Gestión de Múltiples Puntos de Vista en Análisis y Diagnóstico de PyMEs

Santiago Almirón (A-2364/7)

27 de marzo de 2011

Tesina de Grado

Licenciatura en Ciencias de la Computación

Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Directora: Dra. Cecilia Zanni-Merk

cecilia.zanni-merk@insa-strasbourg.fr

Índice

Agradecimientos

Me gustaría aprovechar este espacio para hacer un agradecimiento más extenso que el ámbito propio de este trabajo. Este conjunto de hojas simboliza un cierre de una etapa de mi vida, que realmente me marcó y moldeó muchos rasgos en mí. Formándome como profesional y también como persona. Pero esto se debió principalmente por la gente que está y estuvo a mí alrededor.

Primero que nada, quiero agradecer a mis padres, papá Armando y mamá Susana. Ellos lograron inculcarme excelentes valores como la voluntad, perseverancia y responsabilidad. Me enseñaron como debe ser una persona íntegra simplemente mostrándose como son. Siempre me brindaron todas las oportunidades para que hoy sea lo que soy y siempre sentí su apoyo. A ustedes le dedico este trabajo y los quiero mucho.

Mis hermanos, Esteban y Paula, son mis compañeros de vida. Con su experiencia me ayudaron cada vez que lo necesité. Siempre desinteresados y completamente dados como los mejores hermanos pueden llegar a ser. Aconsejándome en los buenos y malos momentos. No me puedo quejar.

A aquellos compañeros de clase que hoy en día son amigos. Recorrieron conmigo todo este trayecto disfrutando los buenos momentos en grupo y masticando las broncas también. Creo que sin ellos hubiese sido todo muchísimo más difícil y ahora no estaría escribiendo esto. En este momento es donde uno se da cuenta que no se puede trabajar solo. Que las diferentes personas se complementan y en conjunto avanzan con mejor ánimo, salen a flote con un objetivo en común. Por un instante, pensé en hacer una analogía con este trabajo.

Para mis amigos fuera de la universidad, que siempre pero siempre me aguantaron bajo cualquier circunstancia. La verdad que debo admitirlo, soy muy afortunado de tener amigos como los que tengo.

No puedo dejar de nombrar a mi directora Cecilia. Le debo reconocer su paciencia con todas las dificultades que pasamos, su excelente predisposición y reactividad. A pesar de las distancias, nunca se perdió el contacto fluido y a toda hora. Gracias por creer en mí y por brindarme más de una oportunidad.

Muchos de los que me conocen saben que soy un tipo que no suelo expresar todo lo que siento (gracias a una persona especial pude mejorar). Por eso creo que esta fue una buena ocasión para decirles lo que siento de corazón.

1. Introducción

1.1. Problemática

Una de las mayores dificultades que encuentran las PyMEs (Pequeñas y Medianas Empresas) es la gestión de la evolución de la misma. Para poder atacar esta cuestión es necesario tener la capacidad de generar un análisis global de todos los aspectos de la empresa. Como por ejemplo, los aspectos económicos, de producción, recursos humanos, ventas, etc. Es necesario que al desarrollar este análisis sea bajo la perspectiva de la evolución de la compañía. La capacidad de dominar los cambios se convierte en un punto clave para las compañías que deben afrontar una fuerte competencia. Es por eso, que comúnmente las PyMEs que se encuentran en esta situación recurran a servicios de consultoría. En este contexto, surge una cuestión importante: ¿cómo se accede al conocimiento existente para permitir un diagnostico de la PyME, teniendo en cuenta a la vez su evolución?

Hay varios puntos importantes que hay que tener en cuenta respecto a la transferencia de conocimiento. El problema principal es que se maneja un gran volumen de conocimiento, tanto teórico como el que provee el experto en el área. Además, este conocimiento suele ser más detallado de lo que se requiere. Por lo tanto, es importante estructurar y jerarquizar esta gran cantidad de conocimiento para permitir el acceso a distintos niveles.

Esta transferencia de conocimiento puede concluir en dos tipos diferentes de resultados.

En primer lugar, puede permitir al consultor y a la compañía poner fácilmente en práctica soluciones típicas. Estas soluciones pueden ser conocidas pero es posible que todavía no hayan sido implementadas en la empresa. El hecho de que estas soluciones conocidas no se hayan puesto en práctica en la empresa anteriormente significa, de todas maneras, que la empresa ha innovado.

En segundo lugar, la transferencia de conocimiento llevada a cabo por el proyecto MAEOS (en el siguiente apartado se explicará en qué consiste) y el razonador automático que está integrado en él podrá permitir al consultor proponer soluciones originales, generando de esta manera innovaciones de alto nivel o de ruptura debido a que son radicalmente nuevas.

1.2. El Proyecto MAEOS

MAEOS (Modélisation de l'accompagnement de l'évolution organisationnelle et strategique des PMEs) es un proyecto de los laboratorios de investigación del INSA (Institut National des Siences Appliquées) de Estrasburgo, Francia.

El objetivo principal es la modelización del soporte para el desarrollo estratégico y organizativo de las PyMEs [1]. Por lo tanto, busca lograr mejorar la eficiencia y el rendimiento del asesoramiento empresarial para las PyMEs. Para lograr cumplir este objetivo, se creó un grupo de trabajo multidisciplinario, cuyas principales áreas de investigación son: inteligencia artificial, ingeniería de software y ciencias de la gestión. Este proyecto apunta a establecer un conjunto de herramientas de software y métodos para el análisis y diagnóstico de las PyMEs. Las herramientas de software deben estar desarrolladas de acuerdo al estado del arte de las PyMEs, en particular, al ambiente administrativo o legal en que ellas se encuentran. Además, deben reflejar las riquezas y contradicciones inherentes a los modelos provenientes de las ciencias de la gestión. Finalmente, debe ser posible transferir el conocimiento obtenido a los consultores.

1.3. Objetivo

En el proyecto MAEOS se pueden considerar tres aspectos importantes que están estrechamente ligados y son, la manipulación del conocimiento, el uso de conocimientos heterogéneos y la transferencia de ese conocimiento.

Particularmente, este trabajo de tesina está situado dentro de la segunda etapa, el uso del conocimiento. En esta parte del proyecto se busca llevar a la práctica el uso del conocimiento actual aportado por los expertos en cada una de las distintas áreas y la teoría subyacente existente. Para luego mediante utilización de motores de inferencia, inferir nuevo conocimiento útil para los consultores de forma que tengan ayuda en la toma de decisiones sobre la evolución de la PyME.

2. Conocimiento

En la actualidad, es común encontrar que el conocimiento se trata de representar de una manera homogénea. Se cubre todo el dominio del problema dentro de una misma base de conocimiento. El enfoque de este proyecto se busca la mayor pluralidad posible para cada una de las bases de conocimiento. Por lo tanto, se mantiene cada una de las propiedades, restricciones y riquezas de los conceptos. Pero esto a su vez genera mayor dificultad al tratar de combinar luego cada una de la gran variedad fuentes de conocimiento. En particular, las áreas referidas a las PyMEs.

El conocimiento es separado en dos grupos importantes. El primer grupo, se utiliza como modelo base el conocimiento teórico sobre la administración de las PyMEs (organización, estrategia, producción, etc.). El segundo grupo, que es utilizado de manera complementaria, es el conocimiento acumulado gracias a la experiencia obtenida durante la práctica. En este trabajo se centrará únicamente en el primer grupo, el segundo queda pendiente para otra etapa de investigación.

2.1. Ontologías

Para modelar el conocimiento teórico optamos por la utilización de ontologías. Estas nos brindan muchas ventajas que luego serán listadas, pero antes de eso, expondremos la definición de ontología en que nos basaremos.

En el área de Inteligencia Artificial existen muchas definiciones de una ontología. Es por eso, que en este trabajo definiremos una ontología como una descripción formal de conceptos dentro de un dominio determinado. Los conceptos poseen distintas propiedades que lo caracterizan, estas pueden ser atributos o también algún tipo relación con otros conceptos. Particularmente, en este trabajo se hace hincapié en las relaciones entre conceptos de distintas ontologías. A su vez, estas propiedades pueden poseer restricciones y condiciones que se deben cumplir.

2.1.1. Ventajas

Existen muchas ventajas por las cuales decidimos modelar el conocimiento utilizando ontologías [2]:

- Tienen la particularidad de brindar una formalización del conocimiento que puede ser interpretado y manipulado tanto por un software

como por una persona. Logran captar la taxonomía y semántica de los conceptos, aunque sean abstractos o teóricos como también aquellos que son reales o físicos.

- No necesariamente todas deben tener un mismo nivel de detalle. Pueden ser o no muy ricas en lo que representan. Luego el software que se encargue de utilizarlas extraerá fácilmente de ellas solamente la información necesaria sin procesamiento adicional.
- Se puede manipular cada una de las distintas ontologías independientemente del contenido que posean. Fácilmente es posible agregar, quitar o modificar una ontología sin que el software que las utiliza se vea afectado en su funcionamiento. Las ontologías pueden reflejar conocimiento muy diverso y de diferentes temáticas, pero todas podrán ser usadas de la misma manera por el software.
- El conocimiento modelado mediante ontologías puede ser reutilizado. Existe la posibilidad que posteriormente en otro ámbito o proyecto este conocimiento sea nuevamente utilizado, extendiéndolo o sesgándolo dependiendo las necesidades.

2.2. Sistema basado en reglas

Para complementar el uso de las ontologías y a partir de ellas poder inferir nuevo conocimiento utilizaremos un sistema basado en reglas. Para dar una introducción a continuación se explicará lo que es un programa declarativo utilizando reglas y como se compone un sistema basado en reglas.

2.2.1. Programación declarativa

Cuando escribimos el código de un programa que busca solucionar un problema bajo el paradigma de la programación "procedural", definimos que hay que hacer, como hay que hacerlo y particularmente en un orden previamente definido de como se deben hacer. En cambio, un programa en programación declarativa se define que es lo que se debe hacer pero omite a posteriori como debe hacerse. Únicamente se describen los detalles del problema que se quiere resolver [3, p. 15].

2.2.2. Reglas

En nuestro caso utilizaremos programas declarativos basados en reglas. Una regla se asemeja a una instrucción o un comando que aplica en ciertas situaciones. En particular, las podemos comparar con un condicional IF-THEN de la programación imperativa, pero difieren con ella principalmente en el hecho de que las reglas no son ejecutadas según el orden previamente establecido, como por ejemplo el orden en que fueron escritas en el código fuente.

Las reglas están compuestas de dos partes principales. La primer parte, la izquierda (en inglés Left-Hand-Side o LHS), corresponde al predicado o premisas que se deben cumplir para que la regla se active. La segunda parte, la derecha (en inglés Right-Hand-Side o RHS), corresponde a las acciones que se deben realizar o las conclusiones a las que se llegan cuando la regla se dispara; siempre y cuando la regla haya sido previamente activada. El motor de inferencia es el que finalmente decide en qué momento una regla que esta activa se dispara [3, p. 17].

2.2.3. Motor de inferencia

Los programas declarativos deben ser ejecutados por un algún tipo sistema que comprenda la información declarativa y a partir de ella resuelva el problema. Aquí es donde este sistema elige y define el control de flujo de la ejecución del programa [3, p. 20].

En nuestro trabajo este sistema será un motor de inferencia basado en reglas. Utilizando un motor de inferencia es posible agrupar todo el extenso conjunto de reglas ligadas a una determinada ontología e inferir nuevo conocimiento. Al poseer una ontología rica en detalles con un gran conjunto de reglas bien definido es posible concluir en nuevo conocimiento que quizás un experto en el tema en las mismas condiciones no haya podido apreciar. Esto se debe a que el motor de inferencia al momento de deducir tiene en cuenta todas las premisas existentes y deduce todo lo que es posible a su alcance.

2.2.4. Ventajas

Al utilizar reglas se desprenden varias características interesantes que nos son útiles [4]:

- Brindan una forma expresiva y simple de definir que, a partir de la veracidad de un conjunto de premisas se deduce una conclusión.
- Suelen ser más fáciles de comprender que el mismo problema basado en programación imperativa.
- Nos ofrecen la posibilidad de definir premisas relativamente simples pero también aquellas que son potencialmente complejas.
- Son mucho más flexibles al momento de realizar modificaciones. En el trabajo de la modelización del conocimiento suelen surgir cambios muy a menudo, por ende, esta flexibilidad es indispensable.
- En nuestro caso, nos abstraemos del software propiamente dicho del sistema experto. Tanto el experto que modela el conocimiento como también el usuario final en un futuro podrá realizar modificaciones para satisfacer sus necesidades.

En este trabajo utilizamos ontologías heterogéneas por lo tanto se trataran distintas temáticas. Pero sucede que en ciertos puntos las ontologías comparten similitudes entre conceptos, entonces, surge la necesidad de interconectarlas. No necesariamente estos conceptos están modelados de la misma manera, pero el experto puede reconocer que existe una equivalencia semántica entre ellos. Es aquí donde las reglas nos brindan la posibilidad de crear un puente semántico entre diversas ontologías, pudiendo así llegar a conclusiones más integradoras.

3. Sistema Experto

Como se mencionó en la introducción, el propósito de este trabajo es el desarrollo de un software capaz de utilizar e inferir nuevo conocimiento al partir del existente. Se plantea un problema, por ejemplo, el estado actual de una PyME y este devuelve una recomendación de como se podría abordar a una solución.

Teniendo como meta la heterogeneidad y pluralidad del conocimiento es difícil pensar que un solo experto puede atacar un problema con gran diversidad de aristas. De la misma manera que existen diferentes expertos que modelan el conocimiento bibliográfico y él obtenido a partir de la experiencia, tendremos en nuestro sistema informático diferentes expertos que visualizarán el estado del problema y aportarán posiblemente una solución parcial como respuesta. Conformando de esta manera un sistema de experto.

La arquitectura de software que adecua a nuestras necesidades es el Blackboard System [5]. Un Blackboard System consta de tres componentes principales (ver figura 1): el *Blackboard* propiamente dicho, las *fuentes de conocimiento* (también los llamaremos *agentes*) y un subsistema de *control*.

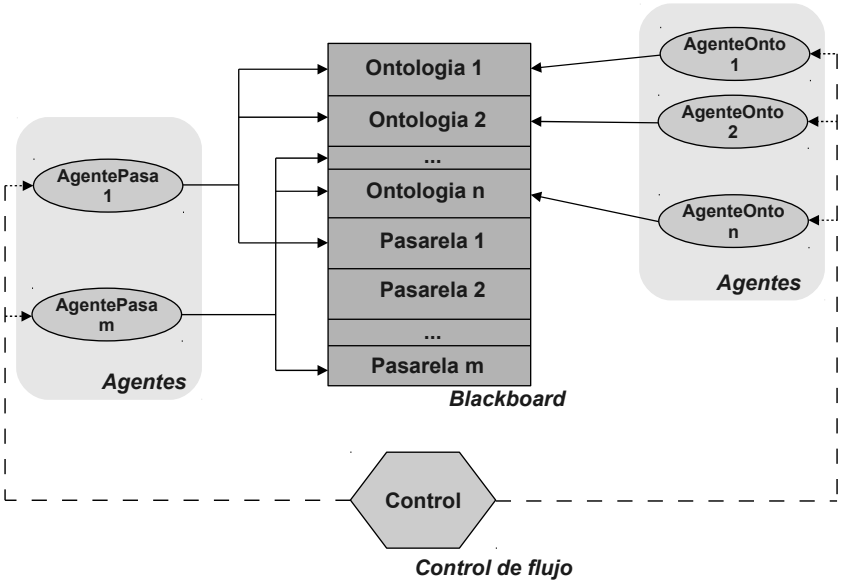

Figura 1: Arquitectura Blackboard

Este sistema lleva el nombre MAMAS (MAEOS Argumentation with a Multi-Agents System) y a continuación se mostrará primero las tecnologías

utilizadas. Luego se explicará cada una de las partes que lo componen y sus características más relevantes.

3.1. Tecnologías

Principalmente en la implementación de MAMAS se utilizan dos herramientas conocidas que influyen fuertemente en la utilización del conocimiento. Protégé-Frames[1] para la representación y manipulación de las ontologías y Jess[2] como motor de inferencia de reglas. Más adelante, en el capítulo 4 se verá como se representa el conocimiento y como realiza la interconexión entre estas dos herramientas.

3.1.1. Protégé-Frames

En este trabajo utilizamos Protégé-Frames el cual está basado en Frames de acuerdo al protocolo OKBC (Open Knowledge Base Connectivity). En Frames el modelado de las ontologías consisten en, un conjunto de clases organizadas jerárquicamente para representar los conceptos, un conjunto de slots asociados a las clases que describen sus propiedades y relaciones, los slots pueden ser restringidos utilizando facetas, y finalmente, un conjunto de instancias de esas clases con sus respectivos slots. Por lo tanto, se condice con la definición de ontología elegida en el capítulo anterior.

Protégé es una plataforma que provee una suite de herramientas que permiten la construcción de modelos y aplicaciones basadas en conocimiento con ontologías. Es gratuita, de código abierto bajo Mozilla Public License desarrollada y mantenida actualmente por la universidad de Stanford.

3.1.2. Jess Rules

Es un motor inferencia mediante reglas basado en el lenguaje de CLIPS[3], pero a diferencia de CLIPS este está escrito enteramente en JAVA. Además, internamente esta implementado utilizando el algoritmo Rete[3, p. 136].

El conocimiento en Jess se puede definir mediante el uso de templates, los cuales poseen slots que lo describen. Las instanciaciones de los templates, se denominan facts. Pero lo principal en Jess es el poder y la facilidad de escribir definiciones de reglas (rules).

[1] Sitio web del editor Protégé-Frames: `http://protege.stanford.edu/`
[2] Sitio web del motor Jess: `http://www.jessrules.com/`
[3] Sitio web de CLIPS: `http://clipsrules.sourceforge.net/`

Jess cumple con nuestras necesidades detalladas en todo el capítulo 2, es por esto, que en nuestro trabajo será el corazón de las fuentes de conocimiento, ya que en ellos yacen las reglas de inferencia sobre la ontología que conocen.

El software es mantenido por Sandia National Laboratories y posee una licencia gratuita para fines de investigación y desarrollo académico con validez por un año. Al igual que Protégé, Jess brinda una API bien documentada para poder poblar y controlar el motor.

3.2. Blackboard

Es un repositorio de datos pasivo donde las fuentes de conocimiento visualizan el estado del problema, extraen información que le es relevante y luego agregan en él una nueva solución parcial del problema. El Blackboard por sí solo no altera ni genera nuevo conocimiento, simplemente almacena. Provee una interfaz que abstrae a las fuentes de conocimiento sobre forma en que los datos se almacenan y facilita la diferenciación de los distintos grupos de soluciones parciales.

Los nuevos facts inferidos por los agentes no son agregados de forma directa dentro del repositorio principal sino que primero deben pasar por una memoria intermedia. Uno o más agentes pueden dejar sus resultados en este lugar, y al final, será el control es el encargado de decidir que resultados finalmente quedaran plasmados en el Blackboard.

Además de almacenar facts también se guarda el orden en que las reglas se fueron disparando, los facts que activaron dichas reglas y los facts deducidos a lo largo de la ejecución de cada agente. Si bien no es imprescindible, ayuda bastante a comprender cómo se obtuvieron los resultados y a detectar posibles errores en las reglas o datos iníciales.

Otra condición a cumplir por Blackboard es debe que ser capaz de diferenciar los distintos tipos grupos de soluciones parciales. Como este proyecto está muy ligado al motor de Jess usaremos los denominados modules o módulos para poder separar el conocimiento. Un módulo en Jess es simplemente un identificador que se antepone a los templates, reglas y facts, que agrupa diferentes conjuntos dentro de un mismo motor. Por lo tanto, a cada agente le asignaremos un módulo y será ese su ámbito de trabajo. Allí podrá recuperar y almacenar nuevo conocimiento sin comprometer directamente al resto de los agentes y al momento de inferir su foco de atención estará da-

do por el módulo. Más adelante, veremos que los módulos tomarán mayor importancia cuando se creen relaciones entre facts de distintas ontologías.

3.3. Fuentes de conocimiento (Agentes)

En nuestro caso cumplen el rol de los expertos especialistas. Poseen un algoritmo o un conjunto de reglas que dependiendo de la información que extraen del Blackboard aportan una solución parcial del problema. Por sí solos no pueden resolver el problema completo, ya que si fuese así la arquitectura Blackboard perdería su sentido. No conocen de la existencia de las demás fuentes de conocimiento, por ende, no tienen una comunicación directa entre sí. Al momento de inferir trabajan de manera independiente, pero las soluciones parciales plasmadas en el Blackboard por otras fuentes de conocimiento pueden afectar indirectamente su resultado.

Hay dos tipos de agentes definidos: los agentes que trabajan infiriendo nuevo conocimiento a partir del existente en el Blackboard (AgenteOnto) y los agentes que se encargan de buscar y definir las equivalencias semánticas existentes entre las diferentes ontologías (AgentePasa).

Las fuentes de conocimiento también las denominamos agentes a lo largo de este informe, debido a que son identidades independientes, que analizan su entorno y mediante su inteligencia acotada infieren soluciones. Además, se comunican con su entorno de forma indirecta mediante el Blackboard.

Una ventaja importante que se desprende de esta arquitectura, es la facilidad con la que se pueden agregar, quitar o modificar los diferentes agentes. Tanto como agregar agentes de los tipos conocidos, como también agregar nuevos tipos de agentes. Esto se debe a dos cuestiones, la primera es que el Blackboard tiene una completa interfaz para acceder a la información, suficientemente clara como para extraer los datos relevantes. La segunda, es que todos los agentes poseen una interfaz homogénea sin importar la tarea que desempeñen.

La interfaz de un agente posee cuatro métodos:

- **Inicialización**: se crea la instancia del agente, se le asigna un área de trabajo y se realizan las tareas propias de inicialización dependiendo el tipo de agente que sea.
- **Verificar la precondición**: con el objetivo que el control pueda tomar una decisión de que agentes ejecutar, primero los consulta para

ver si pueden trabajar o inferir nuevo conocimiento. Por lo tanto, este método devuelve verdadero si que el agente tiene la posibilidad ejecutarse y falso en caso contrario.

- **Ejecutar**: se recuperan todos los facts desde el Blackboard, infiere o trabaja y luego dependiendo la tarea que realiza, guarda los nuevos facts en el Blackboard.

- **Finalización**: cuando el sistema finaliza y concluye en una solución, el control le solicita a los agentes que guarden sus resultados fuera de la memoria principal. Por ejemplo, los AgenteOnto guardan las nuevas instancias en el proyecto de Protégé-Frames asociado.

Los dos tipos de agentes implementados hasta el momento poseen internamente una instancia del motor de Jess. Por lo tanto, cada uno de ellos tiene un repositorio temporal y secundario en el cual pueden trabajar sin alterar el estado del Blackboard.

3.3.1. AgenteOnto

El AgenteOnto es el único encargado de recuperar el proyecto Protégé-Frames asociado a la ontología en la que es experto y las reglas de Jess vinculadas a dicha ontología. Extrae las clases e instancias de la ontología, las convierte en templates y facts de Jess con el fin de después poder utilizar las reglas. Finalmente, puebla el Blackboard con los datos iniciales dentro del módulo que tiene asignado.

Como se explicará en el próximo capítulo el experto al modelar las ontologías debe tener en cuenta las decisiones de diseño. Pero contará con la ayuda del agente que controlará que se satisfagan estas condiciones al momento de la inicialización e informándole donde no se cumplieron.

Cuando se le consulta al agente si se verifica su precondición lo que él hace es ejecutar su motor de inferencia pero con la limitación de que se detenga si define al menos un nuevo fact. Si pudo inferir entonces su precondición es verdadera, de caso contrarió el agente no tiene nada para aportar en la resolución del problema y devolverá falso.

De la misma manera que en la inicialización se convierte el conocimiento de Protégé-Frames a Jess, se realiza el proceso inverso cuando el sistema completo no puede inferir más. Este proceso es más sencillo debido a que solo se debe actualizar las instancias del proyecto Protégé-Frames.

Este tipo de agente solamente manipula y deduce conocimiento sobre la ontología en la es experto. A primera vista, parecería que el trabajo del AgenteOnto no afecta al resto de los agentes, pero es aquí donde el AgentePasa toma injerencia.

3.3.2. AgentePasa

El AgentePasa se encarga de generar un nexo o puente semántico entre conceptos similares de dos ontologías distintas. Su objetivo es interconectar las ontologías y enriquecer las soluciones parciales que van generando independientemente los AgenteOnto. Apuntando a generar soluciones integradoras y totales.

Para su funcionamiento necesita un conjunto de reglas de equivalencia semántica (en el apartado 5.2 se explicará con más detenimiento) y los dos módulos de Jess a los cuales va a interconectar. Tiene la facultad de obtener y agregar facts en sus dos módulos asignados para lograr su objetivo.

Al igual que el AgenteOnto este verifica su precondición infiriendo al menos un nuevo fact y guarda sus resultados en la memoria intermedia del Blackboard. Pero la principal diferencia es no convierte ni manipula proyectos Protégé-Frames debido a que de eso se encargan los AgenteOnto.

3.4. Subsistema de control

Si llevamos a la vida real la arquitectura de un Blackboard tendríamos, por ejemplo, un problema escrito en un pizarrón y muchos estudiantes visualizando el problema al mismo tiempo. Si a cada uno de ellos se le ocurre una idea de cómo solucionar un parte del problema e inmediatamente escriben en el pizarrón su solución, se generaría posiblemente un problema de concurrencia al escribir en él ya que es un recurso compartido entre todos. Además, alguno de los alumnos también podría tener más tiempo para él el pizarrón sin dejar que otros escriban. Por lo tanto, es necesario tener un docente o coordinador de alumnos para que todos puedan aportar lo que saben en tiempo y forma.

Es aquí donde el control de los agentes cumple un rol importante en el funcionamiento del sistema en general. Definiendo cuando comienza la inferencia, eligiendo cual será el próximo agente a ejecutar y finalmente decidiendo cuando se ha llegado a la solución final.

El control tiene conocimiento de todos los agentes del sistema y es el único que controla su funcionamiento. Se pueden implementar diferentes estrategias de razonamiento en él, que explicitan cómo y en qué orden deben ser accionados los diferentes agentes del sistema (se verá en detalle en el apartado 5.4). Cuando los agentes son inicializados se les puede definir una prioridad dentro del sistema pero es el control quien finalmente decide si tomarlas en cuenta.

4. Representación del conocimiento

Una vez planeadas las herramientas a utilizar y la conformación del sistema experto veremos como estas elecciones nos traen aparejado ciertos problemas que debemos sobrellevar tomando determinadas decisiones de diseño. En primer lugar, detallaremos las decisiones de diseño con una breve explicación de su por qué. Luego, mostraremos como se modela una clase en Protégé-Frames y cómo está compuesto un template en Jess. Por último, como es que deben se escribirse las reglas en Jess.

4.1. Convirtiendo Clases en Templates

Los primeros problemas con los que nos encontramos en este trabajo se debieron a la transformación del conocimiento modelado en Protégé-Frames a Jess. En Protégé-Frames tenemos las clases e instancias y su equivalente en Jess son los templates y facts respectivamente. Luego veremos, que el sistema experto toma las instancias, las convierte en facts, las manipula y los facts resultantes vuelven reconvertirse en instancias de Protégé-Frames. El problema principal es que Protégé-Frames es más expresivo que Jess porque posee características y funcionalidades que no pueden ser representadas nativamente en Jess. Por lo tanto, pueden surgir inconsistencias que por suerte pueden ser en mayor parte evitables y en otros casos salvables.

Es por esta razón que fue importante tomar varias decisiones de diseño antes de comenzar a trabajar sobre las ontologías. Dos tipos importantes de decisiones fueron tomadas, de funcionamiento y de implementación.

4.1.1. Funcionalidades

Es necesario acotar a un conjunto reducido pero conocido de funcionalidades y características de Protégé-Frames.

- **Herencia múltiple**: tanto Protégé-Frames como Jess permiten heredar de una clase y template respectivamente. Por lo tanto, las características y slots de sus respectivos padres les serán heredados. Pero en Protégé-Frames existe la posibilidad de que una clase herede desde múltiples padres, en cambio, en Jess un template solo puede extenderse de un único padre. Si bien en el conjunto de ontologías de este trabajo no es necesaria esta característica, no está soportada.

- **Facetas**: Las únicas dos facetas de Protégé-Frames que pueden utilizarse en Jess son, allowed-values y default-value. El resto de las facetas no son soportadas nativamente por Jess. Por ende, se puede y se recomienda utilizar toda la variedad de facetas que provee Protégé-Frames. Con el objetivo de chequear estas restricciones sobre las instancias iniciales, y luego de la ejecución del sistema, sobre las instancias deducidas por el sistema experto. Pero se debe recordar que durante el proceso de inferencia no serán tenidas en cuenta.

- **Clases concretas**: Las clases deben ser concretas en Protégé-Frames, debido a que en Jess no es posible hacer esta distinción entre abstractas y concretas.

- **Tipos de Slots**: Los tipos de datos que son aceptados de Protégé-Frames son Boolean, Integer, Float, String, Symbol e Instance. El tipo Class no es necesario en este trabajo y no esta soportado.

- **Multi-Slots**: Protégé-Frames y Jess soportan la cardinalidad múltiple de los slots, pero particularmente en las ontologías que utilizamos solo serán necesarias para los slots de tipo Instance. Entonces, para simplificar el resto únicamente se debe utilizar slots con cardinalidad single.

4.1.2. Implementación

Las que se deben a cuestiones propias de implementación.

- **Valores por defecto**: una condición necesaria en Jess es que los slots deben poseer siempre un valor por defecto. Esto no es un requerimiento en Protégé-Frames, por lo tanto, trae aparejado algunos inconvenientes. Si en Protégé-Frames un slot ya posee un valor por defecto este es directamente trasladado a Jess. Pero de caso contrario se verá a continuación caso por caso.

- **Slots String y Symbol**: Existe una relación biunívoca entre Protégé-Frames y Jess para estos dos tipos de slots. Jess posee valor nulo en común para estos tipos, denominado nil y por defecto se utiliza este valor en caso de no especificar ninguno.

- **Slots Integer y Float**: Para estos dos tipos solamente existe una restricción importante, es que lo valores deben ser mayores o iguales que cero. En Jess no existe un valor nulo como con los String o Symbol, por lo tanto, utilizaremos el valor -1 por defecto en el caso de no especificar ninguno. En nuestro caso, esta restricción no afecta debido a que todos los valores son positivos.

- **Slot Boolean**: Este tipo de datos en Jess es implementado como un Symbol con los valores permitidos TRUE y FALSE. El sistema experto se encarga de realizar la conversión de Boolean a Symbol y viceversa. Pero se debe tener en cuenta la correcta escritura de los booleanos en las reglas de Jess.

- **Slot Instance**: Todos los facts en MAMAS poseen un slot numérico ID, que define un identificador único de fact dentro de todo el sistema (mas detalles ver el apartado 4.2). Los slots de tipo Instance pueden tener cardinalidad single o multiple, por lo tanto, se implementarán con un valor único numérico o una lista de enteros respectivamente. En particular, la cardinalidad múltiple se implementa con un multislot en Jess. Cuando se quiere asociar un valor al slot solo vasta asignar el ID del fact dentro de su módulo. En el caso de no tener ningún valor asociado, el valor por defecto para un slot es -1 y para un multislot la lista vacía.

- **Codificación de nombres**: Jess y Protégé-Frames pueden manipular nombres de clases, slots, instancias, templates, facts y reglas con caracteres especiales, pero ambos lo realizan de distinta forma. El principal problema está en la escritura de las reglas, debido a que es necesario agregar caracteres adicionales indicando cuales y como son los caracteres especiales. Además, el experto tendría que tener todo el tiempo presente el conjunto de caracteres especiales permitidos. Por lo tanto, pasa simplificar el problema se definió el siguiente conjunto caracteres permitidos, es pequeño pero suficiente {a-z, A-Z, 0-9, _}.

- **Nombre de Clases**: Para facilitar la tarea de la escritura de las reglas, se optó por mantener el mismo nombre para de la clase para su correspondiente template.

4.1.3. Observaciones

- Dentro del campo documentación en las clases de Protégé-Frames se puede agregar información adicional del concepto, pero no modifican el significado formal de la clase.

- La característica de cardinalidad máxima en los slots no es tenida en cuenta. En cambio, la cardinalidad mínima en Jess es siempre 1, ya se necesita siempre al menos un valor por default.

4.2. Instancias y Facts

Por un lado, en Protégé-Frames tenemos las instancias creadas a partir de su clase asociada. De la misma manera, existen los facts definidos a partir de su template en Jess. Como anteriormente buscamos que las clases se correspondan a los templates entonces la correspondencia entre las instancias y los facts se obtiene fácilmente. En único detalle que hay que tener en cuenta, es que las instancias en Protégé-Frames obligatoriamente deben tener un nombre único que las identifique y las diferencie de las restantes instancias. Para mantener esta correspondencia, se agrega un nuevo slot de tipo String llamado Name a cada uno de los templates y en él se copia el nombre de la instancia al momento de poblar los motores de Jess. Este slot es agregado automáticamente por MAMAS, pero es importante que el usuario no cree un slot con este nombre ya que está reservado para el sistema.

Cuando una instancia de Protégé-Frames tiene un o más slots del tipo Instance significa que él está compuesto por otras instancias de su base de conocimiento. En Jess este tipo de slot no existe, es por esta cuestión que fue necesario crearlo a partir de los tipos de datos existentes. Para salvaguardar este inconveniente y llevar a cabo la conversión fue necesario definir un nuevo slot para todos los facts. El slot es denominado ID, es de tipo Integer y contiene un numero identificador fact único dentro de todo el sistema. Jess posee un identificador similar llamado FactID, pero este valor es generado de forma interna y controlado dentro de cada motor de inferencia. Al ser interno de cada motor no se tiene acceso para su creación y modificación. Es por esto que el Blackboard provee una función global de Jess que genera un valor único cada vez que se la invoca. Esta función se llama (getid) y es obligatorio su uso en cada assert dentro del slot ID de cada una de las reglas. Mientras el motor de inferencia dentro de un agente está funcionando no se

puede corroborar un ID incorrecto, pero al finalizar se chequea la existencia de una inconsistencia. Finalmente, cada vez que se quiera asociar un fact en un slot de tipo Instance, deberá utilizarse su número ID. El funcionamiento de este tipo de slot requiere un poco más de cuidado por parte del experto que escribe las reglas. Principalmente porque no se puede garantizar que se cumpla el dominio de clases aceptado definido en Protégé-Frames sobre el slot de tipo Instance. El slot ID es agregado por MAMAS, por lo tanto, está reservado su uso.

4.3. Reglas

El experto debe saber manejar las dos herramientas, por un lado define las clases e instancias en Protégé-Frames y por el otro, escribe las reglas de inferencia en Jess. Por lo tanto, es necesario que tenga conocimiento sobre la escritura de reglas de Jess.

En el apartado 2.2.2 se explicó la base del funcionamiento de las reglas pero en este trabajo no se entrará en mucho detalle sobre las posibilidades que brinda Jess. Para aprender la escritura de reglas se recomienda el capítulo 7 en [3]. Pero si daremos algunos ejemplos reales de su uso.

La mayoría de los temas tocados en los apartados 4.1 y 4.2 dejan el camino bastante más definido hacia cuales son las condiciones y limitaciones al momento de escribir las reglas. Lo más importante que se desprende es posibilidad de trabajar directamente en Jess con los nombres de las clases, slots y los valores que tienen los slots definidos en Protégé-Frames. El sistema se encargará de realizar las conversiones y chequear las posibles inconsistencias entre ambas partes. Por ejemplo, no se puede definir un slot nuevo o cambiar un tipo de datos de un slot en una regla de Jess.

Recordando que a cada agente se le asigna un módulo de Jess, es una buena práctica definir las reglas respetando al módulo al que pertenecen. Para esto, se requiere que el nombre del módulo se escriba como prefijo en los nombres de las reglas y facts. En el caso de los AgentesOnto por defecto se focalizan siempre en su módulo de trabajo, en cambio, los AgentesPasa utilizan un módulo para almacenar sus reglas más dos módulos diferentes para generar los puentes. Por lo tanto, se recomienda siempre definir completamente todas las reglas.

Las reglas debe poseer un nombre y este debe ser único, es un requerimiento de Jess. Además, pueden poseer una descripción pero esta es opcional

y queda a criterio del experto.

Una cuestión muy importante es que solo se admite la declaración de nuevos facts (assert), pero no se pueden modificar (modify) ni eliminar (retract). Particularmente sobre este tema se explicarán las razones en el apartado 5.3.

4.4. Ejemplo

Antes de mostrar los ejemplos, presentaremos las ontologías modeladas y que se utilizaron a lo largo de este trabajo. Daremos una breve explicación de las mismas:

- **Mintzberg**[7]: Esta es un obra de referencia en lo que respecta a la teoría de las organizaciones. Nos permitió caracterizar lo que es una estructura organizacional así como las reglas que definen la evolución de una estructura. Las nociones clásicas de organigrama y de división del trabajo están ampliamente especificadas por los conceptos de, coordinación, flujo de intercambios e identificación de las diferentes partes que constituyen las organizaciones. La notaremos Mintzberg_1982.

- **Boissin**[8]: Este artículo se centra en el rol que cumple el líder de la empresa en la estrategia de crecimiento de la misma. Se presta especial atención al perfil del líder y a sus necesidades que se materializan a través de la organización de la empresa que él dirige hacia el crecimiento. Además, describe sobre todo los criterios para medir el crecimiento en relación con el vínculo entre líder y crecimiento. El crecimiento es un resultado para la empresa y un medio para su líder. La notaremos Boissin_2003.

- **Reyes**[9]: El objetivo del artículo es exponer las características de las empresas medianas. La tendencia actual es la definir las empresas en múltiples categorías. Ya no se tratan como empresas generalmente hablando sino por sus categorías, como por ejemplo, Microempresas, Pequeñas Empresas, Medianas Empresas y Grandes Empresas. En estas condiciones y para poder distinguir estos grupos, es necesario buscar las particularidades las componen para constituir estos conjuntos. El autor comienza esta tarea definiendo en primer lugar las medianas empresas. Intenta mostrar que éstas presentan características de las pequeñas y de las grandes empresas. Este tipo de empresas se desarrollan gracias a aptitudes empresariales originarias de las pequeñas

empresas, pero a su vez, se benefician de la lógica de gestión proveniente de las grandes empresas. La notaremos Reyes_2004.

- **Fiche**: Esta es una ontología de desarrollo propio para la entrada de información referida a la de descripción de la empresa. Corresponde a una ficha de identidad de la empresa que permite utilizarla como referencia. La notaremos Fiche_INSA2010.

4.4.1. Clases (Protégé-Frames)

Una vez presentado el formato que deberán seguir las ontologías mostraremos un ejemplo de un concepto correctamente modelado. En la figura 2 siguiente, se ve la interfaz gráfica de Protégé-Frames con la que deberá trabajar el experto para poder modelar la ontología que conoce.

En este ejemplo se ven varias de las características más comunes que se utilizarán en nuestras ontologías. En particular, mostraremos la clase entreprise de la ontología Reyes.

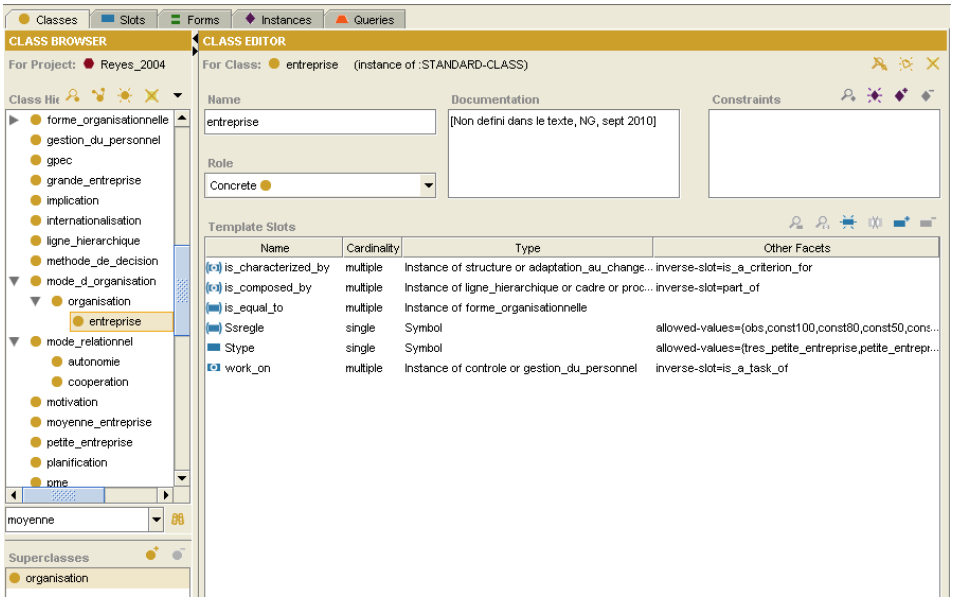

Figura 2: Clase *entreprise* en Protégé-Frames

A la izquierda, en "Class Browser", se puede visualizar la jerarquía de clases y debajo las superclases de la clase en cuestión. Se cumple que el nombre solo posee los caracteres permitidos. Sólo posee una superclase, por

lo tanto, hereda de un solo padre organisation y a su vez, organisation hereda de mode_d_organisation.

A la derecha, en "Class Editor", se define que es una clase Concreta y en "Documentation" hay una descripción textual del concepto.

En "Template Slots" se listan los slots que tiene el concepto. El conjunto de slots está conformado por {is_characterized_by, is_composed_by, is_equal_to, Ssregle, Stype, work_on}. El subconjunto {is_characterized_by, is_composed_by, is_equal_to, Ssregle} es heredado de las superclases y el subconjunto {Stype, work_on} es propio de la clase.

Los slots de tipo Instance son los únicos soportados con cardinalidad múltiple, el resto son single. Por ejemplo, el slot is_composed_by tiene como dominio aceptado el siguiente conjunto {organisation, moyenne_entreprise, petite_entreprise, pme, structure, effectif}, pero hay que recordar que durante la inferencia no se puede garantizar esta restricción.

Por último, las únicas facetas transferidas a Jess son allowed-values y default-value. Por ejemplo, el slot Stype tiene el conjunto permitido de valores {tres_petite_entreprise, petite_entreprise, grande_entreprise, moyenne_entreprise}.

Para aprender en más profundidad la utilización de la interfaz gráfica de Protégé-Frames se recomienda comenzar con la guía publicada en la sección de documentación del sitio oficial de Protégé[6].

4.4.2. Templates (Jess)

Si bien no es necesario que el experto conozca cómo se compone un template de Jess, es bueno saberlo para tener una idea más completa de cómo el conocimiento es manipulado y de cómo se puede expresar. Los AgentOnto se encargan de crear los templates utilizando directamente la API de Jess, pero la forma clásica para definirlos es mediante un archivo de texto. Como el lenguaje está basado en CLIPS hereda de él la sintaxis al estilo LISP. El formato de un template de Jess es el siguiente:

```
(deftemplate template-name
    [extends template-name]
    ["Documentation comment"]
    [(declare  (slot-specific TRUE | FALSE)
               (backchain-reactive TRUE | FALSE)
               (from-class class name)
```

```
            (include-variables TRUE | FALSE)
            (ordered TRUE | FALSE))]
    (slot | multislot slot-name
        ([(type ANY | INTEGER | FLOAT | NUMBER |
     SYMBOL | STRING | LEXEME | OBJECT | LONG)]
        [(default default value)]
        [(default-dynamic expression)]
        [(allowed-values expression+)])*)
```

Continuando con el ejemplo del apartado anterior, la clase entreprise se definiría internamente en MAMAS de la siguiente manera:

```
(deftemplate REYES::entreprise
  extends REYES::organisation
  (slot Stype
    (type SYMBOL)
    (allowed-values tres_petite_entreprise
      petite_entreprise grande_entreprise moyenne_entreprise))
    (multislot work_on
    (type INTEGER))
  (slot Name
    (type STRING))
  (slot ID
    (type INTEGER)
    (default -1)))
```

A partir de este ejemplo se desprenden algunas cuestiones,

- Jess soporta herencia y se nota con la palabra extends. El resto de los slots faltantes se heredan. Pero se requiere primeramente que el template organisation este bien definido.
- El nombre del módulo en donde se alojará el template y sus facts se definen como prefijo del nombre del template sucedido por doble dos puntos (::).
- Se introduce además el slot Name para poder alojar allí el nombre de la Instancia de Protégé-Frames y el slot ID como identificador del fact en el sistema. ID tiene por defecto el valor -1 para detectar una inconsistencia en la escritura de las reglas, porque todos los ID deben ser positivos y asignados al momento de crearse el fact.

- Se pueden visualizar tres tipos de slots diferentes y las dos únicas facetas soportadas, allowed-values y default.

4.4.3. Reglas (Jess)

El siguiente es un breve ejemplo de una regla aplicada a la ontología Mintzberg_1982,

```
(defrule Mintzberg_1982::134_condition_structure_simple
  (Mintzberg_1982::stabilite_de_l__environnement (Sstabilites dynamique))
  (Mintzberg_1982::complexite_de_l__environnement (Scomplexites simple))
  (Mintzberg_1982::systeme_technique (Scomplexites simple))
  (Mintzberg_1982::age_de_l__organisation (Staille faible))
  (not (Mintzberg_1982::regulation_du_systeme_technique))
  (not (Mintzberg_1982::structure_simple))
  =>
  (assert (Mintzberg_1982::structure_simple (ID (getid)))))
```

En el ejemplo, para que la regla se active deben existir al menos un fact de stabilite_de_l_environnement con el slot Sstabilites con el valor dynamique. Si el resto de los slots poseen algún valor es indistinto, ya que solo es relevante el slot Sstabilites. De igual manera para las tres condiciones siguientes, deben existir los facts con sus slots con dichos valores. La quinta condición posee un not, evalúa a verdadero si no existe ningún fact de regulation_du_systeme_technique. Las seis condiciones que hay en el LHS deben ser todas verdaderas para que se active la regla, ya que implícitamente estas condiciones conforman una conjunción entre ellas.

En el RHS está la acción que se debe cumplir una vez activada la regla. En este caso hay un assert. El assert crea un nuevo fact dentro de la memoria de trabajo del motor, en particular se desea crear un nuevo fact de structure_simple dentro del módulo Mintzberg_1982. Al momento de realizar un assert el motor Jess verifica si ya existe un fact igual al que se quiere crear. Como en nuestro sistema tenemos un slot ID que siempre posee un valor único global para todos los facts, nunca existirán dos facts iguales estrictamente hablando. Semánticamente el ID es irrelevante dentro de las ontologías, por lo tanto, es necesario agregar la sexta condición en el ejemplo. Esta verifica que no exista un fact semánticamente equivalente al que queremos agregar dentro de la memoria de trabajo sin tener en cuenta el slot ID. Si no se agrega esta condición, posiblemente se repetirán los facts

dificultando después la lectura de los resultados y en el peor de los casos se podrán crear bucles entre diferentes reglas.

Jess provee muchas herramientas para definir precondiciones más complejas, como por ejemplo la utilización de variables, funciones, etc. Pero esto queda a criterio y necesidad del experto.

5. Aspectos de investigación

A lo largo de este trabajo se produjeron varias marchas y contramarchas, es por eso que se debieron tomar varias decisiones claves para seguir adelante. En este apartado se explicarán cuales fueron, las decisiones que se tomaron y su por qué.

5.1. Protégé-OWL

En el proyecto MAEOS inicialmente se pensaba representar las ontologías en Protégé-OWL[4], pero en los comienzos de este trabajo se realizó un cambio por Protégé-Frames. Esta elección se debió a varias cuestiones. Primero, Protégé-Frames es posiblemente más simple (para los expertos que no tengan buenos conocimientos de lógica) modelar conceptos y principalmente propiedades. De la misma manera se había pensado en la utilización de reglas en SWRL[5] (Semantic Web Rule Language) basado en OWL[6] (Ontology Web Language). Pero al estar basado en OWL se descartó. En cambio, Jess comparte muchas semejanzas con Protégé-Frames en la representación del conocimiento, sumado a que ambos están implementados en JAVA. Por lo tanto, al final se optó por este motor de inferencia.

Protégé-Frames en nuestro caso es suficientemente expresivo para cubrir las necesidades del modelado de las ontologías pertenecientes al proyecto. Además, las ontologías utilizadas dentro proyecto MAEOS se asume que están en un mundo cerrado a diferencia OWL. Otra cuestión importante es que posee un editor con una interfaz gráfica amigable para que el experto que se encarga de realizar la modelización se abstraiga de cuestiones de implementación, y por el otro, brinda al desarrollador una completa y documentada API para acceder y modificar las ontologías.

5.2. Reglas de equivalencia semántica

Los agentes AgenteOnto trabajan de forma independiente, abocados al área que conocen, tratando de solucionar el problema principal por si solos sin tener ningún tipo de contacto con el resto de los expertos. Asimismo, las

[4]Sitio web de Protégé-OWL: `http://protege.stanford.edu/overview/protege-owl.html`

[5]Sitio web de SWRL: `http://www.w3.org/Submission/SWRL/`

[6]Sitio web de OWL: `http://www.w3.org/TR/owl-features/`

ontologías que manipulan generalmente difieren en la forma en que fueron modeladas entre un agente y otro. Esto quiere decir que los agentes no pueden utilizar otra ontología para poder inferir conocimiento. A pesar de esto, suelen existir conceptos de diferentes ontologías los cuales comparten similitudes o conceptualmente son equivalentes.

En el aparatado siguiente, se mostrará un extracto de la investigación sobre la combinación del conocimiento en el proyecto MAEOS. Luego, se explicará cómo se implementó y por último, unas observaciones finales.

5.2.1. Combinación de conocimiento en el proyecto MAEOS

Existen varias herramientas y trabajos sobre la combinación de ontologías [10][11][12][13]. En MAEOS son aplicables cuatro clases de métodos: Merging, Mapping, Aligning y Mediating. Todos estos métodos no pueden ser siempre aplicados de forma automática o sistemática, por lo tanto, la intervención de un experto es requerida[14].

La combinación de varias ontologías implica, al menos, la presencia de conceptos comunes o relacionados entre ellas. Se pueden aplicar diferentes criterios para identificar las similitudes entre dos conceptos[15]: las similitudes de los términos; la similitud de las propiedades; o las similitudes de las entidades que las contienen o que están conteniendo. Algunos métodos, por ejemplo, como el Aligning es mejor que el Merging cuando dos ontologías son complementarias o tienen diferentes niveles semánticos. Por lo tanto, ante la elección de un método deben conocerse muy bien sus limitaciones.

En situaciones reales, pueden surgir varios errores a diferentes niveles. Las disparidades en las definiciones no solo pueden suceder a nivel conceptual, de terminología o de taxonomía, sino también, a nivel sintáctico. Entre dos ontologías relacionadas, es común tener el mismo término con diferentes significados o varios términos refiriéndose al mismo concepto. Las disparidades entre ontologías son numerosas. Estas están resumidas en [10][16][17] con una serie de ejemplos.

Estas diferencias afectan la implementación de los métodos de combinación. El caso más extremo sucede cuando se desea combinar dos ontologías disjuntas, haciendo imposible la aplicación de algún método de combinación. En el caso de tener ontologías relacionadas, no se puede realizar una elección si el grado de similitud de varios términos es equivalente[18]. Incluso si se encuentran las conexiones entre conceptos, no hay garantías de que

éstas sean biyecciones. También pueden surgir conflictos a nivel semántico. Finalmente, la diferencia de granularidad entre ontologías puede resultar en la eliminación o agregación de conceptos. Es importante notar que el número de disparidades incrementa cuando las ontologías son más grandes. Por lo tanto, se deben analizar diferentes posibilidades para minimizar estas disparidades.

5.2.2. Implementación de las equivalencias

En este trabajo, se ha optado por implementar solamente las equivalencias semánticas que son biyecciones, y además, donde el nivel de abstracción de las definiciones de los términos es equivalente.

Por ejemplo, consideremos el concepto entreprise, que es central en nuestro proyecto. En la ontología Fiche_INSA2010, tiene solamente dos atributos, el nombre y un atributo especifico que indica que este concepto fue observado en el caso que se está tratando. En cambio, en la ontología Reyes_2004, el mismo concepto tiene varios atributos type (que puede tomar los valores, {tres_petite_entreprise, petite_entreprise, moyenne_entreprise, grande_entreprise}), work_on, is_characterized_by, is_composed_by y el atributo característico para indicar que este concepto fue observado. Vemos que el nivel de detalle en la descripción no es el mismo. La empresa en Fiche_INSA2010 está mucho menos detallada que la empresa de Reyes_2004.

Es por esta razón, que hemos decidido definir una "equivalencia semántica" (que deberíamos llamar correspondencia semántica para ser rigurosos) que dice que si el usuario instancia el concepto entreprise en Fiche_INSA2010, automáticamente podemos instanciar el concepto entreprise en Reyes_2004 (aunque sin detallar el tipo, por ejemplo). En cambio, si el usuario instancia entreprise en Reyes_2004, no tiene sentido instanciar entreprise de Fiche_INSA2010, ya que en ese caso, parte de la riqueza del concepto en la ontología Reyes_2004 se perdería. La regla Jess que implementa esta "correspondencia semántica" es:

```
(defrule PAS_REYE_FICH::10_semantic_equivalence_9 "NG,_janvier_2011"
  (Fiche_INSA2010::nombre_de_salaries {Name != PAS_REYE_FICH_se9}
    (Ssregle ?x) (Snombre ?y))
  (not (Reyes_2004::effectif (Name PAS_REYE_FICH_se9)
    (Ssregle ?x) (Snombre ?y)))
  =>
```

```
  (assert (Reyes_2004::effectif (Name PAS_REYE_FICH_se9)
    (Ssregle ?x) (Snombre ?y) (ID (getid)))))
```

En cambio, el concepto dirigeant de Reyes_2004 es realmente semánticamente equivalente al concepto chef_d_entreprise de Fiche_INSA2010, por esa razón, vamos a incluir dos reglas para establecer la biyección en cuestión.

```
(defrule PAS_REYE_FICH::7_semantic_equivalence_7 "NG,_janvier_2011"
  (Reyes_2004::dirigeant {Name != PAS_REYE_FICH_se7}
    (Ssregle ?x) (Snom ?y))
  (not (Fiche_INSA2010::chef_d_entreprise (Name PAS_REYE_FICH_se7)
    (Ssregle ?x) (Snom ?y)))
  =>
  (assert (Fiche_INSA2010::chef_d_entreprise (Name PAS_REYE_FICH_se7)
    (Ssregle ?x) (Snom ?y) (ID (getid)))))

(defrule PAS_REYE_FICH::8_semantic_equivalence_7 "NG,_janvier_2011"
  (Fiche_INSA2010::chef_d_entreprise {Name != PAS_REYE_FICH_se7}
    (Ssregle ?x) (Snom ?y))
  (not (Reyes_2004::dirigeant (Name PAS_REYE_FICH_se7)
    (Ssregle ?x) (Snom ?y)))
=>
(assert (Reyes_2004::dirigeant (Name PAS_REYE_FICH_se7)
    (Ssregle ?x) (Snom ?y) (ID (getid)))))
```

5.2.3. Observaciones

Lograr esta equivalencia semántica fue uno de nuestros objetivos a conseguir. En una primera instancia se pensaba manipular directamente los facts desde la API de Jess. Buscando los facts de la equivalencia en cada una de las ontologías y luego creando su reciproco en la otra ontología. Pero teniendo en cuenta la facilidad que se pueden escribir reglas en Jess se optó por esta solución.

El slot Name, como ya se comentó anteriormente, no pertenece originariamente a las ontologías. Pero nos resulta útil para evitar un problema que surge al escribir la equivalencia a partir de dos implicancias. Supongamos que tenemos un fact de la equivalencia en una ontología A y se ejecuta una regla definiendo el fact en la ontología B. Ahora bien, este nuevo fact en B activará la otra regla definiendo un fact en A. Para evitar la definición de este segundo fact innecesario se fijo un nombre que identifica cual fue la

regla que lo creo. Ahora bien, cada vez que se vaya a ejecutar la regla se verifica que este fact no haya sido definido a partir de las propias reglas de equivalencia. En el ejemplo se puede ver en el LHS, Name debe ser distinto a PAS_REYE_FICH_se7 {Name != PAS_REYE_FICH_se7} y en el RHS, se le asigna el Name PAS_REYE_FICH_se7.

Se agrega la segunda condición en el LHS de ambas reglas en el objetivo verificar que no exista un fact equivalente en la base de conocimiento obviando el slot ID (recordar la explicación del apartado 4.4.3).

5.3. Modify y Retract

A lo largo de este trabajo siempre se nombra la definición y no la modificación o remoción de facts. Esto se debe a varias cuestiones:

- La primera y principal, es porque es una política a seguir en este trabajo. El hecho de que siempre se cree nuevo conocimiento y a partir de eso seguir infiriendo lo mas que se pueda. Se ajusta al tipo de problema que se está manipulando y no es necesario que las reglas modifiquen o borren facts.

- Entra en juego lo que significa conceptualmente borrar un fact. Supongamos que tenemos una regla R1, que al ejecutarse se crean los facts A, B y C. Luego, la ejecución de una regla R2 decide que se debe borrar C, por ende, la ejecución previa de la regla R1 queda inconsistente. Los facts A y B que lo "acompañaban" quedarán posiblemente disociados. Además, podría ser el caso de que exista un fact D este compuesto por C, por lo tanto, se debería decidir qué hacer con D. Entonces el borrado de un fact es una tarea no trivial y posiblemente a partir de borrado puede comenzar a generarse nuevo conocimiento inconsistente.

- La modificación de un fact es similar a su borrado (con lo problemas que acarrea) seguido de una definición de un fact. Por ende, puede volver a lanzar nuevamente todas las reglas que dependan de él. Generando posiblemente nuevos facts que sean contradictorios con los facts inferidos antes de ser modificado.

- Como tenemos varias estrategias de control (ver apartado siguiente), en particular la Simultánea trae aparejada varias cuestiones a tener

en cuenta al momento de decidir que facts finalmente quedan en el Blackboard y cuáles no. En esta estrategia, todos los agentes infieren y dejan sus resultados en una memoria intermedia. Puede suceder, que para un agente sea necesario borrar un fact que para el otro agente le es importante para generar una equivalencia semántica. Es aquí donde se debe decidir el control que facts a borrar y a priori el control no tiene tanto conocimiento sobre los datos. Por ende, debería tomar una decisión general previamente fijada.

Varios de estos puntos explicados pueden atacarse, pero se debe ser muy cuidadoso y escapa a este trabajo. Principalmente, por la política tomada y por que se está buscando obtener los primeros resultados con el sistema experto se decidió no tomar este camino por el momento.

5.4. Estrategias de Control

Una estrategia de control es un algoritmo que define como el sistema se va a comportar a lo largo de toda su ejecución. A partir del estado de los agentes (si es que pueden inferir o no) y una estrategia, el control decide cual es el próximo agente a ejecutarse.

En un principio el planteo de diferentes estrategias tenía como objetivo evitar problemas de starvation o deadlock de los agentes. Pero posteriormente en este trabajo se tomaron dos decisiones fundamentales que lograron minimizar estos problemas. La primera, es que tratar de obtener la mayor cantidad de conocimiento inferido por los agentes. Esto se debe a que son los primeros pasos en el desarrollo del sistema experto, entonces, es difícil a priori determinar qué fuente de conocimiento es más relevante para la solución de un determinado problema. Por lo tanto, se busca que todos los agentes que puedan inferir lo hagan y luego con los resultados obtenidos tratar de depurar las fuentes de conocimiento. La segunda, se debe a que los facts nunca son modificados o eliminados. Esto implica que el conocimiento en el Blackboard siempre estará disponible para que un agente pueda inferir en cualquier momento. Por ejemplo, si un agente tiene la posibilidad inferir a partir de un fact A y no es su turno aún para trabajar, en el ínterin hasta que le toque su turno el fact A no será borrado ni modificado, por lo tanto, esto garantiza que el agente podrá utilizarlo cuando sea su momento de inferir.

Por otro lado, las estrategias influyen en la cantidad de iteraciones y ejecuciones de los agentes. Si una base de conocimiento es relativamente grande influye fuertemente en tiempo de procesado. Principalmente, cuando más complejo es el LHS de las reglas más tiempo le requerirá al motor de Jess arribar a una solución. Las dos operaciones más costosas son la de la verificación de la precondición y la ejecución del agente.

Para mostrar el funcionamiento de las estrategias primero se explicarán algunos detalles. Como mencionamos anteriormente el Blackboard posee dos memorias, una intermedia y una principal. A la intermedia la llamaremos pool, que es donde se almacenan los denominados Nodes (nodos). Dependiendo la estrategia elegida, los nodos de la memoria intermedia se plasmaran en la memoria principal del Blackboard. Un nodo es un conjunto de facts generado por un agente, donde se internamente se discriminan los nuevos facts definidos y los ya existentes antes de inferir. Además, existe un conjunto denominado ready donde se almacenan temporariamente aquellos agentes que están listos para inferir.

Existen hasta el momento dos estrategias implementadas, la Secuencial y la Simultánea.

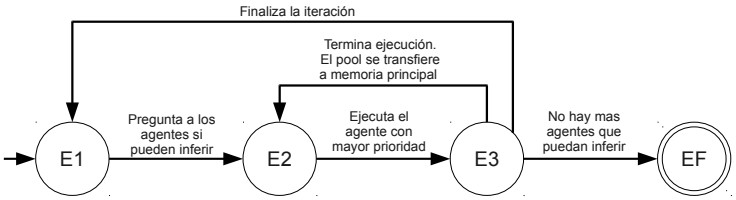

Fig. 3 Diagrama de estados estrategia Secuencial

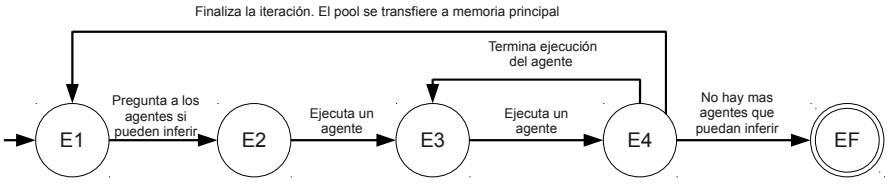

Fig. 4 Diagrama de estados estrategia Simultanea

Estados
Ex = (*Hay un agente ejecutandose*,
Ready vacío,
Pool vacío,
Existe al menos un agente que puede inferir)

E1 = (*NO*, *SI*, *SI*, *SI*)
E2 = (*NO*, *NO*, *SI*, *SI*)
E3 = (*SI*, *NO*, *NO*, *SI*)
E4 = (*NO*, *NO*, *NO*, *SI*)
EF = (*NO*, *SI*, *SI*, *NO*)

En las dos figuras, los círculos corresponden a estados del sistema y las flechas a transiciones entre dos estados. El círculo doble indica que es el

estado final. Los estados están definidos por una tupla de cuatro condiciones y la referencia está dada en Ex. En ambas figuras, E1 es el estado inicial indicado con una flecha sin origen.

5.4.1. Secuencial

En esta estrategia lo primero que hace es verificar a que agente se le cumple su precondición. Aquellos que se verifican son agregados al conjunto ready. Luego se va tomando de a un agente del conjunto teniendo en cuenta su prioridad, se lo ejecuta y el agente guarda el nodo resultante en el pool. Finalmente, el control acepta los facts enviándolos a memoria principal. Nuevamente, vuelve a tomar un agente del conjunto ready y repite el proceso. Cuando el conjunto ready se vacía, se vuelve al inicio a verificar que agentes pueden inferir. Si el conjunto tiene al menos un agente vuelve a realizar el procedimiento anterior, de caso contrario el sistema finaliza. La figura 3

En esta estrategia se tiene muy en cuenta la prioridad que posee el agente, ya que es la al final decide que agente infiere primero. Pero a la vez, evita que algún agente se quede sin inferir. La prioridad es un número entero positivo mayor que cero. A mayor número, mayor prioridad.

5.4.2. Simultánea

De la misma manera que la estrategia secuencial (ver figura 4), el primer paso es verificar a cada uno de los agentes cual puede inferir e ir agregándolos al conjunto ready. Luego, se ejecuta a cada uno de los agentes sin importar su prioridad pidiéndoles que guarden el nodo que generan en el pool. Pero a diferencia de la Secuencial, la transferencia de los facts desde la memoria intermedia a memoria principal no se hace inmediatamente después de la ejecución del agente. Si no que se espera a que todos los agentes infieran y dejen su nodo en el pool. Recién después que el conjunto ready este vacío, el control une todos los nodos y los guarda en la memoria principal del Blackboard. Nuevamente, comienza la iteración buscando agentes que puedan inferir y se repite el proceso. Cuando no existan más agentes que se puedan agregar al conjunto ready el sistema finaliza.

La unión de los nodos en este caso, consiste en unir todos los facts nuevos agregados de cada uno de los nodos y dejar sin tocar el conjunto de facts anteriores.

6. Un ejemplo de prueba

En este apartado se planteará la historia de Pierre Carle y el estado actual de su empresa. A partir de ella se extraerán los puntos más relevantes para modelar el problema utilizando las ontologías existentes. Luego, se analizaran los resultados producidos por MAMAS y finalmente se aprovechará el ejemplo para mostrar las diferencias entre las dos estrategias.

6.1. Planteo del problema: Electricité Services

Nombre de la empresa: Electricité Services (se abreviará como ES)

Dirección: 1 place de Haguenau, 67000, Estrasburgo

Teléfono de Pierre Carle: 06 06 06 06 06

Código APE: 4322B (número de Actividad Principal de la Empresa)

Código Sirte: 555666777 00088 (número de identificación único por establecimiento)

El señor Pierre Carle fue despedido de la empresa donde trabajaba en relación de dependencia. Con una vasta experiencia en el sector de los servicios eléctricos y con buena capacidad para prestar un servicio de alta calidad decidió iniciar un nuevo emprendimiento por cuenta propia y de forma autónoma. Finalmente, en el año 2003 fundó su propia empresa de servicios eléctricos bajo el nombre de "Electricité Services". La persona jurídica con la que fue establecida la compañía se denomina Empresa Unipersonal de Responsabilidad Limitada (EURL es un status vigente en la legislación francesa).

Para prestar un servicio de calidad tuvo como meta, brindar disponibilidad para ser ubicado en cualquier momento, dejar las obras en un estado impecable, ser puntual en las reuniones con los clientes, en caso contrario, avisar con anticipación ante un eventual atraso. Por estas razones, tuvo éxito desde el comienzo y obtuvo un volumen de negocios de 950.000 euros, con un crecimiento casi regular de 10 % anual.

La empresa hoy cuenta con 10 empleados, una secretaria, un responsable de obra, seis electricistas (de los cuales uno siguió a Carle desde su antiguo trabajo y dos fueron contratados el año pasado) y dos nuevos empleados que están en una etapa de aprendizaje.

Carle constata que más a menudo los clientes buscan mejores prestaciones por parte de su empresa, en particular, los encargos se orientan hacia

un trabajo de alta calidad pero a nivel hogareño. Los clientes buscan dos características muy diferentes. Primero, prestan mucha atención al acabado de trabajo realizado. Segundo, las solicitudes son más complejas e integran nuevos dominios en relación con el hogar y su automatización (domótica).

Actualmente, las solicitudes en el ámbito de la demótica se multiplican, y Carle intenta ubicar su empresa en este sector debido a que brinda un gran valor agregado. Al mismo tiempo, para poder cumplir los requerimientos hace presión sobre su responsable de obra para prestar una atención particular al seguimiento y la calidad de las obras en curso.

Por otro lado, Carle está desbordado por las tareas de gestión: obtención de nuevos clientes, elaboración de presupuestos, seguimiento de las obras en curso con su jefe de obra, seguimiento de la facturación y cobranza de las obras terminadas. Además, está dedicando mucho más tiempo a los nuevos clientes en el área de la domótica, ya que la mayoría de respuestas técnicas al cliente sobre un proyecto se realizan al momento de la elaboración del presupuesto. De esta manera, las otras obras potenciales quedan en espera al igual que el seguimiento de la facturación. Los problemas de tesorería comienzan a sentirse ya la cobranza del volumen de negocios no se lleva a cabo correctamente.

Desde el punto de vista de los electricistas, estos se quejan de tener al patrón sistemáticamente presionándolos, además del jefe de obra. Las órdenes son contradictorias a veces. Además, deben regularmente ayudar con su trabajo en una obra que no es la propia, en disminución de la calidad de servicio ante los clientes que el patrón cuida tanto.

A nivel de la calidad de servicio, dos observaciones recurrentes son hechas por el personal: por un lado, el patrón utiliza como argumento de venta justamente la calidad de servicio, los clientes prestan, en consecuencia, mucha atención a los acabados. Por ejemplo, se quejan sistemáticamente de la calidad del enduído, y a esto los electricistas responden “no es nuestro oficio”. Por otro lado, las nuevas obras son muy técnicas y a veces los electricistas se sienten desbordados por las preguntas de los clientes. Aunque los electricistas indican a los clientes que deben contactar el patrón para responder a sus preguntas, éste no se encuentra en general disponible para responderles en el momento en que la empresa interviene en la casa del cliente.

Del punto de vista de la asistente, ella también sufre por la presión por parte de su patrón. El la llama varias veces por día, dándole por teléfono

cada vez más cosas a realizar, siempre más urgentes que las tareas que le había encomendado anteriormente. En esta situación ella no puede organizar su trabajo correctamente y es algo que su patrón también le reprocha.

6.2. Descripción y explicación de la instanciación

A partir del texto del apartado anterior, en donde se detalla el estado de la empresa, se pudieron definir las instancias de las ontologías para modelar el problema. La presentación de los datos se harán primero detallando el lugar del texto o el motivo por el cual se define la instancia y luego la instancia detallada con un formato muy similar al de Jess. La notación es la siguiente:

```
(Ontología::nombre_del_concepto (slot1 valor1)
  (slot2 valor2) ... (slotn valorn))
```

En el caso de ser un slot de valor Instance, el valor o los valores estarán entre corchetes utilizando el Name de la instancia.

- El nombre de la empresa,

```
(Fiche_INSA2010::entreprise (Name Entreprise_ES)
  (Snom "Electricité Services"))
(Fiche_INSA2010::raison_sociale (Name Raison sociale_ES)
  (Snom "Electricité Services"))
```

- La dirección física de la empresa, ciudad y código postal,

```
(Fiche_INSA2010::adresse_de_facturation
  (Name adresse facturation_ES) (Sn_rue "1 place de Haguenau")
  (Scode_postal "67000") (Sville "Strasbourg"))
(Fiche_INSA2010::adresse (Name adresse_ES)
  (Sn_rue "1 place de Haguenau") (Scode_postal "67000")
  (Sville "Strasbourg"))
```

- El número de Actividad Principal de la Empresa,

```
(Fiche_INSA2010::code_ape_nf (Name APE-ES) (Sape "4322B")
```

- El número de identificación único por establecimiento,

```
(Fiche_INSA2010::siret (Name SIRET_ES) (Ssiret "555666777 00088")
```

- El sitio web de la empresa,

```
(Fiche_INSA2010::site_web (Name site web_ES) (Ssite_web "www.es.fr"))
```

- El año del establecimiento/creación de la empresa,

```
(Fiche_INSA2010::date_de_creation (Name date creation_ES)
  (Sdate 2003))
(Mintzberg_1982::age_de_l_organisation (Name age_ES))
```

- Pierre Carle es el empresario que lidera la empresa, agregando sus datos personales y de contacto,

```
(Fiche_INSA2010::contact (Name contact_ES)
  (Stelephone1 "06 06 06 06 06") (Snom "CARLE") (Sprenom "Pierre")
  (Semail "pierre.carle@es.fr"))
(Fiche_INSA2010::chef_d_entreprise (Name chef_d_entreprise_ES)
  (Stelephone1 "06 06 06 06 06") (Snom "CARLE") (Sprenom "Pierre")
  (Semail "pierre.carle@es.fr"))
(Boissin_2003::chef_d_entreprise (Name chef_d_entreprise)
  (Snom "Pierre CARLE"))
(Mintzberg_1982::directeur (Name directeur_ES)
  (Sevolution_future constant))
(Reyes_2004::dirigeant (Name dirigeant_ES) (Snom "Pierre CARLE")
  (is_characterized_by [tache_ES])
```

- Pierre Carle es propietario de su propia empresa,

```
(Fiche_INSA2010::constitution_du_capital
  (Name Constitution_capital_ES) (Sindependance TRUE))
(Reyes_2004::capital (Name capital_ES)
  (Spourcentage_d_independance cent))
(Reyes_2004::propriete (Name proprite_ES)
  (Scoalition_de_personnes_physiques non) (Sproprietaire_unique TRUE)
  (Spersonne_morale non))
```

- La persona jurídica con la que fue establecida la compañía,

```
(Fiche_INSA2010::statut_juridique (Name statut juridique_ES)
  (Sstatut eurl))
```

- El volumen de negocios actual,

```
(Fiche_INSA2010::chiffre_d_affaires_du_dernier_exercice
  (Name CA_ES) (Smontant 950000.0))
(Boissin_2003::chiffre_d_affaires (Name CA_ES))
(Reyes_2004::chiffre_d_affaires (Name CA_ES) (Smontant 950000.0))
```

- La cantidad total de empleados que posee la empresa en la actualidad,

```
(Fiche_INSA2010::nombre_de_salaries (Name Nb_de_salaries_ES)
  (Snombre 10.0))
(Reyes_2004::effectif (Name effectif_ES) (Snombre 10.0)
  (is_composed_by [temps_partiel, Permanents_ES, saisonnier_ER]))
```

- El Sr Carle es el único que piensa en el posicionamiento en el mercado de la empresa,

```
(Boissin_2003::elaboration_de_la_strategie (Name strategie_ES)
  (Sacteur [chef_d_entreprise]) (Sdegre_de_centralisation centralisee))
(Boissin_2003::entreprise (Name Entreprise_ES))
(Mintzberg_1982::elaboration_de_la_strategie (Name strategie_ES)
  (Sdegre_de_centralisation centralisee))
```

- La actividad que realiza la empresa, la electricidad, es considerada como un trabajo artesanal. Se define artesanal porque: el líder está calificado en su profesión, es el jefe de la empresa e independiente que asume solo o con su esposo/a la responsabilidad de todas las funciones de la empresa (producción, gestión y comercialización).

```
(Boissin_2003::dirigeant (Name dirigeant_ES) (Sartisan TRUE))
```

- Como existe un lider en la empresa,

```
(Mintzberg_1982::cadre (Name cadre_ES) (Sevolution_future constant))
```

- Como existe un jefe de obra,

```
(Mintzberg_1982::cadre_intermediaire (Name cadre_intermediare_ES)
  (Sevolution_future constant) (part_of [ligne_hierarchique_ES]))
```

- Como existe un responsable de obra,

```
(Mintzberg_1982::ligne_hierarchique (Name ligne hierarchique_ES)
  (is_composed_by [cadre_intermediare_ES]) (Sdegre faible))
(Reyes_2004::ligne_hierarchique (Name ligne hierarchique_ES)
  (Ssimplicite TRUE))
```

- La evolución del tamaño de la organización,

```
(Mintzberg_1982::taille_de_l_organisation (Name Taille_ES)
  (Sevolution_future croissant))
```

- La forma en que el patrón da las directivas,

```
(Mintzberg_1982::communication_informelle
  (Name communication informelle_ES))
```

- La descomposición del grupo de empleados según el tipo de contrato efectivo,

```
(Reyes_2004::employe_a_temps_partiel (Name temps_partiel)
  (Snombre 0.0) (part_of [effectif_ES]))
(Reyes_2004::employe_permanent (Name Permanents_ES)
  (Snombre 8.0) (part_of [effectif_ES]))
(Reyes_2004::saisonnier (Name saisonnier_ES) (Snombre 0.0)
  (part_of [effectif_ES]))
```

- Las nuevas tareas son siempre más urgentes que las tareas actuales o precedentes,

```
(Reyes_2004::tache (Name tache_ES) (Shorizon_temporel court_terme)
  (Schangement_volume_futur alourdissement)
  (is_a_criterion_for [dirigeant_ES]))
```

- El producto ofrecido se divide en dos dominios muy diferentes, el trabajo de electricidad clásico y la domótica.

```
(Reyes_2004::produit (Name produit_ES) (Snombre 2.0))
```

6.3. Análisis de los resultados

A continuación, se expondrá el informe realizado por el experto sobre el análisis de los resultados del caso ES. La notación entre corchetes indica el concepto instanciado que dio lugar a las conclusiones escritas. El análisis consta de diferentes dimensiones en relación con:

El entorno

ES parece evolucionar en un entorno [environnement] más bien simple e inestable. Esto parece confirmarse por la caracterización de la actividad

que realiza, e igualmente por la caracterización de la evolución necesaria en relación con las necesidades cambiantes de los consumidores.

La organización

El análisis pone en evidencia el hecho de que los comportamientos son formalizados [formalisation des comportement]: esto representa la manera que tiene la organización de limitar el margen de maniobra de sus miembros. Esto pasa por la estandarización, de las calificaciones en primer lugar [standardisation des qualifications], necesaria para una coordinación real y eficaz de todos los gremios asociados a su tarea; de los resultados [standardisation des résultats], necesaria para soportar el hecho que la línea jerárquica [ligne hiérarchique] sea un poco débil con el jefe de la empresa sumergido por tareas que se hacen cada vez más pesadas. No obstante, la empresa deja poca autonomía [autonomie] a su personal, apareada, de hecho, a una limitada capacidad de anticipación [anticipation].

Va a ser necesario rápidamente que el marco del trabajo realizado por el jefe de la empresa se alivie [délégation – évolution future = croissante], en provecho de una verdadera gestión del crecimiento de la empresa; que es efectiva, pero que el dirigente debe considerar como propia [croissance – acteur = dirigeant]. La capacidad de la empresa a responder rápidamente a las evoluciones del entorno [vitesse de réponse] va deber acoplarse con una capacidad interna para adaptarse a los cambios [adaptation au changement] y a una fuerte flexibilidad [flexibilité].

La empresa se beneficia, y es uno de sus puntos fuertes teniendo en cuenta su contexto, de una motivación [motivation] y de una participación [implication] fuerte de su personal. Sin embargo, la comunicación es principalmente informal [communication informelle], lo que no es molesto en sí, salvo por el hecho que la comunicación tiene tendencia a deteriorarse [communication - détérioration].

La estrategia

Habíamos observado que la elaboración de la estrategia estaba centralizada [élaboration de la stratégie]. Parecería también que sea intuitiva, basada solamente en la experiencia del jefe de la empresa. Esto parece confirmarse por el déficit de métodos de decisión [méthode de décision] y la necesidad de implementar, al menos, las primeras herramientas simples de auditoría de la performance [audit des performances].

Estos diferentes elementos hacen oscilar la empresa ES entre una estruc-

tura simple, es decir, una pequeña empresa de realización de obras eléctricas ubicada bajo la autoridad de un dirigente jefe de orquesta competente, y una adhocracia, es decir, una pequeña empresa especializada en profesiones de punta, disponiendo de personal calificado y autónomo, capaces de tomar a cargo las solicitudes de una clientela exigente.

6.4. Diferentes estrategias de control

Al momento de correr este problema con el sistema, se utilizaron las dos estrategias implementadas para visualizar las diferencias. Partiendo del hecho de que los facts siempre estarán disponibles hasta el final de la ejecución de MAMAS los resultados obtenidos son equivalentes. La cantidad y los valores resultantes de ambas estrategias fueron los mismos, en total se dedujeron 349 facts a partir de 38 facts iniciales. Las principales diferencias fueron en qué momento se obtuvieron los resultados parciales y la cantidad de ejecuciones de los agentes en total. La cantidad de ejecuciones influyen directamente sobre la velocidad del sistema. Sobre la calidad de la solución, se dejará la opinión del experto en las conclusiones finales.

Como se comentó anteriormente, las dos operaciones más costosas de los agentes son la verificación de su precondición y la ejecución propiamente dicha. Esto se debe a dos cuestiones. La primera es porque en ambos casos se lanza el motor de inferencia. Si bien la verificación de la precondición busca que se active al menos una regla es una tarea costosa si existen centenas de facts. La segunda se debe a cuestiones de implementación interna de los agentes. Es necesario guardar el estado del motor de inferencia antes de ejecutarlo para luego volver a ese estado por cualquier problema que pueda surgir. Guardar el estado del motor es prácticamente hacer una copia total del objeto, su memoria de trabajo y la agenda. Como usuarios del motor no podemos manipular sus datos internos. Por lo tanto, seguido de la ejecución de la verificación de la precondición debemos volver al estado anterior como si nunca se hubiese definido algún fact. Como acotación, fue necesario aumentar el tamaño del stack que viene por defecto en la maquina virtual de Java por la gran cantidad de llamadas recursivas que necesitaban para cargar y guardar el estado del motor.

Volviendo al ejemplo de este capítulo, la cantidad de datos de entrada es la siguiente:

```
Total de Deftemplate definidos para la ontologia Mintzberg_1982 : 380
```

```
Total de Defrules definidos para la ontologia Mintzberg_1982 : 160
Total de Facts definidos para la ontologia Mintzberg_1982 : 8
Total de Deftemplate definidos para la ontologia Fiche_INSA2010 : 16
Total de Defrules definidos para la ontologia Fiche_INSA2010 : 0
Total de Facts definidos para la ontologia Fiche_INSA2010 : 14
Total de Deftemplate definidos para la ontologia Reyes_2004 : 76
Total de Defrules definidos para la ontologia Reyes_2004 : 79
Total de Facts definidos para la ontologia Reyes_2004 : 11
Total de Deftemplate definidos para la ontologia Boissin_2003 : 20
Total de Defrules definidos para la ontologia Boissin_2003 : 20
Total de Facts definidos para la ontologia Boissin_2003 : 5
Total de Defrules definidos para la ontologia Reyes_2004_Boissin_2003 : 14
Total de Defrules definidos para la ontologia Boissin_2003_Fiche_INSA2010 : 8
Total de Defrules definidos para la ontologia Reyes_2004_Fiche_INSA2010 : 12
Total de Defrules definidos para la ontologia Reyes_2004_Mintzberg_1982 : 48
Total de Defrules definidos para la ontologia Mintzberg_1982_Fiche_INSA2010 : 4
Total de Defrules definidos para la ontologia Mintzberg_1982_Boissin_2003 : 8
```

El orden de ejecución de las estrategias es (lo que si visualiza es una salida típica de MAMAS):

Simultanea

```
Comenzado Inferencia
Iteracion 1 | 9 agentes que pueden inferir
Trabajando: Mintzberg_1982
Mintzberg_1982 disparo 4 reglas
Trabajando: Reyes_2004
Reyes_2004 disparo 47 reglas
Trabajando: Boissin_2003
Boissin_2003 disparo 4 reglas
Trabajando: Reyes_2004_Boissin_2003
Reyes_2004_Boissin_2003 disparo 9 reglas
Trabajando: Boissin_2003_Fiche_INSA2010
Boissin_2003_Fiche_INSA2010 disparo 9 reglas
Trabajando: Reyes_2004_Fiche_INSA2010
Reyes_2004_Fiche_INSA2010 disparo 9 reglas
Trabajando: Reyes_2004_Mintzberg_1982
Reyes_2004_Mintzberg_1982 disparo 10 reglas
Trabajando: Mintzberg_1982_Fiche_INSA2010
Mintzberg_1982_Fiche_INSA2010 disparo 3 reglas
Trabajando: Mintzberg_1982_Boissin_2003
Mintzberg_1982_Boissin_2003 disparo 6 reglas

Iteracion 2 | 8 agentes que pueden inferir
Trabajando: Mintzberg_1982
Mintzberg_1982 disparo 59 reglas
Trabajando: Reyes_2004
Reyes_2004 disparo 1 reglas
Trabajando: Boissin_2003
```

```
Boissin_2003 disparo 11 reglas
Trabajando: Reyes_2004_Boissin_2003
Reyes_2004_Boissin_2003 disparo 6 reglas
Trabajando: Boissin_2003_Fiche_INSA2010
Boissin_2003_Fiche_INSA2010 disparo 2 reglas
Trabajando: Reyes_2004_Fiche_INSA2010
Reyes_2004_Fiche_INSA2010 disparo 8 reglas
Trabajando: Reyes_2004_Mintzberg_1982
Reyes_2004_Mintzberg_1982 disparo 25 reglas
Trabajando: Mintzberg_1982_Boissin_2003
Mintzberg_1982_Boissin_2003 disparo 3 reglas

Iteracion 3 | 7 agentes que pueden inferir
Trabajando: Mintzberg_1982
Mintzberg_1982 disparo 17 reglas
Trabajando: Boissin_2003
Boissin_2003 disparo 4 reglas
Trabajando: Reyes_2004_Boissin_2003
Reyes_2004_Boissin_2003 disparo 1 reglas
Trabajando: Boissin_2003_Fiche_INSA2010
Boissin_2003_Fiche_INSA2010 disparo 5 reglas
Trabajando: Reyes_2004_Fiche_INSA2010
Reyes_2004_Fiche_INSA2010 disparo 1 reglas
Trabajando: Reyes_2004_Mintzberg_1982
Reyes_2004_Mintzberg_1982 disparo 12 reglas
Trabajando: Mintzberg_1982_Boissin_2003
Mintzberg_1982_Boissin_2003 disparo 4 reglas

Iteracion 4 | 3 agentes que pueden inferir
Trabajando: Reyes_2004_Boissin_2003
Reyes_2004_Boissin_2003 disparo 4 reglas
Trabajando: Reyes_2004_Fiche_INSA2010
Reyes_2004_Fiche_INSA2010 disparo 2 reglas
Trabajando: Reyes_2004_Mintzberg_1982
Reyes_2004_Mintzberg_1982 disparo 2 reglas

Iteracion 5 | 2 agentes que pueden inferir
Trabajando: Reyes_2004_Mintzberg_1982
Reyes_2004_Mintzberg_1982 disparo 1 reglas
Trabajando: Mintzberg_1982_Boissin_2003
Mintzberg_1982_Boissin_2003 disparo 1 reglas

Iteracion 6 | 0 agentes que pueden inferir
Inferencia finalizada
```

Secuencial

```
Comenzado Inferencia
Iteracion 1 | 9 agentes que pueden inferir
Trabajando: Reyes_2004_Boissin_2003
Reyes_2004_Boissin_2003 disparo 9 reglas
Trabajando: Boissin_2003_Fiche_INSA2010
```

```
Boissin_2003_Fiche_INSA2010 disparo 10 reglas
Trabajando: Reyes_2004_Fiche_INSA2010
Reyes_2004_Fiche_INSA2010 disparo 15 reglas
Trabajando: Reyes_2004_Mintzberg_1982
Reyes_2004_Mintzberg_1982 disparo 12 reglas
Trabajando: Mintzberg_1982_Fiche_INSA2010
Mintzberg_1982_Fiche_INSA2010 disparo 3 reglas
Trabajando: Mintzberg_1982_Boissin_2003
Mintzberg_1982_Boissin_2003 disparo 8 reglas
Trabajando: Mintzberg_1982
Mintzberg_1982 disparo 63 reglas
Trabajando: Reyes_2004
Reyes_2004 disparo 48 reglas
Trabajando: Boissin_2003
Boissin_2003 disparo 15 reglas

Iteracion 2 | 5 agentes que pueden inferir
Trabajando: Reyes_2004_Boissin_2003
Reyes_2004_Boissin_2003 disparo 7 reglas
Trabajando: Boissin_2003_Fiche_INSA2010
Boissin_2003_Fiche_INSA2010 disparo 4 reglas
Trabajando: Reyes_2004_Fiche_INSA2010
Reyes_2004_Fiche_INSA2010 disparo 5 reglas
Trabajando: Reyes_2004_Mintzberg_1982
Reyes_2004_Mintzberg_1982 disparo 35 reglas
Trabajando: Mintzberg_1982_Boissin_2003
Mintzberg_1982_Boissin_2003 disparo 5 reglas

Iteracion 3 | 4 agentes que pueden inferir
Trabajando: Reyes_2004_Boissin_2003
Reyes_2004_Boissin_2003 disparo 2 reglas
Trabajando: Boissin_2003_Fiche_INSA2010
Boissin_2003_Fiche_INSA2010 disparo 2 reglas
Trabajando: Mintzberg_1982
Mintzberg_1982 disparo 17 reglas
Trabajando: Boissin_2003
Boissin_2003 disparo 4 reglas

Iteracion 4 | 3 agentes que pueden inferir
Trabajando: Reyes_2004_Boissin_2003
Reyes_2004_Boissin_2003 disparo 2 reglas
Trabajando: Reyes_2004_Mintzberg_1982
Reyes_2004_Mintzberg_1982 disparo 3 reglas
Trabajando: Mintzberg_1982_Boissin_2003
Mintzberg_1982_Boissin_2003 disparo 1 reglas

Iteracion 5 | 0 agentes que pueden inferir
Inferencia finalizada
```

Como se puede ver, la cantidad de iteraciones difieren en una sola unidad. Pero lo importante son la cantidad de ejecuciones. En total la Simultánea

realizó 29 ejecuciones y 60 verificaciones de la precondición de los agentes. En cambio, la Secuencial realizó 21 ejecuciones en total y 50 verificaciones de precondición. Además, el tiempo ejecución desde el comienzo hasta el final de la Simultánea fue de 25 segundos contra 20 segundos de la Secuencial. La Simultánea resultó en este caso aproximadamente un 25 % más lenta.

Los valores mostrados son figurativos, simplemente se trata de mostrar en un ejemplo bastante grande las diferencias que suceden en ambas estrategias. Por el algoritmo de la estrategia Secuencial usualmente va a tender a tener una mejor respuesta que la Simultánea. Esto se debe a que en la estrategia Secuencial no se debe esperar a que termine la iteración para visualizar los cambios en el Blackboard. Los cambios producidos por un agente pueden ser rápidamente tomados por otro agente en la misma iteración. La Simultánea esta pensada a futuro, cuando se tengan que tomar decisiones entre varias soluciones parciales o cuando se realicen modificaciones y eliminaciones de facts.

7. Conclusiones y planes futuros

En este trabajo se presentaron los primeros resultados de un sistema experto que utiliza conocimiento heterogéneo para la ayuda de los consultores en el análisis y diagnostico de las PyMEs. Tanto al momento de iniciar como también a lo largo de todo el desarrollo del sistema fueron surgiendo varios problemas, marchas y contramarchas. Esto llevó a tomar varias decisiones claves para lograr su buen funcionamiento. A la par del desarrollo de la aplicación, el conocimiento estaba siendo modelado por expertos en distintas áreas a partir de la bibliografía. Por lo tanto, ellos también sufrían en diferente medida los continuos cambios en el diseño.

El ejemplo de prueba del capítulo 6 es uno de los primeros problemas grandes en el cual se utilizan múltiples ontologías y los puentes semánticos propuestos en este trabajo. Según el experto en estrategias y gestión, que se encargó en el modelado del conocimiento y del problema, los resultados obtenidos con el sistema fueron buenos salvo por una particularidad. Esto es una buena noticia, debido a que no se dedujeron conclusiones erróneas o falsas y los resultados estaban dentro de lo que se esperaba. Con respecto a la particularidad que le llamó la atención al experto fue la conclusión del último párrafo del aparatado 6.3. En donde dice que la empresa pasa a ser de una estructura simple a una adhocracia. Esta conclusión a priori no es una respuesta natural para el experto, pero analizando el caso globalmente no aparenta ser ilógico y fue por eso que se incluyó finalmente en el informe. Esta particularidad, es un ejemplo de lo que se estaba buscando con el proyecto.

A partir de esta experiencia se puede remarcar el objetivo de este trabajo. El hecho de obtener resultados válidos teniendo en cuenta todas las aristas del problema, y que además, estos posiblemente no estén dentro de la visión experto o no sean intuitivos en primera instancia. Recordando también, que el experto en este caso posee buena experiencia previa y vasto conocimiento teórico en el tema.

Existen varios planes a futuro, algunos de ellos fueron comentados a lo largo del informe. Como por ejemplo, la modificación o eliminación de facts durante el proceso de inferencia y la aceptación o rechazo de soluciones parciales provistas por los agentes bajo diferentes estrategias de control. Además, durante este trabajo estuvo presente la idea de reconocer posibles facts deducidos que sean contradictorios. Para esto se planteaba la imple-

mentación de un tipo nuevo de agente que detectara facts contradictorios bajo un determinado criterio y luego al finalizar el sistema los notifique al usuario. Con respecto al proyecto MAEOS, otro objetivo es implementar agentes que razonen a partir del enfoque basado en casos. Utilizando conocimiento obtenido gracias a la experiencia previa en el área.

Actualmente, se siguen mejorando y aumentando las diversas ontologías y reglas. Este proyecto aún no está finalizado, sino que le queda camino por recorrer en su perfeccionamiento hasta la versión final que será entregada a la empresa privada asociada al proyecto.

Apéndices

A. Diseño del Sistema Experto

Se optó por no poner el código fuente en este texto, principalmente porque no daría claridad al trabajo. Pero sí se adjunta un disco que contiene todos los códigos fuentes y las librerías necesarias para correr la aplicación. Igualmente, para tener una visión global del sistema se muestra a continuación el diagrama de clases en UML. Las interfaces están resumidas a las más importantes y relevantes para tener una noción del funcionamiento de las clases.

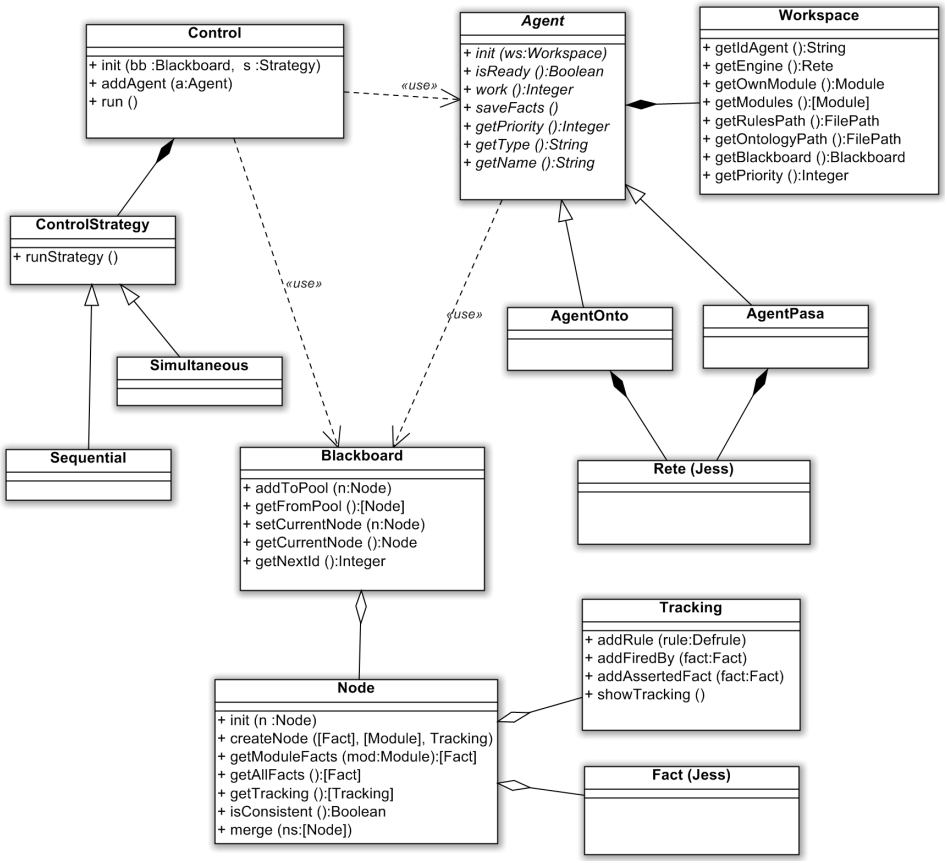

Figura 3: Diagrama de clases de MAMAS

La clase Agent es abstracta, de manera que la implementación de un agente es necesaria realizarla por completo. El AgentOnto y AgentPasa no

extienden la interfaz, por ende, pueden ser tratados indistintamente por el Control, sin que él conozca a priori cual es la tarea que realizan. Los agentes son instanciados a partir de un Workspace, en donde se definen la dirección de los archivos del proyecto Protégé-Frames y las reglas en Jess, el módulo que les pertenece y los módulos en donde también trabajarán (el caso del AgentPasa), donde es que se encuentra el Blackboard y cuál será su identificador en el sistema. Los AgenteOnto y AgentPasa esta compuestos ambos dos por un motor de inferencia Rete provisto por la librería de Jess.

El Blackboard principalmente brinda una interfaz para manipular Nodes, ya sea para agregar y obtener Nodes del Pool, obtener el Node actual y finalmente para definir el Node actual. Al definir el Node actual, el Pool se vacía automáticamente. Por último, posee un método para obtener un identificador de fact único en todo el sistema.

Los Nodes simplifican y abstraen la tarea del Blackboard, ya que si existe algún cambio muy posiblemente se haga en Node y no en la interfaz del Blackboard. En un Node se almacenan los facts nuevos, la traza de las reglas ejecutadas para la creación de los nuevos facts y los módulos que fueron modificados. Un Node está fuertemente ligado a un Node pasado, por lo tanto, cuando se instancia la clase es necesario asignarle Node (generalmente, el que se obtiene con getCurrentNode en el Blackboard). Esto se debe, a que internamente se encarga de diferenciar cuáles fueron los facts que se mantuvieron a lo largo de la inferencia y cuáles fueron los nuevos agregados. Provee un método para diferenciar los facts respecto del módulo al que pertenecen, desligando la tarea de separación al Agent. Cuando hay que unir varios Nodes se utiliza el método mergeNodes, que toma una lista de Nodes, mantiene los facts anteriores y une todos los nuevos. Desde el momento de la instanciación de la clase Node hasta la creación propiamente dicha, el Node pasa por un estado intermedio inconsistente. Por lo tanto, para saber si un Node es consistente con su pasado existe el método isConsistent.

Para obtener el trazado de las reglas a lo largo de la ejecución de un motor Jess se deben capturar los eventos que él genera. Para esto se utiliza un eventHandler que captura determinados eventos y actúa directamente sobre un objeto Tracking cada vez que una regla se activa, guardando en él la regla activada, los facts que generaron la activación y los facts resultantes luego de la ejecución de la regla.

El control posee una interfaz muy simple. Al momento de instanciarlo

se le definen los Agent que van a intervenir en el sistema, se le asigna una estrategia y el Blackboard. Su funcionamiento depende fuertemente de la estrategia de control elegida. Como existe más de una estrategia y posiblemente se puedan agregar nuevas, se optó por separar el algoritmo de la clase Control.

B. KES AMSTA 2011

Basandose en este trabajo se escribió un artículo con el título: "A Multi-Agents System for Analysis and Diagnosis of SMEs", con la autoría de Cecilia Zanni-Merk, Santiago Almirón y Dominique Renaud. Este fue enviado a KES AMSTA 2011 (5th International KES Conference on Agents and Multi-agent Systems, Technologies and Applications - `http://amsta-11.kesinternational.org/`) en diciembre de 2010 y finalmente en marzo de 2011 fue aceptado para su publicación y presentación. Se incluye el texto del articulo a este informe.

C. Ejecución del sistema

Con este informe, se adjunta un disco con el código fuente, las librerías utilizadas, las ontologías con las instancias para el ejemplo del aparatado 6 y el JAR ya empaquetado para su uso. Recordar que la licencia de Protégé-Frames acepta su distribución y uso, en cambio, la de Jess es mucho más restrictiva. En este empaquetado, se incluye la versión académica con validez de uso hasta marzo de 2012. La versión de JAVA utilizada fue la SE 1.6. La estructura de directorios del disco se compone de la siguiente manera:

```
/src (todos los archivos fuentes propios del proyecto)
/lib (librerías de Protégé-Frames y Jess)
/jar (el empaquetado jar listo para usar)
/doc (documentación de uso)
/example (ejemplo ES del aparatado 6)
```

Los archivos de las reglas Jess, deben tener extensión .rules y el nombre debe coincidir con el nombre del proyecto Protégé-Frames. Además, el mismo archivo .rules se utiliza para definir al agente que utiliza las reglas. Por esto, dentro de ese archivo, antes de la definición de las reglas debe estar detallado lo siguiente para los AgenteOnto,

```
;type AgentOnto
;ownmodule Mintzberg_1982
;priority 1
```

Y lo siguiente para los AgentePasa,

```
;type AgentPasa
;ownmodule PAS_REYE_BOIS
;moduleA Reyes_2004
;moduleB Boissin_2003
;priority 3
```

type define el tipo de agente, las dos posibilidades son AgentOnto y AgentPasa

ownmodule define el módulo en donde depositará todo el conocimiento

moduleA define uno de los módulos donde el AgentePasa realiza la equivalencia

moduleB define el otro módulo donde el AgentePasa realiza la equivalencia

priority define la prioridad del agente, debe ser un valor entero positivo.

Para ejecutar desde la línea de comandos, es necesario pasar ciertos parámetros a la maquina virtual de JAVA. Por otro lado, MAMAS solo toma un parámetro de entrada y es el directorio donde se encuentran los proyectos de Protégé-Frames y los archivos con las reglas de Jess. Suponiendo que los archivos de entrada está en la carpeta c:\ejemplo y estamos en la misma carpeta donde se encuentra el mamas.jar, se debe ejecutar la siguiente línea:

```
java -Xss512k -Xmx256m -jar mamas.jar c:\ejemplo
```

Los resultados serán guardados en los respectivos proyectos de Protégé-Frames.

Referencias

[1] Renaud, D., Bouché, P., Gartiser, N., Zanni-Merk, C., Michaud, H.P.: Knowledge Transfer for Supporting the Organizational Evolution of SMEs A Case Study. In: International Conference on Innovation through Knowledge Transfer 2009 (InnovationKT). Hampton Court Palace, London (2009)

[2] Natalya F. Noy and Deborah L. McGuinness. "Ontology Development 101: A Guide to Creating Your First Ontology". Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880 (2001).

[3] Ernest Friedman-Hill. "Jess in Action: Rule-Based Systems in Java". Manning Publications Co. (2003).

[4] George Rudolph. "Some Guidelines For Deciding Whether To Use A Rules Engine". `http://www.jessrules.com/jess/guidelines.shtml` (2008).

[5] Maximiliano Cristiá. "Catálogo Incompleto de Estilos Arquitectónicos". Cátedra Ing. Software Universidad Nacional de Rosario (2006).

[6] Eliza Sachs. Getting Started with Protege-Frames. User documentation Protégé Stanford University. `http://protege.stanford.edu/doc/tutorial/get_started/table_of_content.html` (2006).

[7] Mintzberg, H.: The Structuring of Organizations. Prentice-hall, ISBN 0-13-8555270-3 (1979).

[8] Reyes G. La moyenne enterprise est-elle spécifique? In Proceedings of 7ème Congrès International Francophone en Entrepreneuriat et PME. Montpellier, France. (2004).

[9] Boissin JP et al. Profils de dirigeant et croissance des jeunes entreprises Innovantes. In Proceedings of Congrès de l'AIMS. (2008).

[10] Klein M. (2001),Combining and relating ontologies: An analysis of problems and solutions, [in:] IJCAI'01, Workshop on Ontologies and Information Sharing Eds. G.A. Perez, M. Gruninger, H. Stuckenschmidt, M. Uschold, pp: 53-62. Seattle, WA.

[11] Choi N., Song I.-Y., Han H., 2006, A survey on ontology mapping, SIGMOD Record, Vol. 35, Issue 3, pages: 34-41.

[12] De Bruijn J., Ehrig M., Feier C., Martíns-Recuerda F., Scharffe F., Weiten M. (2006), Ontology mediation, merging, and aligning, [in:] Semantic Web Technologies, Eds. John Davies, Rudi Studer, Paul Warren. John Wiley and Sons, Ltd.

[13] Flouris G., Manakanat D., Kondylakis H., Plexousakis D., Antoniou G. (2007), Ontology change: classification and survey, [in:] The Knowledge Engineering Review. (2), 117-152. Cambridge University Press.

[14] Noy N., Musen M.A. (1999), SMART : Automated support for ontology merging and alignment, [in:] KAW'1999, Proceedings of the Workshop on Knowledge Acquisition, Modeling and Management. Eds. Gaines B. R., Kremer B., Musen M. A. Banff, Alberta, Canada.

[15] Maedche A., Motik B., Silva N., Volz R. (2002), MAFRA: A mapping framework for distributed ontologies, [in:] EKAW'2002, Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Lecture Notes in Computer Science, vol. 2473. Springer Verlag.

[16] Visser P.R.S., Jones D.M., Bench-Capon T.J.M., Shave M.J.R. (1997), An analysis of ontology mismatches: Heterogeneity versus interoperability, [in:] AAAI 1997 Spring Symposium on Ontological Engineering, Stanford University. Stanford CA.

[17] Hameed A., Preece A., Sleeman D. (2004), Ontology reconciliation, [in:] Handbook on Ontologies, Eds. S. Steffen, R. Studer, Springer, Berlin.

[18] Colomb R.M., Ahmad M.N. (2007), Merging ontologies requires interlocking institutional worlds, Applied Ontology, Vol. 2, No. 1.