept;

public class Persona implements Concept {

	private String nombres;

	public String getNombres() {
	  return nombres;
	}

	public void setNombres(String n) {
	  nombres = n;
	}
}
```

Objeto.java

```
package bomberosOntology;

import jade.content.Concept;

public class Objeto implements Concept {

	private String tipo;

	public String getTipo() {
	  return tipo;
	}

	public void setTipo(String t) {
	  tipo = t;
	}
}
```

Predicados: PersonaRevisada y ObjetoRevisado

PersonaRevisada.java

```
package bomberosOntology;

import jade.content.Predicate;

public class PersonaRevisada implements Predicate {

	private Persona persona;

	public Persona getPersona() {
		return persona;
	}

	public void setPersona(Persona p) {
		persona = p; } }
```

ObjetoRevisado.java

```
package bomberosOntology;

import jade.content.Predicate;

public class ObjetoRevisado implements Predicate {

	private Objeto objeto;

	public Objeto getObjeto() {
		return objeto;
	}

	public void setObjeto(Objeto o) {
		objeto = o;
	}
}
```

Acciones: RevisarPersona y RevisarObjeto

RevisarPersona.java

```
package bomberosOntology;

import jade.content.AgentAction;

public class RevisarPersona implements AgentAction {

	private Persona persona;

	public Persona getPersona() {
		return persona;
	}

	public void setPersona(Persona p) {
		persona = p;
	} }
```

RevisarObjeto.java

```
package bomberosOntology;

import jade.content.AgentAction;

public class RevisarObjeto implements AgentAction {

	private Objeto objeto;

	public Objeto getObjeto() {
		return objeto;
	}

	public void setObjeto(Objeto o) {
		objeto = o;
	}
}
```

Luego en la clase principal BomberosOntology se incrementó al código fuente, las clases anteriormente creadas, de la siguiente manera:

BomberosOntology.java

```
//Vcabulario de los conceptos
public static final String PERSONA_ = "Persona";
public static final String PERSONA_PERSONA = "nombres";

public static final String OBJETO_ = "Objeto";
public static final String OBJETO_OBJETO = "tipo";


// Vocabulario para los predicados

public static final String PERSONA_REVISADA_ = "PersonaRevisada";
public static final String PERSONA_REVISADA_PERSONA = "persona";

public static final String OBJETO_REVISADO_ = "ObjetoRevisado";
public static final String OBJETO_REVISADO_OBJETO = "objeto";


// Vocabulario para las acciones
public static final String REVISAR_PERSONA_ = "RevisarPersona";
public static final String REVISAR_PERSONA_REVISAR = "persona";
 public static final String REVISAR_OBJETO_ = "RevisarObjeto";
 public static final String REVISAR_OBJETO_REVISAR = "objeto";



// BomberosOntolgy extiende la ontología básica
super(ONTOLOGY_NAME, base);

try {
		// Añade a la ontologia los conceptos, los predicados y las acciones junto con sus
		//clases asociadas
		add(new ConceptSchema(PERSONA_), Persona.class);
```

```
add(new ConceptSchema(OBJETO_), Objeto.class);

add(new PredicateSchema(PERSONA_REVISADA_), PersonaRevisada.class);
add(new PredicateSchema(OBJETO_REVISADO_), ObjetoRevisado.class);

add(new AgentActionSchema(REVISAR_PERSONA_), RevisarPersona.class);
add(new AgentActionSchema(REVISAR_OBJETO_), RevisarObjeto.class);

//-------------------------------------------------------------------------
//-------------------------------------------------------------------------

// Estructura del esquema para el concepto PERSONA_
ConceptSchema persona = (ConceptSchema) getSchema(PERSONA_);
persona.add(PERSONA_PERSONA, (PrimitiveSchema)
getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

// Estructura del esquema para el concepto OBJETO_
ConceptSchema objeto = (ConceptSchema) getSchema(OBJETO_);
objeto.add(OBJETO_OBJETO, (PrimitiveSchema)
getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);

//-----------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------


// Estructura del esquema para el predicado PERSONA_REVISADA
PredicateSchema personaRevisada = (PredicateSchema)
getSchema(PERSONA_REVISADA_);
personaRevisada.add(PERSONA_REVISADA_PERSONA, (ConceptSchema)
getSchema(PERSONA_),  ObjectSchema.MANDATORY);

// Estructura del esquema para el predicado OBJETO_REVISADO
PredicateSchema objetoRevisado = (PredicateSchema)
getSchema(OBJETO_REVISADO_);
objetoRevisado.add(OBJETO_REVISADO_OBJETO, (ConceptSchema)
getSchema(OBJETO_), ObjectSchema.MANDATORY);

//-----------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------

// Estructura del esquema para la acción REVISAR_PERSONA
AgentActionSchema revisarPersona = (AgentActionSchema)
getSchema(REVISAR_PERSONA_);
revisarPersona.add(REVISAR_PERSONA_REVISAR, (ConceptSchema)
getSchema(PERSONA_));

// Estructura del esquema para la acción REVISAR_OBJETO

AgentActionSchema revisarObjeto = (AgentActionSchema)
getSchema(REVISAR_OBJETO_);
revisarObjeto.add(REVISAR_OBJETO_REVISAR, (ConceptSchema)
getSchema(OBJETO_));
```

Luego, se modificó los agentes ya existentes, Alarmado y CentralBomberos de tal manera que se comuniquen con las clases adicionadas.

Con la idea de que en un futuro se pueda realizar pruebas con esta ontología para que interactúe con otra aplicación que reconozca patrones, se modificó de tal manera que los agentes lea un archivo en el cual se encuentra la devolución de que una persona u objeto ha sido observado sospechosamente (persona u objeto), adhiriéndole el siguiente código:

Alarmado.java

```
    try {

      // Apertura del fichero y creacion de BufferedReader para poder
      // hacer una lectura comoda (disponer del metodo readLine()).
      archivo = new File ("C:\\archivo.txt");
      fr = new FileReader (archivo);
      br = new BufferedReader(fr);

      // Lectura del fichero

      while((linea=br.readLine())!=null)
        linea2 = linea;
        linea = linea2;

    } catch(Exception e){
           e.printStackTrace();
    } finally{
      // En el finally se cierra el fichero, para asegurar
      // que se cierra tanto, si todo va bien como si salta
      // una excepcion.

      try{
        if( null != fr ){
          fr.close();
        }
      }catch (Exception e2){
        e2.printStackTrace();
      }
    }

.
.
            // Crea el concepto Persona
           if (linea.equals("Persona")) {
               Persona p = new Persona();
              p.setNombres(linea);


            // Crea la accion RevisarPersona
            RevisarPersona rp = new RevisarPersona();
            rp.setPersona(p);
            Action a = new Action(getAID(), rp);
                      .
```

```
                                .
            // Crea la accion RevisarObjeto
               RevisarObjeto rp = new RevisarObjeto();
               rp.setObjeto(o);
               Action a = new Action(getAID(), rp);
            .
            .
            .
                  // Decodifica el mensaje ACL recibido mediante el lenguaje de contenido y la ontologia
                  actual

                  if (ce instanceof PersonaRevisada){
                      // Recibido un INFORM con contenido correcto
                      PersonaRevisada pr = (PersonaRevisada) ce; // Transforma el contenido en el
                      objeto predicado FuegoApagado de la ontologia

                      System.out.println("Central de seguridad " + inform.getSender().getName() + "
                      informa que ha revisado a la persona "+pr.getPersona().getNombres());
                  .
                  .
                  .
                  if (ce instanceof ObjetoRevisado){
                      // Recibido un INFORM con contenido correcto
                      ObjetoRevisado or = (ObjetoRevisado) ce; // Transforma el contenido en el objeto
                      predicado FuegoApagado de la ontologia

                      System.out.println("Central de seguridad " + inform.getSender().getName() + "
                      informa que ha revisado el objeto "+or.getObjeto().getTipo());
                      .
                      .
                      .
```

CentralBomberos.java

```
.
..
       // Apertura del fichero y creacion de BufferedReader para poder
       // hacer una lectura comoda (disponer del metodo readLine()).
       archivo = new File ("C:\\archivo.txt");
       fr = new FileReader (archivo);
       br = new BufferedReader(fr);

       // Lectura del fichero

       while((linea=br.readLine())!=null)
         linea2 = linea;
         linea = linea2;

  }
     catch(Exception e){
       e.printStackTrace();
     }finally{
       // En el finally se cierra el fichero, para asegurar
       // que se cierra tanto, si todo va bien como si salta
       // una excepcion.
       try{
         if( null != fr ){
           fr.close();
         }
       }catch (Exception e2){
         e2.printStackTrace();
       } }
```

```
.
..
        if (linea.equals("Persona")){
          RevisarPersona ce = (RevisarPersona) a.getAction();
                    if (ce instanceof RevisarPersona){
             // Recibido un INFORM con contenido correcto
             RevisarPersona rp = (RevisarPersona) ce;
             String nombres=rp.getPersona().getNombres();
              System.out.println("Central "+getLocalName()+": Hemos recibido una llamada de " +
              request.getSender().getName() + " diciendo que ha visto  a "+nombres);
              System.out.println("Central "+getLocalName()+": Salimos corriendo");

             TiempoLlegada tl = new TiempoLlegada();
             tl.setTiempo((int)(Math.random()*10));
             EstoyDisponible ed = new EstoyDisponible();
             ed.setTiempo(tl);

             getContentManager().fillContent(agree, ed);

           }else throw new NotUnderstoodException("Central de bomberos alemana, no entiendo el
             mensaje.");

         }

        if (linea.equals("Objeto")){
          RevisarObjeto ce = (RevisarObjeto) a.getAction();
                    if (ce instanceof RevisarObjeto){
                        // Recibido un INFORM con contenido correcto
                        RevisarObjeto ro = (RevisarObjeto) ce;
                        String tipo=ro.getObjeto().getTipo();
                        System.out.println("Central "+getLocalName()+": Hemos recibido una llamada
                        de " + request.getSender().getName() + " diciendo que ha visto  a "+tipo);
                        System.out.println("Central "+getLocalName()+": Salimos corriendo");


                    TiempoLlegada tl = new TiempoLlegada();
                    tl.setTiempo((int)(Math.random()*10));
                    EstoyDisponible ed = new EstoyDisponible();
                    ed.setTiempo(tl);

                    getContentManager().fillContent(agree, ed);
           }else throw new NotUnderstoodException("Central de bomberos alemana, no entiendo el
             mensaje.");

         }
```

Una vez realizados todos estos cambios, se obtiene el esquema final de la ontología. En la figura 17 se observa el diagrama conceptual instanciado desde Protégé.

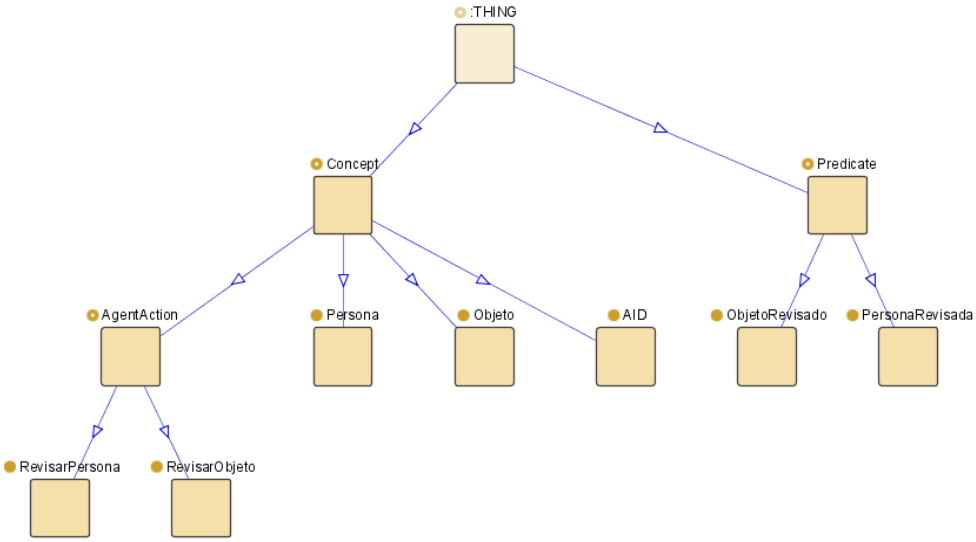


Figura 17: Diagrama conceptual instanciado en Protégé.

2.6.3 FORMALIZACION, IMPLEMENTACION Y MANTENIMIENTO

Estas son las últimas actividades presentadas por METHONTOLOGY, la formalización, se encuentra ya implícita, en los cambios realizados en cada prototipo.

Como ya se mencionó, la evaluación es una actividad muy importante y es por esto que en la sección siguiente se detallará los resultados obtenidos y los criterios de evaluación en cada cambio realizado.

En cuanto al mantenimiento, es una puerta abierta para futuras investigaciones, de tal manera que la amplíen a la ontología y pueda interactuar con otras aplicaciones que reconozcan patrones y así obtener un sistema de vigilancia completo.

2.6.4 DIAGRAMA PARA GENERAR ALARMAS

Como ya se especificó inicialmente, la idea es generar dos alarmas, indicando si una persona se encuentra en una zona prohibida y un objeto fue olvidado en cierto lugar, con la ayuda de dos agentes (CentralBomberos y Alarmado) y la ontología final, el esquema general en el cual se observa como los agentes interactúan con la ontología se puede ver en la figura 18, Además en la figura 19 y 20 se detalla dos casos de uso generales, para observar esta interacción.

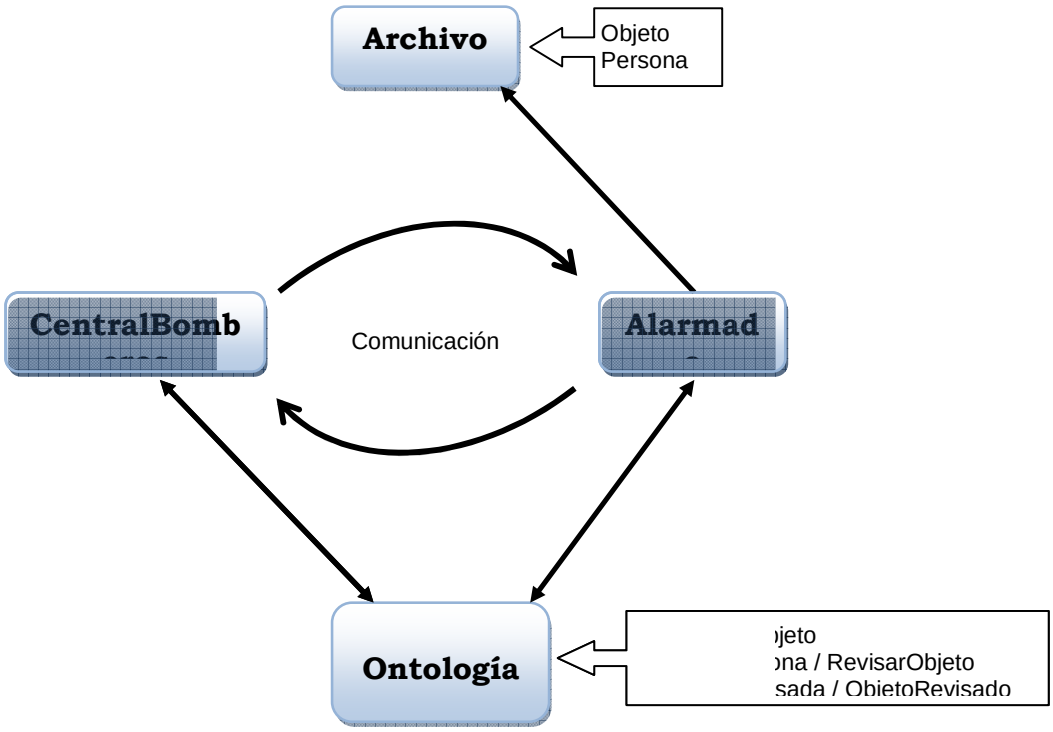


Figura 18: Diagrama de la interacción de los agentes con la ontología final.

Caso de Uso: Funcionamiento General

El diagrama de casos de uso general, se puede observar en la figura 18 y en la tabla 2, se describe la secuencia de este funcionamiento.

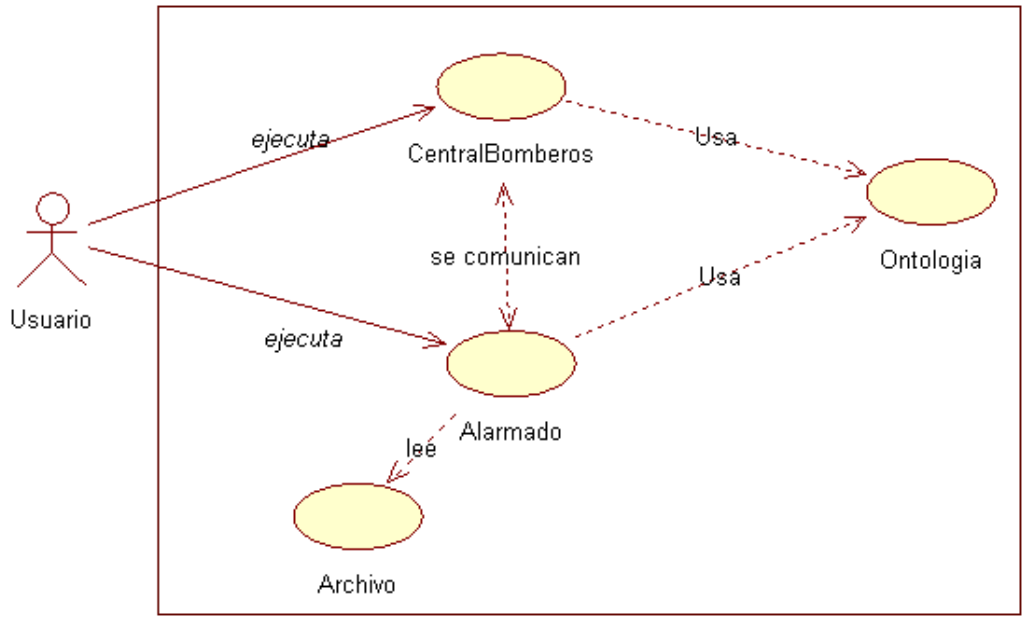


Figura 19: Diagrama de casos de uso, demostrando la ejecución de los agentes.

USUARIO	CENTRALBOMBEROS	ALARMADO	ONTOLOGÍA
1. EJECUTA AGENTE CENTRALBOMBEROS CON EL SIGUIENTE CÓDIGO: *JAVA JADE.BOOT –CONTAINER C1:CENTRALBOMBEROS C2:CENTRALBOMBEROS C3:CENTRALBOMBEROS C4:CENTRALBOMBEROS*			
	2. DEVUELVE: PENDIENTE DE ALARMA		
3. EJECUTA AGENTE ALARMADO CON EL SIGUIENTE CÓDIGO: *JAVA JADE.BOOT –CONTAINER ALARMADO:ALARMADO(C1 C2 C3 C4)*			

		4. LEE ARCHIVO	
		5. INFORMA A LA ONTOLOGÍA SI HA VISTO A PERSONA U OBJETO	
			6. REVISA REGLAS Y DEVUELVE RESULTADOS
	8. COMUNICACIÓN CON ALARMADO BASADO EN LAS RESPUESTAS DE LA ONTOLOGÍA	7. COMUNICACIÓN CON CENTRALBOMBEROS BASADO EN LAS RESPUESTA DE LA ONTOLOGÌA	

Tabla 2: Secuencia para ejecución de los agentes CentralBomberos y Alarmado

Caso de uso: Comunicación entre agentes

Este es el diagrama de casos de la comunicación entre los angentes se puede observar en la figura 19 y de igual manera el la tabla 3, se detalla la secuencia de esta.

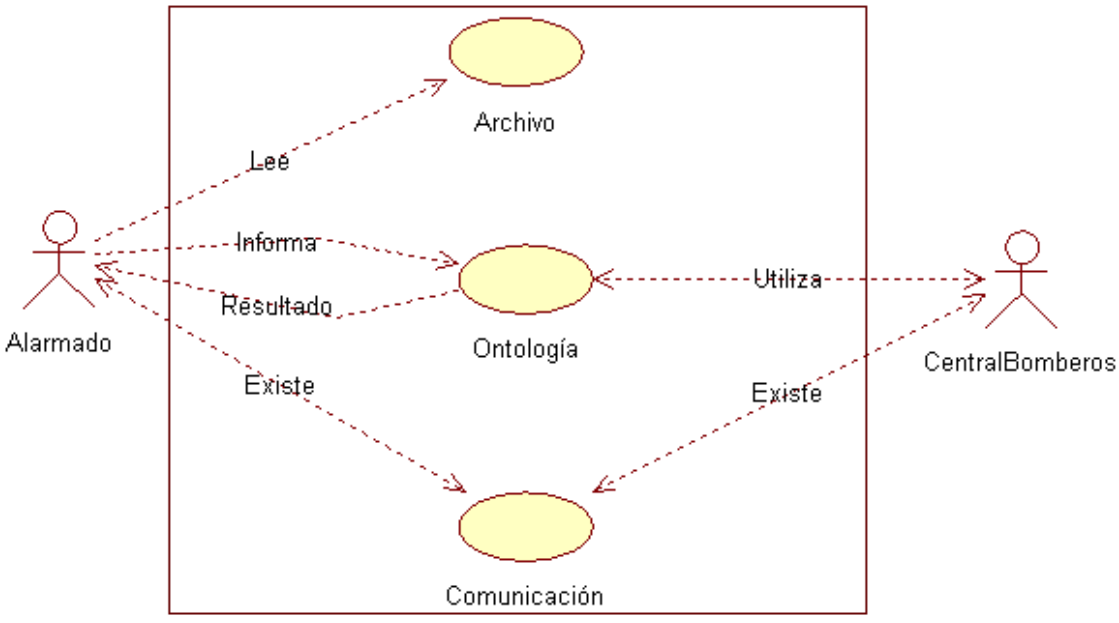


Figura 20: Diagrama de casos de uso de la comunicación entre los agentes CentrslBomberos y Alarmado

ALARMADO	ARCHIVO	ONTOLOGÌA	CENTRALBOMBEROS
1. LEE ARCHIVO			
	2. DEVUELVE A PERSONA U OBJETO		
		3. DEPENDIENDO DE LO LEIDO, REVISA LAS REGLAS Y DEVUELVE RESULTADOS	
5. SE COMUNICA CON CENTRALBOMBEROS, BASADO EN LOS RESULTADOS DE LA ONTOLOGÍA			4. SE COMUNICA CON ALARMADO, BASADO EN LOS RESULTADOS DE LA ONTOLOGÍA

Tabla 3: Secuencia de la comunicación entre los agentes CentralBomberos y Alarmado

2.7 PERSONALIZACIÓN Y VALIDACIÓN DE LA ONTOLOGÍA

Las pruebas fueron realizadas al final de cada una de las versiones de la ontología personalizadas, con la finalidad de poder validar su correcto funcionamiento y en base a los resultados obtenidos, modificarlas hasta llegar al modelo final; el cual consiste en que los agentes encargados de automatizar la ontología se comuniquen de tal manera que permita que ésta pueda determinar si una persona u objeto fue revisado exitosamente.

Los cambios realizados en cada una de las versiones se detallan a continuación, iniciando con el ejemplo de Central de bomberos que da el manual de programación de Jade. En el Anexo D se encuentra detallada la forma de hacer correr estos agentes, mediante JADE.

Los criterios que determinaron el correcto funcionamiento de la ontología fueron:

- El correcto acoplamiento de nuevas clases denominada “concepto, predicados y acciones”, que me permite caracterizar a personas y objetos.

- Lectura correcta de un archivo de texto por parte de los agentes.
- El correcto funcionamiento de la ontología.

2.7.1 ONTOLOGÍA DE BOMBEROS INICIAL

Resultados que se obtienen al hacer correr el diagrama inicial (prototipo inicial).

Agente Central Bomberos

```
----------------------------------------
Central c4: Pendiente de alarmas...
Central c1: Pendiente de alarmas...
Central c3: Pendiente de alarmas...
Central c2: Pendiente de alarmas...
```

Agente Alarmado

```
----------------------------------------
Solicitando ayuda a varias centrales de bomberos...
Central de bomberos c3@DellUser-PC:1099/JADE informa que NO puede apagar el
fuego. Motivo: Fuego demasiado lejos.
Central de bomberos c4@DellUser-PC:1099/JADE informa que puede apagar el
fuego. Tardar├í en llegar 4 min
Central de bomberos c2@DellUser-PC:1099/JADE informa que NO puede apagar el
fuego. Motivo: Fuego demasiado lejos.
Central de bomberos c1@DellUser-PC:1099/JADE informa que puede apagar el
fuego. Tardar├í en llegar 8 min
```

Agente Central Bomberos

```
----------------------------------------
Central c3: Hemos recibido una llamada de alarmado@DellUser-PC:1099/JADE
diciendo que ha visto fuego a 5 Km.
Central c3: Fuego demasiado lejos. Pasamos!!
Central c1: Hemos recibido una llamada de alarmado@DellUser-PC:1099/JADE
diciendo que ha visto fuego a 5 Km.
Central c1: Salimos corriendo
Central c2: Hemos recibido una llamada de alarmado@DellUser-PC:1099/JADE
diciendo que ha visto fuego a 5 Km.
Central c2: Fuego demasiado lejos. Pasamos!!
Central c4: Hemos recibido una llamada de alarmado@DellUser-PC:1099/JADE
diciendo que ha visto fuego a 5 Km.
Central c4: Salimos corriendo
Central c4: Hemos vuelto de apagar el fuego.
Central c1: Hemos vuelto de apagar el fuego.
```

En este ejemplo se puede observar como los dos agentes interactúan, simulando una conversación entre dos personas, con la idea de solucionar el problema de apagar el fuego, comunicando su disponibilidad y si se solucionó el problema.

2.7.2 MODIFICACIONES

- Lograr que la ontología funcione correctamente luego de incrementarle nuevas clases (persona y objeto). Obteniendo buenos resultados como se observa a continuación:

Agente Central Bomberos

```
-----------------------------------------
Central c4: Pendiente de alarmas...
Central c2: Pendiente de alarmas...
Central c3: Pendiente de alarmas...
Central c1: Pendiente de alarmas...
Central c4: Hemos recibido una llamada de alarmado@DellUser-
PC:1099/JADE diciendo que ha visto  a Objeto
Central c4: Salimos corriendo
Central c2: Hemos recibido una llamada de alarmado@DellUser-
PC:1099/JADE diciendo que ha visto  a Objeto
Central c2: Salimos corriendo
Central c3: Hemos recibido una llamada de alarmado@DellUser-
PC:1099/JADE diciendo que ha visto  a Objeto
Central c3: Salimos corriendo
Central c1: Hemos recibido una llamada de alarmado@DellUser-
PC:1099/JADE diciendo que ha visto  a Objeto
Central c1: Salimos corriendo
Central de seguridad c2: Hemos vuelto de realizar la tarea.
Central de seguridad c1: Hemos vuelto de realizar la tarea.
Central de seguridad c3: Hemos vuelto de realizar la tarea.
Central de seguridad c4: Hemos vuelto de realizar la tarea.
```

Agente Alarmado

```
Solicitando ayuda a varias centrales de seguridad...
-------------------------------------------

Central de seguridad c3@DellUser-PC:1099/JADE informa que puede
realizar la tarea. Tardar├í en llegar 6 min
Central de seguridad c2@DellUser-PC:1099/JADE informa que puede
realizar la tarea. Tardar├í en llegar 4 min
Central de seguridad c1@DellUser-PC:1099/JADE informa que puede
realizar la tarea. Tardar├í en llegar 3 min
Central de seguridad c4@DellUser-PC:1099/JADE informa que puede
realizar la tarea. Tardar├í en llegar 1 min
```

- Conseguir que los agentes puedan leer un archivo y no cambie el funcionamiento normal de la ontología. Con la finalidad de que en un futuro pueda interactuar con una aplicación inicial (reconocimiento de patrones), en el cual le devuelva un archivo plano que diga que una persona u objeto ha sido reconocido. Obteniendo resultados positivos como se muestra a continuación, pero solo con el agente alarmado:

```
-------------------------------------------
Central c2: Pendiente de alarmas...
Central c4: Pendiente de alarmas...
Central c3: Pendiente de alarmas...
Central c1: Pendiente de alarmas...

-------------------------------------------
Central c3: Hemos recibido una llamada de alarmado@DellUser-
PC:1099/JADE diciendo que ha visto  a Desconocido
Central c2: Hemos recibido una llamada de alarmado@DellUser-
PC:1099/JADE diciendo que ha visto  a Desconocido
Central c4: Hemos recibido una llamada de alarmado@DellUser-
PC:1099/JADE diciendo que ha visto  a Desconocido
Central c1: Hemos recibido una llamada de alarmado@DellUser-
PC:1099/JADE diciendo que ha visto  a Desconocido
```

- De igual manera se busca que el agente el otro agente CentralBomberos pueda leer un archivo. Obteniendo resultados favorables y los cambios efectuados son aceptados. Pero hasta el

momento no devuelve las alarmas propuestas inicialmente, por lo que se sigue realizando los cambios.

Agente Central Bomberos

```
-----------------------------------------
Central c1: Pendiente de alarmas...
Central c2: Pendiente de alarmas...
Central c4: Pendiente de alarmas...
Central c3: Pendiente de alarmas...
Central c4: Hemos recibido una llamada de alarmado@DellUser-
PC:1099/JADE diciendo que ha visto  a Objeto
Central c4: Salimos corriendo
Central c3: Hemos recibido una llamada de alarmado@DellUser-
PC:1099/JADE diciendo que ha visto  a Objeto
Central c3: Salimos corriendo
Central c2: Hemos recibido una llamada de alarmado@DellUser-
PC:1099/JADE diciendo que ha visto  a Objeto
Central c2: Salimos corriendo
Central c1: Hemos recibido una llamada de alarmado@DellUser-
PC:1099/JADE diciendo que ha visto  a Objeto
Central c1: Salimos corriendo
Central de seguridad c4: Hemos vuelto de realizar la tarea.
Central de seguridad c2: Hemos vuelto de realizar la tarea.
Central de seguridad c3: Hemos vuelto de realizar la tarea.
Central de seguridad c1: Hemos vuelto de realizar la tarea.
```

Agente Alarmado

```
-----------------------------------------
Solicitando ayuda a varias centrales de seguridad...
Central de seguridad c4@DellUser-PC:1099/JADE informa que puede
realizar la tarea. Tardar├í en llegar 3 min
Central de seguridad c3@DellUser-PC:1099/JADE informa que puede
realizar la tarea. Tardar├í en llegar 1 min
Central de seguridad c2@DellUser-PC:1099/JADE informa que puede
realizar la tarea. Tardar├í en llegar 2 min
Central de seguridad c1@DellUser-PC:1099/JADE informa que puede
realizar la tarea. Tardar├í en llegar 4 min
```

- Funcionamiento normal de la ontología incrementando todas las clases necesarias para validar a una persona. Es por esto que se incremento a la ontología, el predicado PersonaRevisada y la acción RevisarPersona. Obteniendo exitosamente los siguientes resultados:

Agente Central Bomberos

```
-----------------------------------------
Central c4: Pendiente de alarmas...
Central c1: Pendiente de alarmas...
Central c2: Pendiente de alarmas...
Central c3: Pendiente de alarmas...
Central c1: Hemos recibido una llamada de alarmado@DellUser-
PC:1099/JADE diciendo que ha visto  a Desconocido
Central c1: Salimos corriendo
Central c2: Hemos recibido una llamada de alarmado@DellUser-
PC:1099/JADE diciendo que ha visto  a Desconocido
Central c4: Hemos recibido una llamada de alarmado@DellUser-
PC:1099/JADE diciendo que ha visto  a Desconocido
Central c4: Salimos corriendo
Central c3: Hemos recibido una llamada de alarmado@DellUser-
PC:1099/JADE diciendo que ha visto  a Desconocido
Central c3: Salimos corriendo
Central de seguridad c4: Hemos vuelto de revisar a la persona.
Central c2: Salimos corriendo
Central de seguridad c3: Hemos vuelto de revisar a la persona.
Central de seguridad c2: Hemos vuelto de revisar a la persona.
Central de seguridad c1: Hemos vuelto de revisar a la persona.
```

Agente Alarmado

```
-----------------------------------------
Solicitando ayuda a varias centrales de seguridad...
Central de seguridad c1@DellUser-PC:1099/JADE informa que puede
revisar a la persona. Tardar├í en llegar 0 min
Central de seguridad c4@DellUser-PC:1099/JADE informa que puede
revisar a la persona. Tardar├í en llegar 6 min
Central de seguridad c3@DellUser-PC:1099/JADE informa que puede
revisar a la persona. Tardar├í en llegar 7 min
Central de seguridad c2@DellUser-PC:1099/JADE informa que puede
revisar a la persona. Tardar├í en llegar 3 min
```

2.7.3 ONTOLOGÍA FINAL

- Lograr el funcionamiento total de la ontología ya sea que reciba como entrada a una persona o un objeto. Entonces, se incrementó el predicado ObjetoRevisado y la acción RevisarObjeto y así completar la ontología. Obteniendo los siguientes resultados positivos:

Agente Central Bomberos

```
-----------------------------------------
Central c4: Pendiente de alarmas...
Central c3: Pendiente de alarmas...
Central c2: Pendiente de alarmas...
Central c1: Pendiente de alarmas...
Central c1: Hemos recibido una llamada de alarmado@DellUser-
PC:1099/JADE diciendo que ha visto  a Objeto
Central c2: Hemos recibido una llamada de alarmado@DellUser-
PC:1099/JADE diciendo que ha visto  a Objeto
Central c2: Salimos corriendo
Central c4: Hemos recibido una llamada de alarmado@DellUser-
PC:1099/JADE diciendo que ha visto  a Objeto
Central c3: Hemos recibido una llamada de alarmado@DellUser-
PC:1099/JADE diciendo que ha visto  a Objeto
Central c3: Salimos corriendo
Central c4: Salimos corriendo
Central de seguridad c4: Hemos vuelto de revisar al Objeto.
Central c1: Salimos corriendo
Central de seguridad c3: Hemos vuelto de revisar al Objeto.
Central de seguridad c2: Hemos vuelto de revisar al Objeto.
Central de seguridad c1: Hemos vuelto de revisar al Objeto.
```

Agente Alarmado

```
-----------------------------------------
Solicitando ayuda a varias centrales de seguridad...
Central de seguridad c1@DellUser-PC:1099/JADE informa que puede
revisar al Objeto. Tardar├í en llegar 0 min
Central de seguridad c4@DellUser-PC:1099/JADE informa que puede
revisar al Objeto. Tardar├í en llegar 6 min
Central de seguridad c3@DellUser-PC:1099/JADE informa que puede
revisar al Objeto. Tardar├í en llegar 7 min
Central de seguridad c2@DellUser-PC:1099/JADE informa que puede
revisar al Objeto. Tardar├í en llegar 3 min
```

Estas son las modificaciones realizadas en cuanto a la ontología original y la obtenida para el análisis de persona y objetos.

En la tabla 4, se resume los criterios evaluados.

Criterio de evaluación	Modificaciones	Resultados
La ontología inicial funciona incrementando una nueva clase.	Creación de clase *Persona* y *Objeto*, e incrementarla en la ontología inicial.	La clase persona fue creada con éxito e interactúa normalmente en la ontología
Hacer que los agentes puedan leer un archivo.	Modificación realizar en el código de cada agente.	Los agentes leen el archivo.
Observar como es el funcionamiento con el objeto *Persona* para verificar si los agentes se comunican con la ontología	Se creó un predicado *PersonaRevisada* y una acción *RevisarPersona*, se incrementó estas clases en la ontología y se modificó el código de los agentes.	La simulación es correcta, se observa como los agentes interactúan con la ontología.
Completar la ontología con la clase *Objeto*.	Se creó un predicado *ObjetoRevisado* y una acción *RevisarObjeto*, se incrementó estas clases en la ontología y se modificó el código de los agentes.	Finalmente la ontología devuelve exitosamente los resultados deseados.

Tabla 4: Resultados de la personalización y evaluación.

Lo que hay que tomar en cuenta es que por cada modificación que se vaya realizando, se debe hacer los cambios en la clase principal de la ontología, en cuando a la generación de nuevas clases.

CAPÍTULO III

DISCUSIÓN

3.1 INTRODUCCIÓN

En este capítulo se detallará las decisiones tomadas en cada paso del proceso de la personalización de la ontología y analizando los resultados obtenidos por cada objetivo planteado.

3.2 ONTOLOGÍA

Para obtener el prototipo inicial se investigó dos ontologías: Caretaker que trabaja con la extracción del conocimiento obtenido mediante grandes colecciones multimedia, grabadas a través de las redes de cámaras y micrófonos desplegados en sitios reales y Central de Bomberos en donde la ontología toma decisiones de pedir ayuda a centrales cercanas disponibles, en el caso que se dé un incendio.

El entorno de estudio analizado es en un área de oficina en la cual se restringen el ingreso a ciertas personas y la detección de objetos, de tal manera que se pueda alertar que una persona fue detectada en zona prohibida o que un objeto aparece en cierto lugar.

Por lo tanto se optó por utilizar la ontología Central de Bomberos porque se adapta a la idea de que se esté vigilando un lugar específico y dependiendo de lo que ocurre se genere una alarma.

3.3 METODOLOGÍA

Existen distintas alternativas para poder desarrollar Ontologías, por lo que no existe una metodología correcta para cierto desarrollo, más bien se diría que la metodología es alternativa dependiendo de la aplicación que se va a realizar. Para el presente proyecto se investigó a dos metodologías, como son: Methontology y

On-to-knowlegde pero la que se adapta es la primera porque permite trabajar con reingeniería y prototipos, personalizando una ontología ya desarrollada.

3.4 RESULTADOS

Cumpliendo con los objetivos propuestos para el presente trabajo, se analizará cada uno de ellos y los resultados obtenidos.

El comportamiento humano es muy difícil de inferir a una persona, mucho más para una computadora, es por eso que la ontología final se la delimitó de tal manera que genere dos alarmas: revisar a la persona que se encuentra en zona prohibida y revisara a un objeto que fue detectado.

La ontología final consiste en dos conceptos: persona y objeto, dos predicados PersonaRevisada y objetoRevisado y dos acciones RevisarPersona y RevisarObjeto, esto en cuanto a la ontología, sin embargo y para poder observar el funcionamiento de la misma, se optó por modificar a dos agentes prediseñados, como son: Alarmado y CentralBomberos propuestos en el mismo ejemplo de Jade.

3.5 FUTURAS INVESTIGACIONES

En base a lo investigado puedo concluir, que un sistema completo de vigilancia basado en ontologías, se resumiría en tres fases, como se ve en la figura 21:

- Desarrollo de un sistema de visualización artificial: Con la finalidad de observar y analizar en forma automática, el entorno a vigilar, mediante cámaras de seguridad y procesamiento necesario para el reconocimiento de patrones.

CONCLUSIONES

- Elaboración de una ontología en la que se conceptualice el entorno, es decir, se defina a los elementos que intervienen en el mismo, con sus características, propiedades y acciones.
- Interacción del sistema de Vigilancia con la Ontología. (Por ejemplo: Mediante un archivo)
- Construcción de agentes, capaces de interactuar con la ontología antes estructurada y estimulen a que la ésta tome decisiones en base a lo reconocido en el sistema de visualización artificial y genere una alarmas en tiempo real.

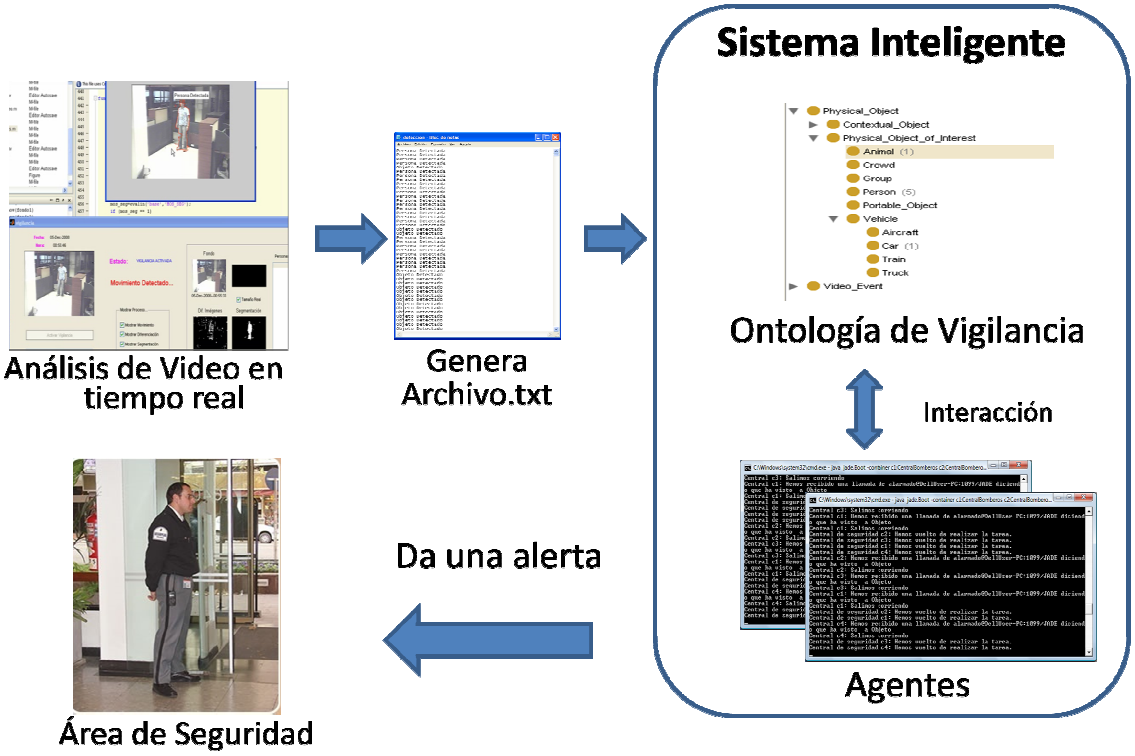


Figura 21: Esquema para un Sistema de Vigilancia en tiempo real.

CONCLUSIONES

- ✓ Como apoyo para la culminación de esta tesis, se investigó a dos metodologías, como son: Methontology y On-to.knowlegde, de las cuales se optó por la primera debido a que esta a demás de trabajar con prototipos, permite hacer reingeniería de las ontologías, utilizando archivos ya desarrolladas y probadas para adaptarlas al entorno de estudio.

- ✓ CARETAKER es una ontología bastante completa, resuelve problemas complejos de información y de conocimiento, como el desarrollo de nuevas herramientas, está probada en el metro de Roma y Turín, además está basada en clases para objetos, personas, multimedia, video y sonido, y es posible instanciar a las clases dependiendo de lo que se va a modelar.

- ✓ La ontología CARETAKER tiene varios componentes que no son necesarios para cumplir con los objetivos del presente trabajo, por lo que se optó por personalizar a la ontología Central de Bomberos que está estructurada mediante clases java y permite trabajar con agentes.

- ✓ La ontología Central de Bomberos, está estructurada mediante tres elementos principales, como son: conceptos, predicados y acciones. Y para cumplir con el objetivo del presente proyecto, se creó dos objetos (persona, objeto), dos acciones (RevisarPersona, RevisarObjeto) y dos predicados (PersonaRevisada, Objeto Revisado), además, se modificó el código de los agentes para que puedan leer un archivo de tal manera que la ontología genere dos alarmas: Persona detectada en zona prohibida, por lo tanto se debe revisar, objeto detectado y de igual manera se debe revisar.

CONCLUSIONES

- ✓ Las modificaciones realizadas a la ontología Central de bomberos: lectura de un archivo, presentación de mensajes y la agregación de clases, objetos y métodos, permite que ésta se adapte como una ontología de seguridad.

- ✓ Predecir el comportamiento de una persona, es una tarea bastante difícil, es por esto que una ontología para poder tomar decisiones en base al comportamiento humano, es necesario que tenga otros parámetros de entrada, como: persona identificada y lugar en donde fue identificada. Con la finalidad de que la ontología resuelva si la persona identificada esta o no en zona prohibida.

RECOMENDACIONES

- ✓ Dependiendo de lo que se va a modelar, se puede utilizar cualquier tipo de metodología, solamente se debe tener claro definir bien el entorno de estudio.

- ✓ En el internet existen varias ontologías ya probadas, que permiten adaptarse a cualquier entorno de estudio.

- ✓ Para iniciar programando a una ontología mediante clases java y con la estructura de conceptos, predicados y acciones, se puede utilizar el proyecto SimpleJADEAbstractOntology.pprj, propuesto por jade, en donde muestra la estructuración en forma de árbol estas clases, lo que permite identificar exactamente como crear a un concepto, predicado o acción.

- ✓ Si se le agregan más objetos y más métodos es posible calcular en el que se demoraría un vigilante a revisar a la persona u objeto detectado y el lugar en donde fue visto.

- ✓ Para poder llegar a reconocer el comportamiento de las personas es necesario estudiar acerca de la antropometría, que será de mucha ayuda para este propósito.

- ✓ Como investigaciones futuras propongo el sistema de vigilancia que se muestra en el capítulo III, que serviría de mucha ayuda para tener un vigilante las 24 horas del día y diagnosticar las situaciones que se presenten, en forma automática.

BIBLIOGRAFÍA

- [Agudo2003] Agudo, B. y Fernandez, R. "Introducción a JADE". Departamento de sistemas Informáticos y Programación. UCM. Marzo del 2003.

- [AgentesJADE] Introducción a los Agentes y JADE. Disponible en: http://pegasus.javeriana.edu.co/~mad/Agentes%20y%20Jade.pdf

- [Bremond2004] Bremond, F., Maillot, N., Thonnat, M. y Vu, V. "Ontologies for Video Events", INRIA. April 2004.

- [Bellifemine2007] Bellifemine, F., Caire, G. y Greenwood, D. "Usando lenguaje de contenido y una ontología". Disponible en: http://agentes.unsl.googlepages.com/trans_cont_onto.pdf. 2007.

- [Caretaker2006a]Publicaciones de los resultados obtenidos al trabajar con CARETAKER. Definición de ontologías y representación del conocimiento. Disponible en: http://sceptre.king.ac.uk/caretaker/deliverables.html

- [Caretaker2006b] Publicaciones de los resultados obtenidos al trabajar con CARETAKER. Análisis del Estado del Arte de CARETAKER. Disponible en: http://sceptre.king.ac.uk/caretaker/docs/TA/SoA%20(D2.1)/CARETAKER_INRIA_DR_P_D21_b1.pdf

- [Corvee2006] Corvee, E., Bremond, F., Odobez, J. "Definición de la ontología y representación del conocimiento". 2006. Disponible en:

http://sceptre.king.ac.uk/caretaker/docs/TA/2nd-annual-review/CARETAKER_INRIA_DR_P_D231_b1.pdf

- [Fipa] Página principal de la fundación FIPA. Disponible en: http://www.fipa.org/

- [Gómez 2003] Gómez, A., Fernández, M y Corcho, O. "Ontological Engineering". Facultad de Informática de la Universidad Politécnica de Madrid. Madrid-Spain. May 2003.

- [IntroProtégé] Sistemas de Representación y Procesamiento Automático del Conocimiento. Introducción a Protégé. Facultad de Informática. Universidad Politécnica de Valencia. Febrero del 2003. Disponible en: http://personales.upv.es/ccarrasc/extdoc/p1_1_parte.pdf

- [Jade] Página principal de Jade. Disponible en: http://jade.tilab.com/

- [Jambalaya] Descarga del plugin Jambalaya,. Disponible en: http://protegewiki.stanford.edu/index.php/Jambalaya_2.6.0

- [JadeOntologías2007] Manual de programación de la plataforma de desarrollo de agentes JADE ("Java Agent Development Framework"). Disponible en: http://programacionjade.wikispaces.com/Ontolog%C3%ADas. Junio del 2007.

- [Martinez2008] Bachiller, M., Martinez, R., Rincon, M., Mira, J. "On the correspondence between objects and events for the diagnosis or situations in visual surveillance task". Science Direct. Department of

Artificial Intelligence. National University of Distance Education. Madrid. 2008. Disponible en: http://portal.acm.org/citation.cfm?id=1363507

- [Ontologias2007] Código de ontologías para multimedia. Septiembre 2007. Disponible en: http://comm.semanticweb.org/examples.

- [Protégé] Página principal de Protégé. Disponible en: http://protege.stanford.edu/

- [RAE] Diccionario en línea de la Real Academia Española. Vigésima Segunda edición. Disponible en: http://www.rae.es/rae.html.

- [Redondo2009] Redondo, A., Roblero, M. y Romero, M. Artículo sobre Ontologías. Disponible en: http://es.geocities.com/recupdeinformacion_ontologias/home.htm. En febrero del 2009.

- [Rota2000] Rota, N. and Thonnat, M. “Activity recognition from video sequences using declarative models.” In Proceedings of the 14th European Conference on Artificial Intelligence (ECAI’00), pages 673–680, Berlin, Germany. 2000. Disponible en: http://www-sop.inria.fr/orion/Publications/Articles/ECAI00.pdf

- [Zoe2006] Zoe, Ll. “Ontología del mapa de la planta de un edificio con Protege-OWL”. Departamento de Ingeniería y Ciencia de Computadores. Universidad Jaume I. October 2006. Disponible en: http://www.dicc-cid.uji.es/InfTec/reports/ICC%202006-10-01.pdf

ANEXOS

ANEXO A. ESQUEMA GENERAL DE LA ONTOLOGÍA CARETAKER

- Physical_Object
  - Contextual_Object
    - Equipment
    - Wall
    - Zone
  - Physical_Object_of_Interest
    - Animal
    - Crowd
    - Group
    - Person
    - Portable_Object
    - Vehicle
      - Aircraft
      - Car
      - Train
      - Truck
- Video_Event
  - Composite_Event
    - CE_Audio_Video
      - shouts_close_to_agitated_group
      - vandalism_against_window
    - CE_Video_Monitoring
      - buying_ticket
      - CE_Airport_Apron_Monitoring
        - Aircraft_Arrival
        - Crew_Arrival
        - GPU_Arrival
        - Jetbridge_Arrival
        - Loading_Luggage
        - Tanker_Operation
        - Tow_Tractor_Operation

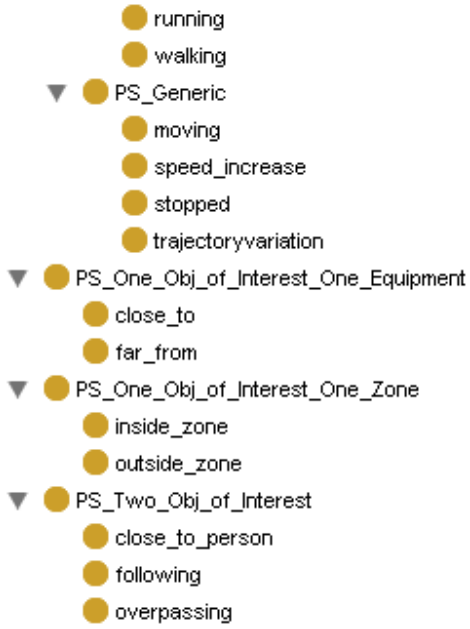

Unloading_Luggage
Validating_ticket
CE_Video_Surveillance
CE_Bank_Surveillance_Event
customer_queuing_at_counter
customer_toward_counter_and_goes_away
customer_waiting
customet_at_ATM
Safe_attack_1person_back_counter
Safe_attack_1person_back_counter_and_door_opened
Safe_attack_1person_infront_counter
Safe_attack_1person_infront_counter_and_door_opened
Safe_attack_2persons
Safe_attack_2persons_and_door_opened
Safe_attack_2persons_back_counter
Safe_attack_2persons_back_counter_and_door_opened
Safe_attack_2persons_back_infront_counter_1
Safe_attack_2persons_back_infront_counter_2
Safe_attack_2persons_back_infront_counter_and_door_opened_1
Safe_attack_2persons_back_infront_counter_and_door_opened_2
Safe_attack_2persons_infront_counter
Safe_attack_2persons_infront_counter_and_door_opened
Safe_attack_2persons_inside_safe_entrance
Safe_attack_2persons_inside_safe_entrance_and_door_opened
Safe_attack_3persons_back_infront_counter
Safe_attack_3persons_back_infront_counter_and_door_opened
Safe_attack_3persons_back_infront_counter_entrance
Safe_attack_3persons_back_infront_counter_entrance_and_door_opened
Safe_attack_3persons_back_infront_counter_safe
Safe_attack_3persons_back_infront_counter_safe_and_door_opened
CE_Crowd
crowd_panic
crowd_splits
crowds_merge

ANEXO A

- oposite_direction
- overcrowding
- rapid_increase_of_crowding_level
- unbalanced_floor_occupation
- CE_Generic
  - access_to_forbidden_area
  - backward_escalator
  - blocking_ZOI
  - enters_restricted_zone
- CE_Group
  - attacking
  - group_staying_in_zone
  - group_stopped_in_zone
  - Vandalism_against_ticket_machine_two_men
- CE_Person
  - dealing_drug
  - following_someone
  - graffiti
  - Jumping
  - jumping_on_the_seat
  - jumping_over_barrier
  - pickpocketing_one_man
  - pickpocketing_several_men
  - sells
  - Vandalism_against_ticket_machine_one_man
  - vandlism
- CE_Train_Surveillance
  - attack
  - individual_meets_a_sitting_person
  - vandal_close_to_window
  - violent_theft
- Composite_State
  - fighting
  - stays_at

- stays_far_from
- stays_inside_zone
- stays_outsize_zone
- waiting
- Primitive_Event
  - PE_One_Physical_Object_of_Interest
    - starts_moving
    - starts_running
    - stops
  - PE_One_Physical_Object_of_Interest_One_Equipment
    - moves_away_from
    - moves_close_to
  - PE_One_Physical_Object_of_Interest_Zone
    - changes_zone
    - enters_zone
    - leaves_zone
  - PE_Two_Physical_Objects_of_Interest
    - moves_away_from_POI
    - moves_close_to_POI
- Primitive_State
  - PS_Audio
    - person_close_to_broken_glass
    - person_close_to_shout
    - person_close_to_tag
    - person_close_to_train_arrival
  - PS_One_Obj_of_Interest
    - One_Group
      - groupwidthvariation
      - quicksplit
    - One_Person
      - legs_up
      - lyingPerson
      - person_sitting
      - person_standing_up

ANEXO A

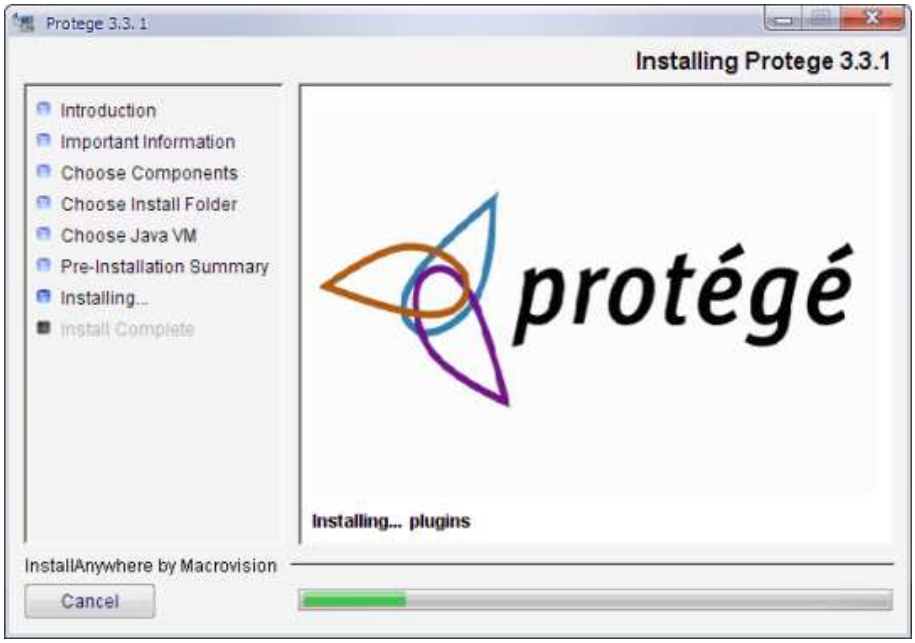

running
walking
PS_Generic
moving
speed_increase
stopped
trajectoryvariation
PS_One_Obj_of_Interest_One_Equipment
close_to
far_from
PS_One_Obj_of_Interest_One_Zone
inside_zone
outside_zone
PS_Two_Obj_of_Interest
close_to_person
following
overpassing

ANEXO B. DESARROLLO DE CLASES DE UNA ONTOLOGÍA USANDO PROTÉGÉ Y BEANGENERATOR

Esto nos sirve cuando trabajamos con ontologías grandes, el trabajo de crear las clases en Java puede llevar mucho tiempo. *Protégé* es un editor de ontologías open source de libre distribución. Como eclipse, Protégé es una herramienta que puede trabajar con muchos plugins. Gracias a un plugin llamado *Beangenerator* implementado por C.J. van Aart del departamento (SWI) de la Universidad de Amsterdam, es posible definir ontologías usando Protégé y dejando a BeanGenerator el trabajo de crear el código fuente.

Esto nos permite trabajar con una interfaz gráfica a la hora de definir nuestras ontologías en vez de tener que escribir el código fuente para las clases.

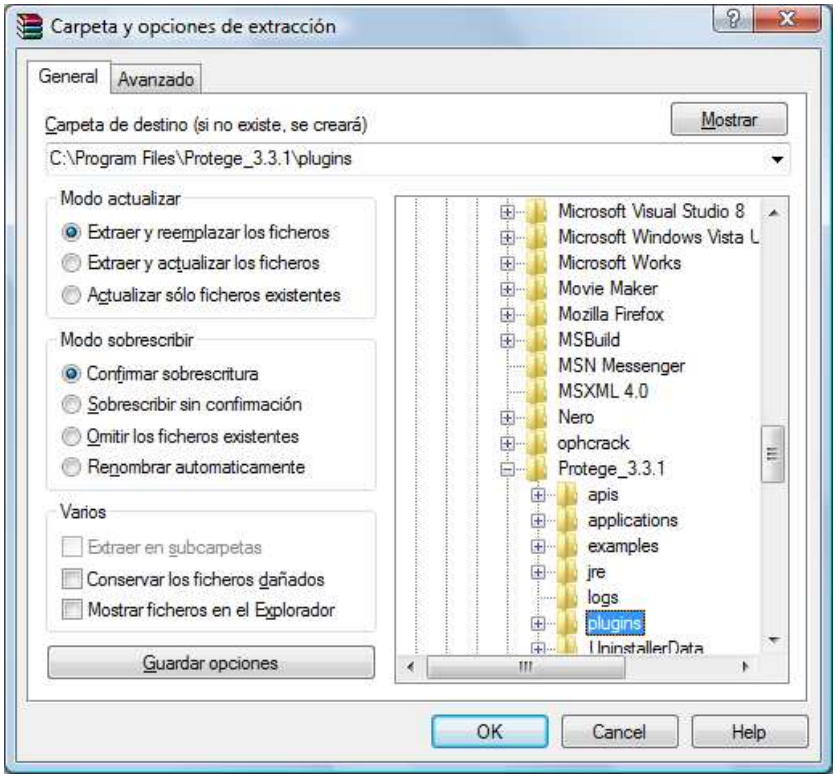
Podemos descargar Protégé de forma gratuita desde la dirección *http://protege.stanford.edu/ .* Una vez descargado se procederá a la instalación, para la cual solo es necesario que tengamos una máquina virtual de Java instalada en nuestro sistema versión 1.5 o superior, de lo contrario deberemos bajarnos el instalable que la incluye.



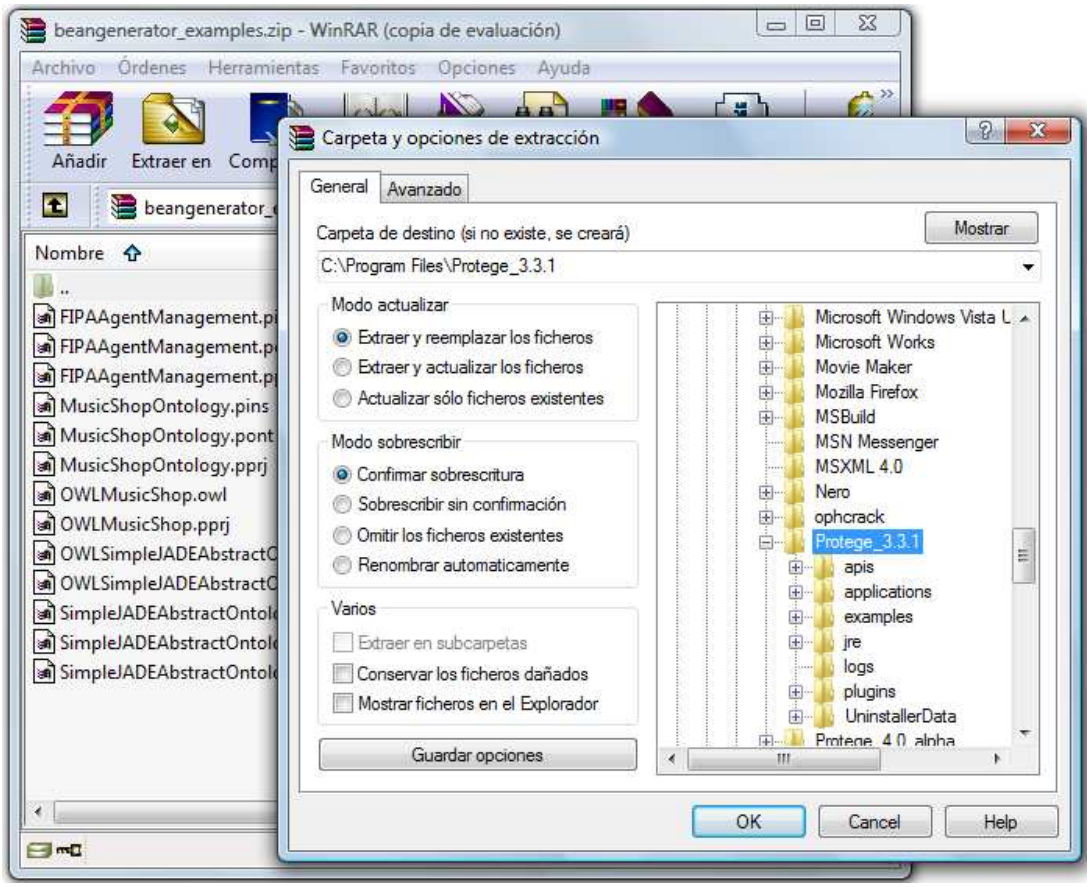
Ahora tendremos que instalar el plugin **BeanGenerator**, que nos permitirá generar las clases de la ontología para trabajar con Jade.

Desde la dirección *http://protege.cim3.net/cgi-bin/wiki.pl?OntologyBeanGenerator* podremos descargar la última versión de este plugin. En el momento de realizar este documento la última versión era la 3.1.1.

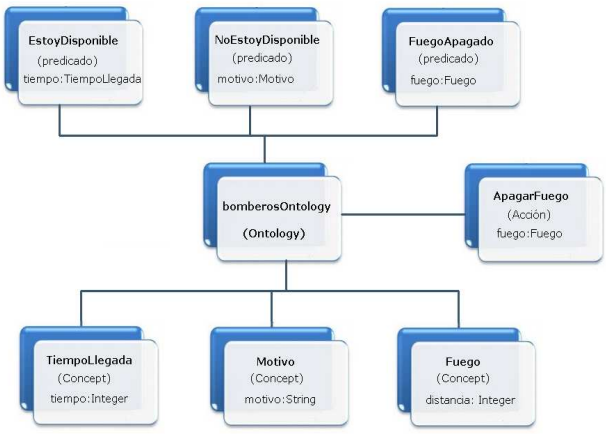
Para instalarlo solo tendremos que descomprimir el archivo **beangenerator_bin_Protege3.1.1.zip** en el directorio **plugins** que cuelga del directorio donde hemos instalado Protege.



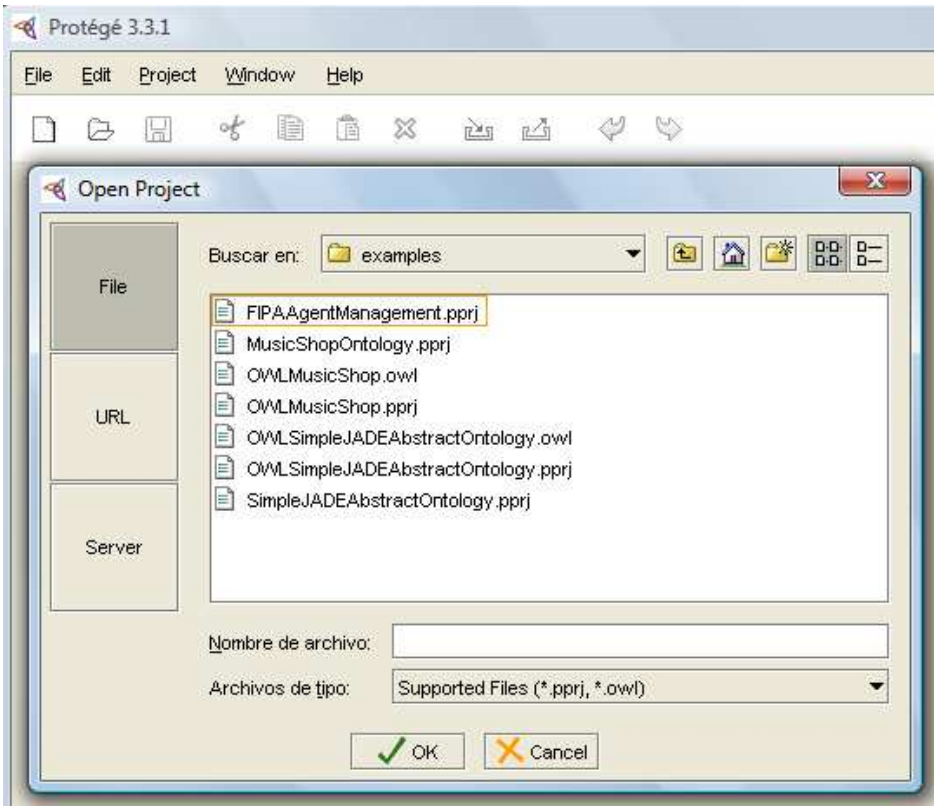
También nos bajaremos el fichero de ejemplos *beangenerator_examples.zip* y lo descomprimiremos en el directorio donde hayamos instalado Protégé.



Ahora ya podemos crear nuestras ontologías con Protégé. Vamos a ver cómo podríamos implementar la ontología **bomberosOntology** para JADE, usando este programa.

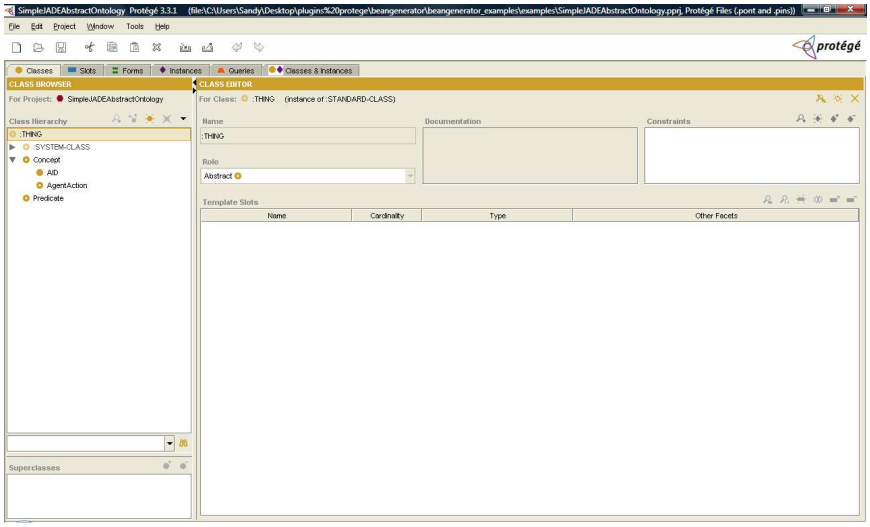


Lo primero será iniciar Protégé y desde el menú **File** seleccionamos **Open…** y abriremos el proyecto **SimpleJADEAbstractOntology.pprj** que cuelga del directorio **Protege_3.2.1\examples**.

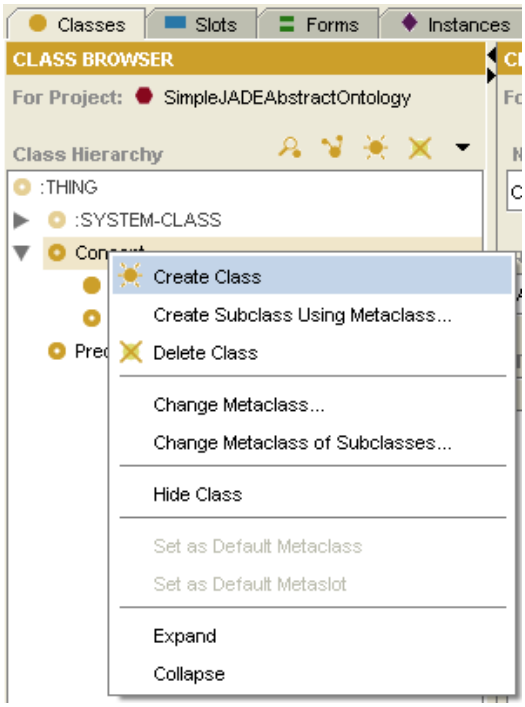


A partir de ahora ya podremos:

- Crear conceptos como subclases de la clase Concept.
- Crear acciones como subclases de la clase AgentAction.
- Crear agentes como subclases de la clase AID .
- Crear predicados como subclases de la clase Predicate.

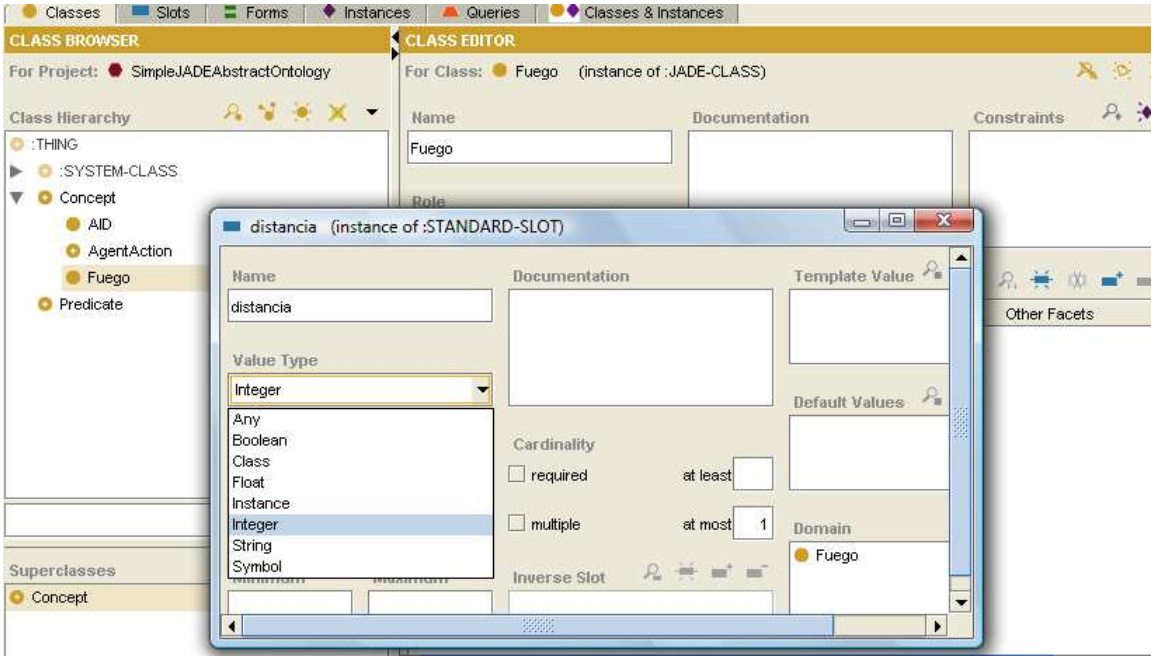


Para crear un concepto solo tenemos que hacer click con el botón derecho sobre la clase **Concept** y pulsar **Create Class**, luego daremos un nombre al concepto. Es importante crear los conceptos antes que los predicados o las acciones, ya que estos últimos van a requerir algún concepto para su definición.

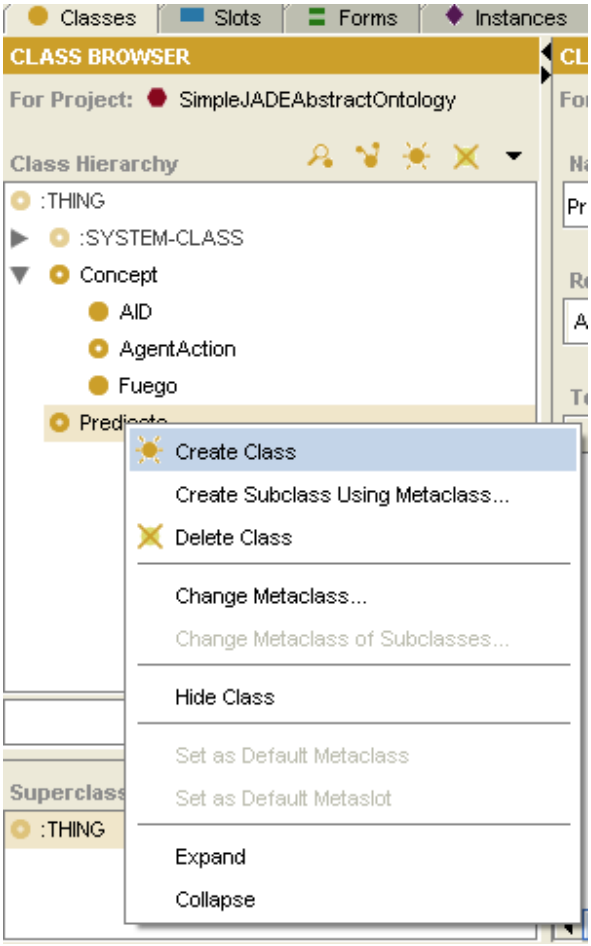


En este caso daremos el nombre **Fuego** al concepto y como este tiene un atributo llamado distancia, tendremos que crearlo haciendo clic sobre el icono **Crear Slot**. Desde la ventana de creación de slot seleccionaremos el tipo del atributo, en este caso **Integer**. Podremos crear slots de tipos básicos como integer, string…etc pero también de clases que hayamos definido.

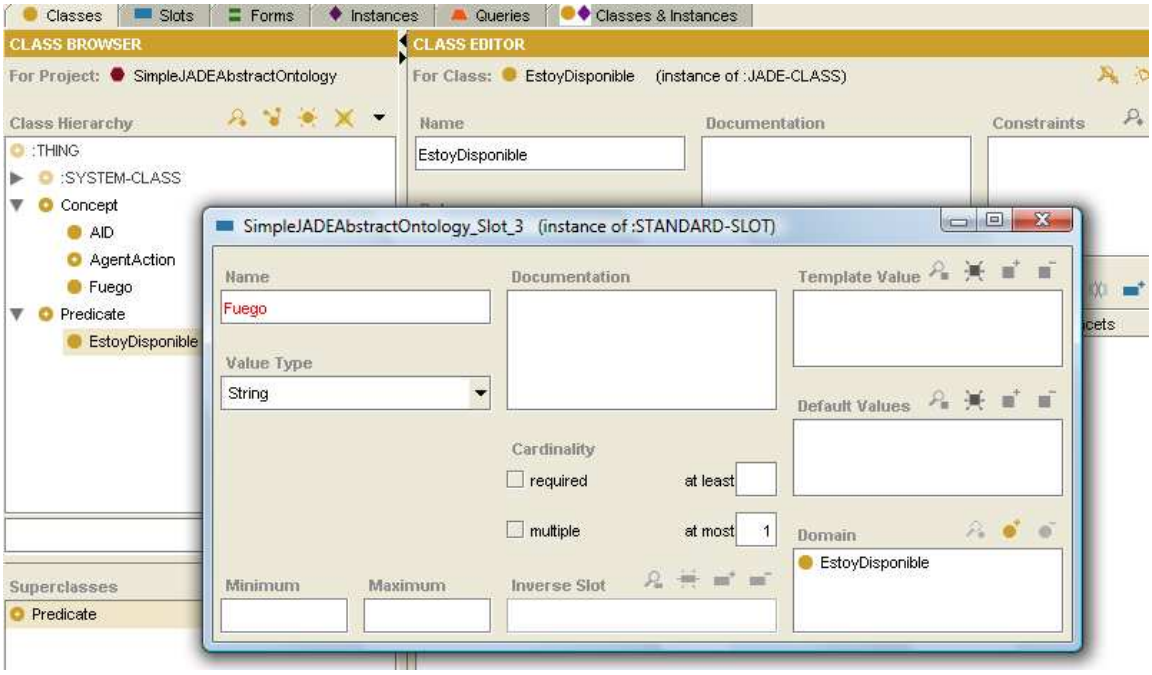
Cuando lo hayamos definido cerramos la ventana y automáticamente se guardará.



Para crear un predicado seguiremos los mismos pasos, seleccionaremos **Create Class** desde el menú desplegable y daremos un nuevo nombre al predicado.

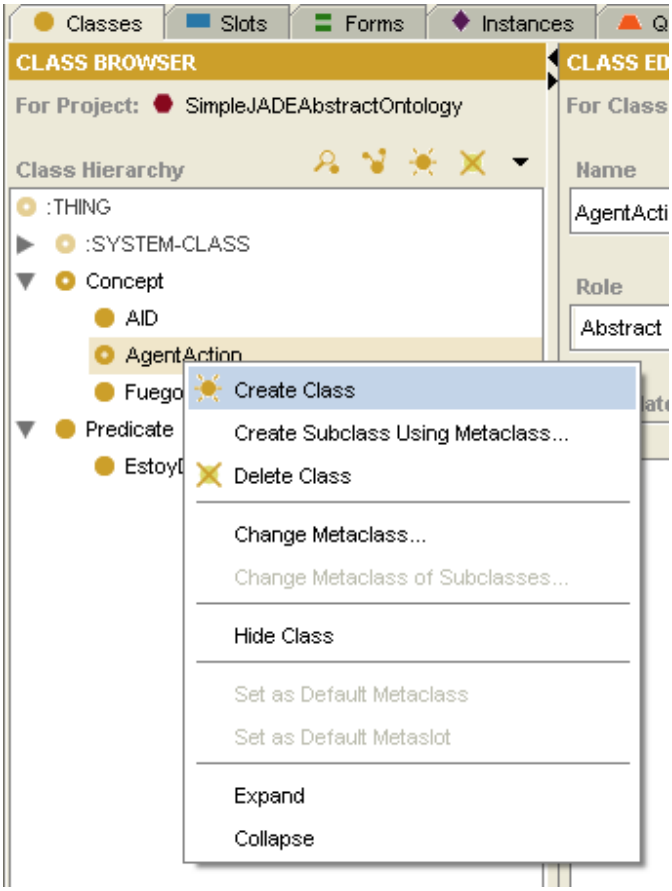


A la hora de asignar nombre tanto a clases como a slots, tendremos que tener cuidado de no elegir nombres que ya estén asignados. Si es así, se mostrarán en color rojo y no nos permitirá crear la clase o el slot.



Si el atributo de una clase tiene como tipo otra clase, como pasa en el caso del predicado EstoyDisponible, solo tendremos que seleccionar **Class** en **Value Type**, pulsar el icono **Add Class** y seleccionar la clase correcta.

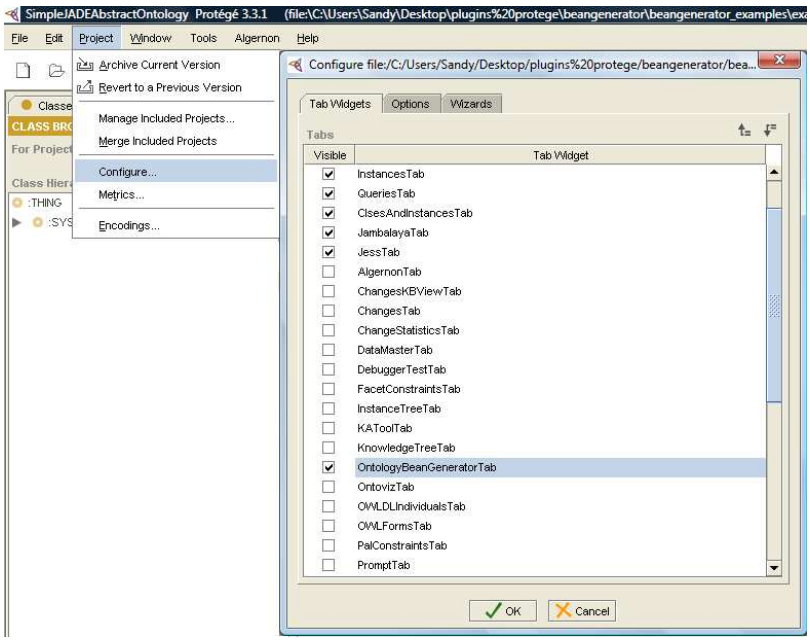
Después de definir todos los conceptos y los predicados, vamos a definir las acciones, en este caso definiremos la acción **ApagarFuego**, lo haremos de forma análoga a como hemos creado los conceptos y los predicados.



Si tenemos que añadir algún atributo que ya se ha definido previamente, como es el caso del atributo FUEGO, que ya ha sido usado en el predicado FuegoApagado, pulsaremos el icono **Add slot** y desde ahí seleccionaremos el atributo correcto, en este caso **FUEGO**.

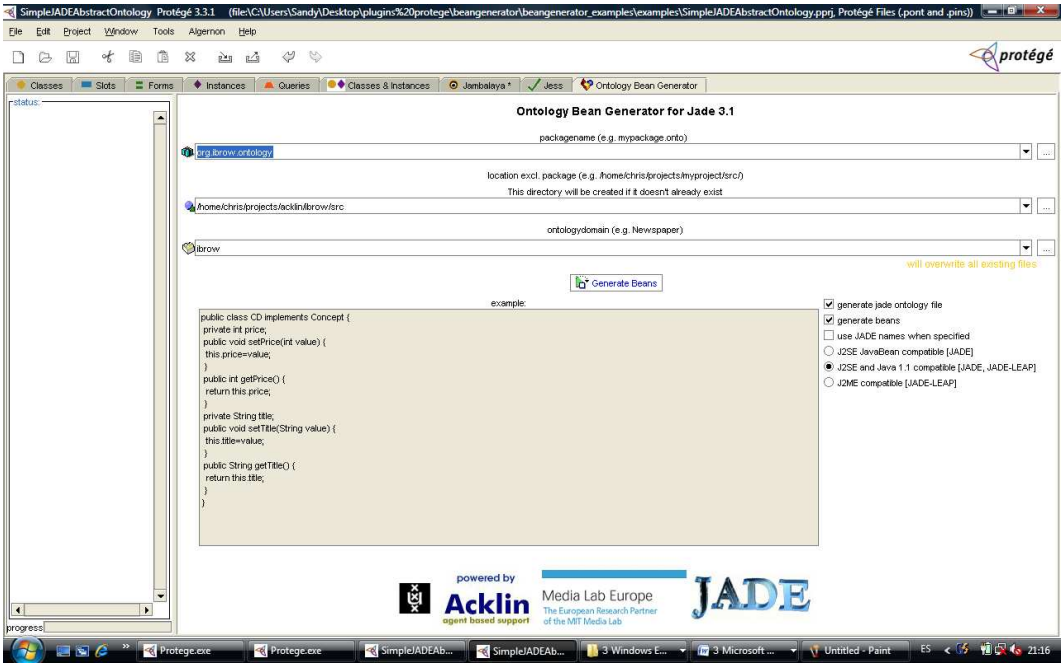
ANEXO B

Ahora que ya hemos definido la ontología, vamos a usar BeanGenerator para que nos genere el código fuente en Java para JADE. Primero activemos el plugin. Desde el menú **Project** seleccionamos la opción **Configure…** y marcamos la opción **OntologyBeanGeneratorTab** y pulsamos **OK**.



Ahora desde la pestaña **Ontology Bean Generator** elegimos el nombre del paquete en que se va a englobar la ontología, luego el directorio donde queremos que nos genere el código y por último el nombre que queremos para nuestra ontología.

ANEXO B


SimpleJADEAbstractOntology Protégé 3.3.1
File Edit Project Window Tools Algernon Help
protégé
Classes Slots Forms Instances Queries Classes & Instances Jambalaya * Jess Ontology Bean Generator
status:
Ontology Bean Generator for Jade 3.1
packagename (e.g. mypackage.onto)
org.ibrow.ontology
location excl. package (e.g. /home/chris/projects/myproject/src/)
This directory will be created if it doesn't already exist
/home/chris/projects/acklin/ibrow/src
ontologydomain (e.g. Newspaper)
ibrow
will overwrite all existing files
Generate Beans
example:
public class CD implements Concept {
private int price;
public void setPrice(int value) {
this.price=value;
}
public int getPrice() {
return this.price;
}
private String title;
public void setTitle(String value) {
this.title=value;
}
public String getTitle() {
return this.title;
}
}
generate jade ontology file
generate beans
use JADE names when specified
J2SE JavaBean compatible [JADE]
J2SE and Java 1.1 compatible [JADE, JADE-LEAP]
J2ME compatible [JADE-LEAP]
powered by
Acklin
agent based support
Media Lab Europe
The European Research Partner of the MIT Media Lab
JADE
progress:
Protege.exe
Protege.exe
SimpleJADEAb...
SimpleJADEAb...
3 Windows E...
3 Microsoft ...
Untitled - Paint
ES
21:16

ANEXO C. INSTALACIÓN DE JADE

JADE (Java Agent Development Framework) es una plataforma de desarrollo de aplicaciones multiagente conforme a los estándares de FIPA (Fundation for Intelligent Physical Agents). Ha sido implementado completamente en Java como código abierto. Se compone de dos elementos principales: una plataforma para agentes conformes a las especificaciones de FIPA y un paquete para desarrollar estos agentes en Java.

Contenido de la instalación de jade-2.61

Se supone un PC con sistema operativo Windows y el entorno Java preparado. Si es necesario deberemos ejecutar en la correspondiente ventana del intérprete de comandos la línea:

set PATH=%PATH%;Directorio_Instalación_Java\bin

Los binarios de Jade-2.61 han sido desarrollados con el JDK-1.2 pero también han sido probados con el JDK-1.3. Los ejemplos de este documento se probaron con la versión JDK-1.4.1_01. Se pueden descargar de la dirección http://sharon.cselt.it/projects/jade/. Además es posible obtener en esta dirección las fuentes, documentación y ejemplos de aplicación.

En nuestra instalación actual, la máquina tendrá una instalación de Jade en el directorio C:\Jade-2.61. Este directorio será referenciado en el resto del documento como JADE. Colgando del directorio JADE tenemos cuatro subdirectorios:

1. *demo*. Ejemplo de uso de Jade. Ficheros para la ejecución y de explicación.

2. *doc*. Documentación de Jade. Incluye la API y manuales de programación, administración, ontologías y seguridad. También hay una explicación de los ejemplos contenidos en el subdirectorio src\examples.

3. *lib*. Jar de Jade.

4. *src*. Contiene dos subdirectorios, demo y examples, El primero contiene los fuentes correspondientes al ejemplo en el directorio demo del punto 1. El segundo contiene las fuentes de varios ejemplos más simples con Jade.

Los ejemplos siguientes han sido extraídos del administratorsguide.pdf, programmersguide.pdf y CLOntoSupport.pdf.

Instalación

Para poder trabajar con Jade es necesario hacer disponibles las correspondientes librerías. Los sistemas multiagente se componen de múltiples elementos, los agentes. Aunque en la mayor parte de este tutorial se empleará un IDE gráfico puede ser necesario arrancar agentes desde la línea de comandos. Para hacer disponibles las librerías desde la línea de comandos del sistema operativo hay que hacer:

prompt> set CLASSPATH=%CLASSPATH%;JADE\lib\jade.jar;JADE\lib\jadeTools.jar;JADE\lib\Base64.jar;JADE\lib\iiop.jar;.

En el caso de un IDE gráfico, nuestro proyecto tendría que tener disponibles las librerías señaladas en la línea anterior jade.jar, jadeTools.jar, Base64.jar y iiop.jar.

Arranque

Los agentes de Jade arrancan dentro de contenedores gestionados por la propia plataforma. A través de estos contenedores se les proporcionan los servicios básicos de ciclo de vida y comunicación. Para arrancar el contenedor principal en modo interactivo empleamos la línea de comandos:

prompt> java jade.Boot –gui

Es posible inicializar contenedores en otras plataformas con la opción host. Para no sobrecargar la máquina con IDE's es recomendable que al menos la plataforma se arranque desde la línea de comandos.

Con el contenedor principal arrancado podemos iniciar agentes con la línea:

java [options] [AgentSpecifier list]

donde *AgentSpecifier list* es una secuencia de strings separados por espacios. Cada string es de la forma

NombreAgente:ClaseAgente(Argumentos)

donde:

NombreAgente es el nombre del agente en la plataforma.

ClaseAgente es el nombre cualificado de la clase que implementa el agente. El contenedor cargará dinámicamente esta clase.

Argumentos es la lista de argumentos que se pasan al agente en su creación.

Un ejemplo con los agentes del directorio examples sería:

Prompt> java jade.Boot –container
sender1:examples.receivers.AgentSender

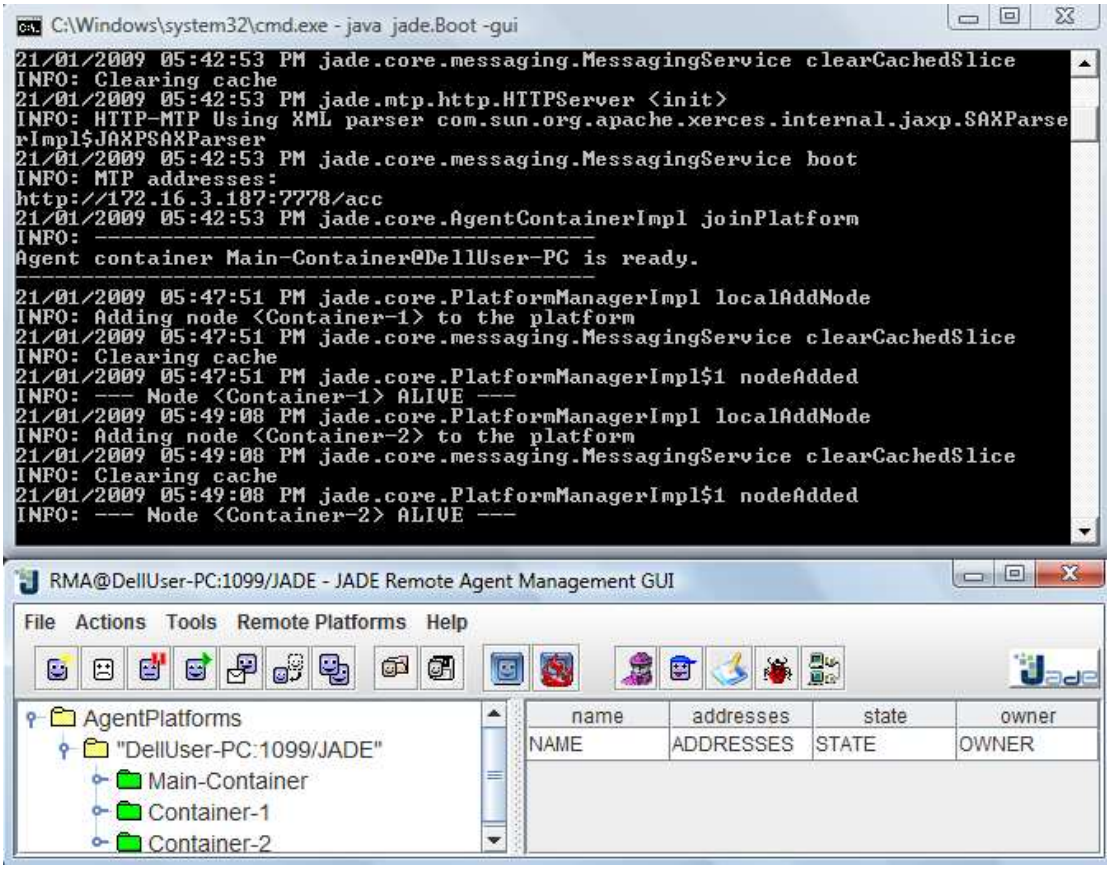
ANEXO D. EJECUCIÓN DE JADE, PARA QUE INTERACTÚEN LOS AGENTES CON LA ONTOLOGÍA FINAL.

Se sigue los siguientes pasos:

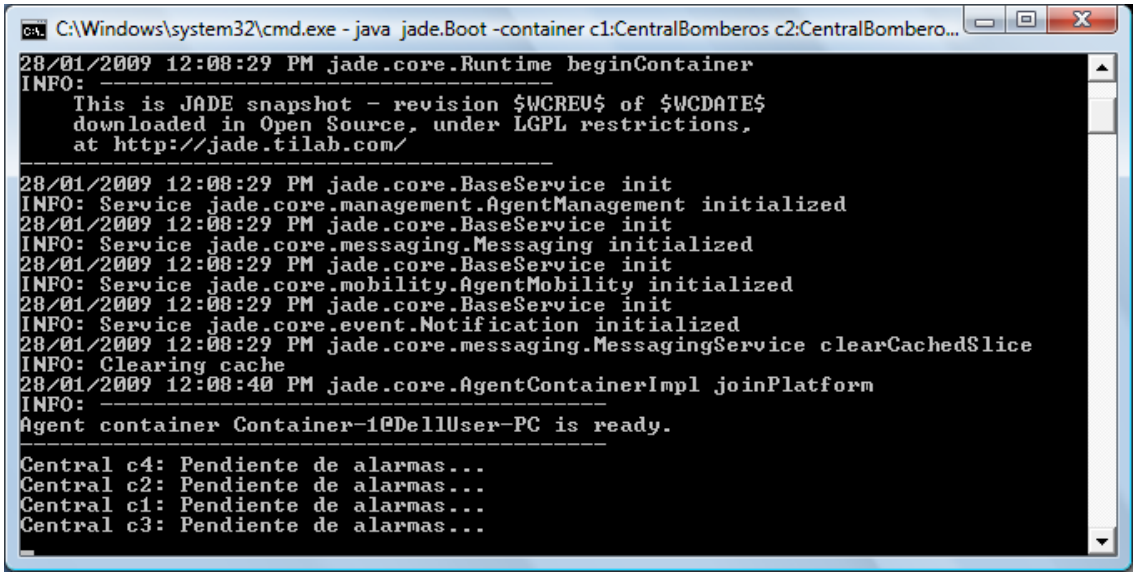
1. Se abren tres consolas de DOS para observar la comunicación entre agentes. En primer lugar en cada consola se establece el CLASSPATH para incluir los archivos JAR en el subdirectorio lib y el directorio actual. De la siguiente manera:

   set CLASSPATH=%CLASSPATH%;.;c:\jade\lib\jade.jar;c:\jade\lib\jadeTools.jar;c:\jade\lib\Base64.jar;c:\jade\lib\iiop.jar

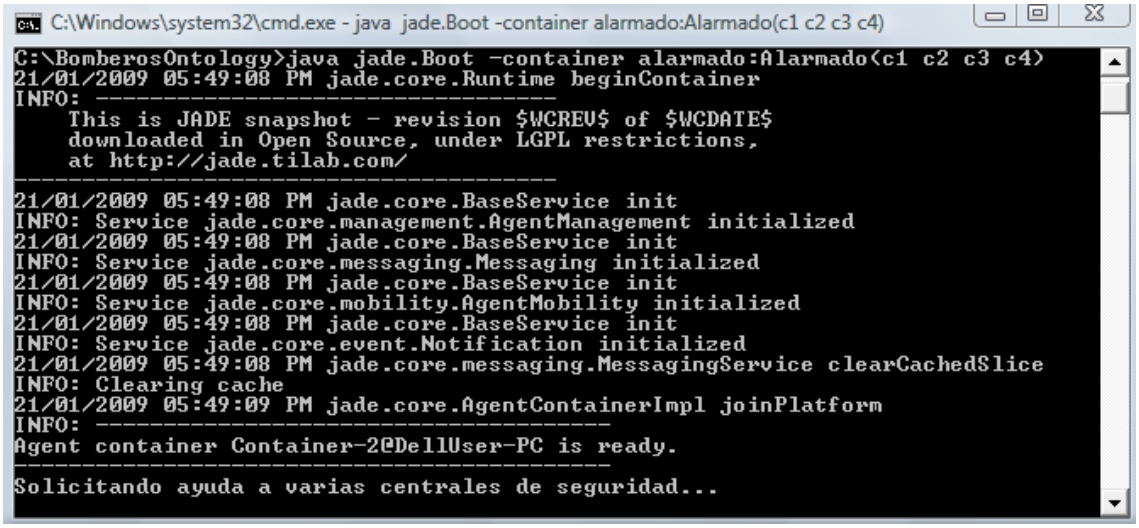
2. En la consola 1, se inicia el contenedor principal de la interfaz de usuario grafico de Jade, con la siguiente sentencia, **java jade.Boot –gui**



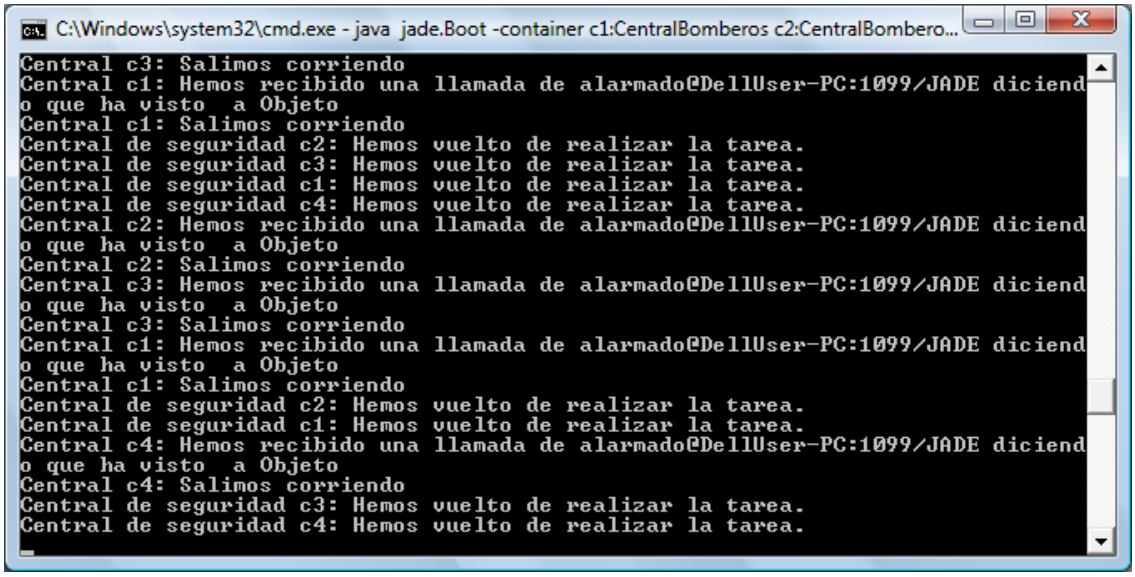
3. Primeramente, en la consola 2, como se observa a continuación, se lanza el primer agente Centralbomberos con el siguiente comando: **java jade.Boot -container c1:CentralBomberos c2:CentralBomberos c3:CentralBomberos c4:CentralBomberos**



Luego, en la consola 3, Lanzamos el segundo agente Alarmado, pasándole como parámetros las centrales de bomberos, con el comando: **java jade.Boot -container alarmado:Alarmado(c1 c2 c3 c4)**



ANEXO C



Y como se puede observar se da la comunicación entre los agentes, CentralBomberos se encuentra pendiente de que haya una alarma, entonces Alarmado observa una persona desconocida en un área prohibida y dispara una alarma en donde solicita ayuda a las centrales, como CentralBomberos se encuentra pendiente de la alarma, al recibir este aviso envía a la central que esté disponible a realizar la tarea de Revisar lo sucedido y regresa devolviendo un mensaje que la tarea ya fue realizada