Razonamiento abductivo aplicado al diagnóstico de PyMEs

Lucas Boullosa

Tesina de

Licenciatura en Ciencias de la Computación

3 de diciembre de 2015

Universidad Nacional de Rosario
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Departamento de Ciencias de la Computación

Directora
Dra. Cecilia Zanni-Merk
ICUBE/BFO Team (UMR CNRS 7357)

Co Directora
Dra. Ana Casali
UNR - CIFASIS

Colaboradora
Nathalie Gartiser
INSA de Strasbourg/LGeCo

Resumen

Las pequeñas y medianas empresas tienen un rol preponderante en las economías de todos los países. Este sector se caracteriza por proveer productos y servicios singulares, a diferencia de las grandes empresas que suelen enfocarse hacia soluciones más estandarizadas.

Estas empresas padecen problemáticas específicas que pueden afectar desde su funcionamiento hasta su permanencia en el mercado. Para tener éxito en este ámbito, son indispensables habilidades gerenciales adicionales. Sin embargo, a menudo no cuentan con personal calificado en esas áreas por lo que deben contratar servicios de consultoría.

El proyecto MAEOS viene desarrollando herramientas que permiten modelar esta realidad para asistir al consultor en su práctica cotidiana. Por un lado, mediante la formalización del conocimiento teórico disponible en la materia. Por otro, construyendo *software* que permite realizar inferencias con dicho conocimiento para asistir en el proceso de análisis. Estas inferencias corresponden a razonamiento de tipo deductivo.

El presente trabajo completa el proceso de asesoramiento brindado a pequeñas y medianas empresas en el marco del proyecto MAEOS mediante la incorporación de un módulo de diagnóstico. En el proceso de diagnóstico utilizamos otro tipo de razonamiento denominado abductivo.

En el desarrollo de esta nueva funcionalidad aplicamos razonamiento abductivo al conocimiento formalizado existente, para obtener una representación de la situación de la empresa, a partir de la cual el experto puede generar recomendaciones acerca de las acciones a realizar para que la misma alcance los objetivos deseados.

Para poder llevar a cabo lo anterior tuvimos que implementar nuevamente parte de las aplicaciones que conformaban el proyecto. Con la utilización del mismo lenguaje en los diferentes módulos logramos una mejor integración entre los mismos.

Palabras claves: razonamiento abductivo, razonamiento deductivo, encadenamiento hacia atrás (*backward chaining*), encadenamiento hacia adelante (*forward chaining*), análisis, diagnóstico, PyMEs (Pequeñas y Medianas Empresas).

Agradecimientos

A Cecilia por el impulso que me dio, la paciencia que me tuvo y su predisposición al trabajo. A Ana por revisar minuciosamente el informe.

A los profesores de la carrera por todo lo que me han enseñado, especialmente lo “no computable”.

A Gabriela por darme el tiempo para hacer este trabajo y acompañarme en el proceso. A Valentina, que quizás cuando sea más grande entienda porque “tenía que estudiar” en lugar de jugar con ella.

A mis padres por todo y por todo lo demás también.

Índice general

Capítulo 1

Introducción

The popular view that scientists proceed inexorably from well-established fact to well-established fact, never being influenced by any improved conjecture, is quite mistaken. Provided it is made clear which are proved facts and which are conjectures, no harm can result. Conjectures are of great importance since they suggest useful lines of research.

- Alan Turing

1.1. Problemática

Uno de los principales problemas que enfrentan las PyMEs (Pequeñas y Medianas Empresas) es que están inmersas en un ambiente muy dinámico, influenciado por factores externos (como las condiciones del mercado, financiamiento, cuestiones legales, etc) e internos (como características del producto, de los empleados, etc). Poder evolucionar en dicho ambiente requiere la capacidad de realizar un análisis global de todos los aspectos de la compañía junto con la habilidad de interpretar los resultados obtenidos; ya que la toma de decisiones adecuadas es lo que en última instancia determinará el éxito de la compañía [40].

Las PyMEs generalmente no cuentan con personal especializado para realizar estas tareas, por lo que recurren a la contratación de servicios de consultoría en el área de administración de empresas para que realicen una evaluación de la misma tendiente a identificar fallas en el funcionamiento de la organización. Como resultado del diagnóstico el consultor puede presentar recomendaciones basadas tanto en su conocimiento teórico como en su experiencia.

Por otro lado, debido a que existe abundante bibliografía disponible en las Ciencias de Gestión y Empresariales [29][28], el uso de herramientas que asistan al experto se vuelve primordial.

1.2. Trabajos relacionados

Numerosos trabajos científicos publicados en el área de la Ingeniería Industrial ([41], [5], [30], [34], entre otros) enfatizan la necesidad de las PyMEs de

consejos expertos para asegurar su evolución y su crecimiento.

Se han realizado diversas investigaciones en el área de la Inteligencia Artificial que abordan diferentes problemáticas de las PyMEs. Por ejemplo en [14] describen dos sistemas (PDG y eRisc) que pueden verse como sistemas expertos por la funcionalidad que proveen. PDG es un sistema que provee diagnósticos y recomendaciones para PyMEs implementado en forma de cuestionario en el que se le asigna un puntaje las respuestas. Dicho cuestionario fue desarrollado por un equipo de expertos de diferentes disciplinas y posteriormente refinado y validado por los dueños de empresas. El grupo de expertos definió una serie de indicadores a los que se les asigna un peso que permite comparar el desempeño de la empresa con el de otras similares de un grupo de control. En base al valor obtenido se generan recomendaciones. eRisc es una aplicación web para el manejo de riesgo en las inversiones. Permite identificar, medir y manejar los diferentes factores de riesgo que pueden comprometer el éxito de una PyME. Consiste en un cuestionario dinámico que se utiliza para recolectar la información del proyecto de la PyME que busca financiamiento. El cuestionario se desarrolló a partir de los factores de riesgo que afectan el éxito de las PyMEs identificados luego de un extenso estudio de la bibliografía del tema. Las respuestas se graduan para producir un índice de riesgo. Tras analizar las falencias de ambos sistemas, los autores reconocen que hubiera sido beneficioso haber incorporado técnicas de IA en su implementación.

Otros trabajos se enfocan sólo en una parte del desarrollo, como en [12] que utilizan una técnica llamada *Ripple Down Rules* para mejorar la fase de adquisición del conocimiento; o sólo en problemas específicos como el de mantenimiento de maquinarias [13] mediante un sistema basado en reglas; o en [4] donde desarrollan una ontología para organizar la información tecnológica (herramientas utilizadas/desarrolladas) de la empresa, lo que le permite detectar relaciones entre conceptos y disparar eventos para responder a determinados cambios. En general, ninguna de estas publicaciones logran una solución integral que es una de las novedades del presente trabajo.

Por otro lado, entre los primeros en subrayar la utilidad del razonamiento abductivo en la Inteligencia Artificial están los trabajos de [31] y [39] en la década del '70. Tal como señalan los autores de [37] entre otros, hay un amplio consenso en que los humanos utilizan abducción en el proceso de diagnóstico. Por todo esto, el diagnóstico es uno de los dominios de aplicación más representativos y mejor comprendidos de razonamiento abductivo. Se han construido e implementado satisfactoriamente varios sistemas expertos basados en diagnóstico abductivo, principalmente en el dominio médico [24][37]. En estos sistemas la base de conocimientos contiene relaciones causales entre trastornos y síntomas. Las inferencias consisten en generar hipótesis para explicar las enfermedades.

Para finales de los '80 muchos investigadores reconocieron la aplicabilidad de la abducción a varios campos relacionados con la Inteligencia Artificial entre los que se destacan interpretación de lenguaje natural [35], aprendizaje automatizado [23], visión artificial y reconocimiento de imágenes [38], generación de planes [3] y razonamiento basado en casos [27].

En [15] citan numerosas aplicaciones de la abducción y reconocen que si bien su utilidad ha sido ampliamente demostrada, existen pocas aplicaciones que excedan en nivel de meros ejemplos académicos; es decir que no se ha desarrollado todo su potencial a una escala industrial. Por esto concluyen que el desafío es desarrollar aplicaciones que tengan un impacto real en la industria. Sin embargo,

al estudiar la bibliografía no se han encontrado trabajos similares al desarrollo presentado aquí.

1.3. El proyecto MAEOS

MAEOS[1] es un proyecto de los laboratorios de investigación del INSA[2] de Estrasburgo, Francia; cuyo objetivo es desarrollar un conjunto de herramientas de software para el análisis y diagnóstico de PyMEs y con esto mejorar la eficiencia y el rendimiento del asesoramiento brindado a las empresas.

Entre las características a destacar de estas herramientas se encuentran:

- la capacidad de evolucionar a la par del ambiente de las PyMEs (particularmente en lo administrativo y/o legal).
- la capacidad de reflejar las riquezas y contradicciones inherentes a los modelos de las Ciencias de Gestión y Empresariales.
- deben permitir que el consultor acceda en forma eficiente al conocimiento proveniente de diversas fuentes.

Para lograr este objetivo, se creó un equipo de trabajo multidisciplinario, cuyas principales áreas de investigación son: Inteligencia Artificial, Ingeniería de Software y Ciencias de la Gestión. Entre los resultados obtenidos por este grupo se destacan dos sistemas y una librería de software.

El primero, llamado DONNA[3] es una aplicación web que permite manipular las bases de conocimiento. Fue diseñada para que el experto en Ciencias de Gestión pueda formalizar el conocimiento extraído de artículos y libros. DONNA posibilita la creación, modificación, borrado de ontologías, conceptos, relaciones entre conceptos y edición de reglas que utilizan esos conceptos [10].

El otro sistema llamado DISKO[4] provee una interfaz amigable que facilita la adquisición de datos y la visualización de resultados. Está desarrollado como un *plugin* del IDE Eclipse con el propósito de simplificar la instanciación de los conceptos formalizados en DONNA. Otra de sus funciones es la de presentar los resultados en forma de grafo donde los nodos son los conceptos y los arcos representan equivalencias semánticas entre conceptos de distintas ontologías [9].

Por último, la librería llamada MAMAS[5] desarrollada en [2] se incorpora a DISKO para permitir inferir nuevo conocimiento a partir de un caso ingresado. A nivel conceptual, pretende modelar la noción de múltiples puntos de vista, cada uno representado por un agente que posee su propia base de conocimiento. Estos agentes operan en forma similar a un panel de expertos en el que cada uno razona por su cuenta para luego volcar sus conclusiones en un área común de manera que puedan ser aprovechadas por los otros agentes. A nivel de implementación, utiliza un motor de inferencia para generar nuevo conocimiento a partir de los datos ingresados.

El software implementado para el proyecto MAEOS (DONNA, MAMAS y DISKO), apunta al desarrollo de un modelo conceptual de los conocimientos

[1] Modélisation de láccompagnement de lévolution organisationnelle et strategique des SMEs
[2] Institut National des Siences Appliquées
[3] Designer for ONtologies with Navigators and Assistant
[4] Development Interface for Sme Knowledge Organization
[5] MAEOS Argumentation with a Multi-Agents System

asociados a la gestión de la evolución estratégica de las PyMEs. Esto requiere, en consecuencia, una capitalización y una estructuración de los conocimientos existentes en dichos dominios.

Las características innovadoras del proyecto MAEOS están asociadas a la naturaleza de los conocimientos a manipular y a los objetivos finales a alcanzar. No se intenta construir un servidor de conocimientos para la gestión de una memoria de la empresa (como podría ser el caso de herramientas existentes tipo ERP[6]), sino desarrollar un marco de estructuración de conocimientos teóricos y prácticos sobre el análisis estratégico de la evolución de las PyMEs. Esta fuerte interacción entre el campo de las Ciencias de la Computación y el de las Ciencias de Gestión hace que el proyecto MAEOS sea un desafío científico.

1.4. Características de las empresas

En las siguientes secciones formalizaremos los distintos aspectos que nos interesan al momento de realizar la evaluación de una PyME en el marco del proyecto MAEOS.

1.4.1. Estados

Durante el proceso de análisis y diagnóstico, la empresa transita por diferentes estados para los cuales consideramos necesario brindar definiciones precisas. Dichos estados son:

Estado Inicial:
es el estado de partida. Se obtiene mediante un relevamiento realizado en la empresa y permite identificar sus características distintivas (actividad principal, composición, estado jurídico, etc).

Estado Actual:
describe la situación actual de la empresa. A partir el estado inicial se ejecuta el proceso de inferencia hacia adelante; los resultados obtenidos le permiten al consultor determinar si debe recabar más datos en la empresa. Alcanzar este estado puede involucrar varias iteraciones.

Estado Final:
es el estado que el dirigente de la empresa aspira alcanzar. Puede estar compuesto tanto por modificaciones de valores del estado actual, como también por agregado de otras características deseadas.

Estado Ideal:
es un posible estado que posee las condiciones necesarias para evolucionar hacia el estado final. Luego de aplicar el proceso de inferencia hacia atrás al estado final se obtienen posiblemente un conjunto de estados ideales. El experto selecciona de este conjunto el que según su criterio reune las condiciones más favorables; ese es el que designamos aquí como estado ideal.

En la Figura 1.1 se ilustran las transiciones entre los distintos estados.

[6]Enterprise Resource Planning

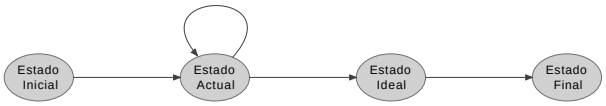

Figura 1.1: Transición de estados

1.4.2. Descripción del proceso

Al realizar la evaluación el consultor comienza completando un formulario definido en [2] que consta de la información básica de la empresa (nombre, rubro, etc); de algunos indicadores de su estructura (cantidad de empleados, estado legal, etc) y posición en el mercado (ingresos anuales, capital, etc).

La información recabada se ingresa al sistema mediante DISKO y describe el *estado inicial* a partir del cual puede utilizar MAMAS para deducir nuevos datos que describen el *estado actual*. En este punto el consultor puede notar que omitió observar algo en la empresa o que algo que debería observar no está presente. El primer caso puede ser un error de su parte, mientras que el segundo puede constituir un problema de la empresa. Resolver estas cuestiones es lo que hace que determinar el *estado actual* pueda requerir varias visitas a la empresa.

Provisto de esta información el consultor puede presentar un informe al dueño de la empresa con los puntos que habría que mejorar. A su vez el dueño también puede requerir la incorporación de otros ítems que no estén presentes en la actualidad pero que formen parte de los objetivos a futuro.

Todo esto permite definir el *estado final*, a partir del cual se puede utilizar el módulo de inferencia hacia atrás para generar **automáticamente** un posible *estado ideal* que eventualmente evolucione hacia el *estado final* mencionado.

El proceso descripto está esquematizado en la Figura 1.2. La flecha punteada entre el *estado actual* y el *estado ideal* indica una transición deseada, para que ocurra realmente, el consultor puede comparar ambos estados y sugerir un plan de acción.

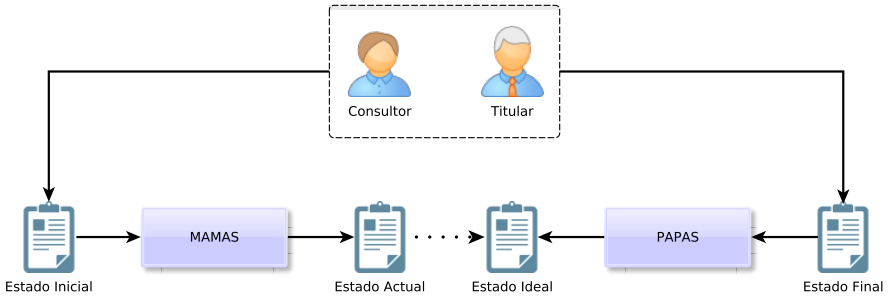

Figura 1.2: Ciclo de interacción con la empresa

1.5. Contribuciones generales

En este trabajo realizamos aportes al proyecto MAEOS en varias dimensiones. Por un lado, continuamos aumentando el conjunto de funcionalidades ofrecidas: incorporamos la capacidad de realizar recomendaciones para que el consultor pueda sugerir posibles condiciones necesarias para alcanzar un estado deseado de la empresa. Llamamos a este componente PAPAS (Program for Abductive Proposal of Actions for SMEs), constituye el principal aporte de este trabajo y si bien lo aplicamos al diagnóstico de PyMEs, en realidad es lo suficientemente general para ser aplicado en otros dominios.

Por otro lado, para llevar esto a cabo resultó evidente que sería beneficioso una transformación en la representación del conocimiento que se encontraba formalizado en el proyecto en forma de ontologías.

Como se menciona en la Sección 1.3, el proyecto MAEOS ha ido incorporando funcionalidades a lo largo de los años. En este trabajo, teniendo un panorama más concreto de su alcance, planteamos una reestructuración de la arquitectura para adecuarla a las tendencias y tecnologías actuales en el desarrollo de sistemas.

Otra contribución no menos importante, es que reemplazamos el motor de inferencias que se venía utilizado en el proyecto que posee una licencia privativa, por uno con licencia libre que además utiliza un algoritmo de inferencia más eficiente.

Por último, en una etapa preliminar de esta investigación, publicamos [10] donde definimos una arquitectura de capas para sistemas basados en conocimientos en el ámbito de las PyMEs.

1.6. Organización del trabajo

Este informe está organizado de la siguiente manera. Primeramente en el Capítulo 2 se explican los fundamentos teóricos utilizados en el resto del trabajo. A continuación en el Capítulo 3 se realiza una exposición detallada de los aportes mencionados en la Sección 1.5. En el Capítulo 4 se aplican las herramientas desarrolladas a un caso de una empresa real. Por último, el Capítulo 5 está dedicado a las conclusiones y posibles direcciones de investigación de trabajos futuros. En el Apéndice A se describen los criterios utilizados para organizar el código desarrollado.

Capítulo 2

Conceptos generales

En este capítulo se exponen los conceptos teóricos utilizados en el contexto del presente trabajo. En cada sección se mencionan tanto los formalismos subyacentes en la implementación existente al momento de comenzar este trabajo como los implicados en la nueva implementación. No se intentan desarrollar los detalles de cada teoría, sino más bien enunciar las definiciones y los principales resultados de cada una. Por último, se exponen los argumentos que motivaron los cambios.

2.1. Sistema expertos

Los sistemas expertos son sistemas de software que hacen vasto uso de conocimiento especializado para resolver problemas al nivel de un especialista humano. El especialista es una persona que posee conocimientos o habilidades especiales en cierta área que otros no disponen. Esto le permite resolver problemas que la mayoría no podría o hacerlo con mayor eficiencia [19].

En la Figura 2.1 se muestra la arquitectura típica. El usuario ingresa los hechos o información al sistema experto y recibe asesoramiento experto como respuesta.

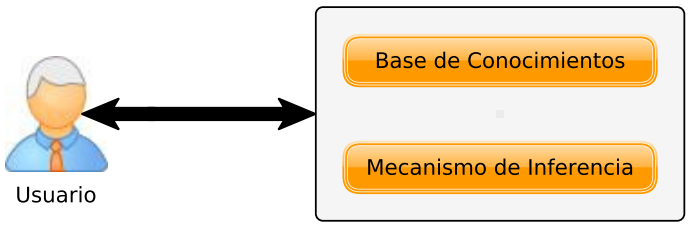

Figura 2.1: Arquitectura de un Sistema Experto

Internamente el sistema experto incluye dos componentes principales. La **base de conocimiento** contiene el conocimiento que le permite al **mecanismo de inferencia** obtener conclusiones; éstas son las respuestas del sistema a las consultas del usuario.

La **base de conocimiento** es un elemento clave en este tipo de sistemas. Permite almacenar tanto el conocimiento factual como el heurístico. El factual es el que puede encontrarse en libros y es compartido por los expertos en la materia; mientras que el heurístico es menos riguroso y está basado en la experiencia individual del experto [6]. En otras palabras, una base de conocimientos está constituida por un conjunto de representaciones de hechos acerca del mundo.

Existen diversos formalismos para representar el conocimiento, algunos de los cuales se analizan en la Sección 2.2. En el diseño de un sistema experto, la elección de una representación adecuada es crucial ya que afecta el desarrollo y eficiencia del mismo.

El **mecanismo de inferencia** organiza y controla los pasos que se siguen hacia la resolución del problema. Estos métodos de resolución son utilizados por programas llamados **motores de inferencia**, que manipulan la base de conocimiento para crear una línea de razonamiento [6]. Este componente puede funcionar de diferentes formas que se explican en la Sección 2.2.2; mientras que algunas herramientas que lo implementan se muestran en la Sección 2.4. Cada herramienta está basada en un determinado formalismo de representación del conocimiento.

2.2. Representación del conocimiento

La representación del conocimiento es un área de la Inteligencia Artificial que estudia como representar simbólicamente la información de un dominio de forma que pueda ser manipulada por sistemas computacionales para resolver tareas complejas [11].

Existen diferentes paradigmas para estructurar y representar el conocimiento, que pueden caracterizarse de acuerdo a la manera de expresar el conocimiento estático y al procedimiento de inferencia que utilizan.

Entre los más importantes se destacan los formalismos lógicos, los sistemas de producción y los estructurales. Dentro de los estructurales los que aplican al presente trabajo son los marcos (*frames*), los objetos y las ontologías.

A continuación se describen brevemente cada uno siguiendo la exposición de [6].

2.2.1. Formalismos lógicos

En estos esquemas, el conocimiento se representa mediante un conjunto de fórmulas lógicas bien formadas, mientras que el procedimiento empleado para realizar inferencias es el método deductivo del sistema lógico.

La más utilizada es la lógica de primer orden en la que las fórmulas bien formadas consisten en una combinación de predicados, variables, constantes, conectores, cuantificadores y funciones. El método para efectuar deducciones es la **resolución** [6].

2.2.2. Sistemas de producción

En los sistemas de producción el conocimiento se representa mediante una base de conocimientos. La base de conocimiento está formada por una **base de**

hechos y una **base de reglas**.

La **base de hechos** describe el estado del dominio o del contexto del problema y se va modificando a medida que se aplican los procesos de inferencia.

La **base de reglas** contiene los elementos de deducción básicos que utiliza el sistema. Las reglas suelen seguir una estructura uniforme del tipo:

```
SI condiciones ENTONCES acciones
```

En la cual las `acciones` situadas en el consecuente se ejecutan si se cumplen las `condiciones` situadas en el antecedente de la regla. Las `condiciones` (también llamadas LHS[1]) pueden hacer referencia a hechos de la base de hechos relacionados con conectores lógicos (`and/or`). Las `acciones` (o RHS[2]) generalmente son expresiones que modifican la base de hechos (tanto para agregar un hecho nuevo, como para modificar/eliminar uno existente), aunque también pueden ser instrucciones de control (para terminar la ejecución) o que emiten alguna salida (para mostrar algún mensaje al usuario).

El otro componente de los sistemas de producción es **motor de inferencias** que actúa sobre los dos componentes anteriores según una estrategia de deducción determinada.

El funcionamiento del motor de inferencia comienza con una fase de **reconocimiento de patrones** en la que compara los hechos presentes en la base de hechos con las condiciones presentes en las reglas. En un sistema con gran cantidad de reglas y hechos en la base de hechos, puede ocurrir que haya varias reglas que puedan ejecutarse en un determinado momento. Estas reglas se dice que están en conflicto y se almacenan en la **agenda**.

Por eso, la siguiente fase es la de **resolución de conflictos** en la que se selecciona una regla de la agenda para su ejecución de acuerdo a una política propia de cada motor. Ejemplos de estas políticas pueden ser: según el orden de las reglas, seleccionar la regla más general, o la más específica, entre otras.

Por último, la fase de **ejecución** de la regla puede dar lugar a cambios en la base de hechos. Estos cambios pueden provocar la activación de otras reglas, lo que se conoce como encadenamiento. Cuando el encadenamiento se da desde el problema a la solución se denomina hacia adelante; si en cambio ocurre desde las hipótesis hacia los hechos se denomina hacia atrás [19]. Estos esquemas se explican en las secciones siguientes.

Una de las características más importantes de los sistemas de producción es que las reglas se diseñan de manera que sean independientes entre sí. Con esto se consigue que la base de conocimientos sea modular y se puedan agregar o quitar reglas en cualquier momento, favoreciendo el desarrollo incremental del sistema.

A diferencia de la programación imperativa en la que se listan las instrucciones a ejecutar; en los sistemas de producción el conocimiento se expresa en forma declarativa, es decir que no se asume un orden de ejecución de las reglas sino que es el motor de inferencias el que determina cual ejecutar en cada ciclo.

Encadenamiento hacia adelante

El encadenamiento hacia adelante (*forward chaining*) es una estrategia dirigida por los datos. A partir de hechos insertados en la memoria de trabajo, se

[1] Left Hand Side
[2] Right Hand Side

pueden hacer verdaderas las condiciones de varias reglas que son incorporadas a la agenda para su posterior ejecución, la cual puede derivar hechos adicionales y así sucesivamente hasta que no pueden obtenerse más conclusiones.

Esta estrategia esta esquematizada en la Figura 2.2. Allí se ve que el primer paso es determinar las posibles reglas a disparar en base a los hechos presentes en la memoria de trabajo. Estas reglas se denominan **conjunto de conflicto**. Para seleccionar una regla de este conjunto se utiliza la **estrategia de resolución de conflictos**. Con la regla seleccionada se procede a su ejecución, lo que puede dar lugar a nuevos hechos en la memoria de trabajo y a un nuevo comienzo del ciclo.

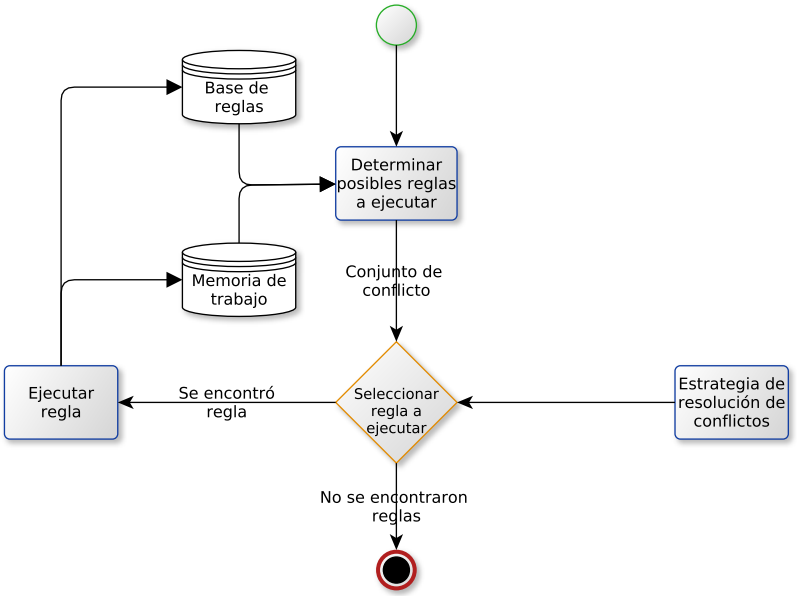

Figura 2.2: Algoritmo de encadenamiento hacia adelante

Encadenamiento hacia atrás

El encadenamiento hacia atrás (*backward chaining*) es una estrategia dirigida por objetivos. Esto quiere decir que el motor de inferencias intenta verificar un hecho (alcanzar un objetivo) buscando aquellas reglas que tengan ese hecho en su conclusión para luego intentar verificar sus premisas. Dichas premisas se convierten a su vez en nuevos hechos a verificar.

El algoritmo se esquematiza en la Figura 2.3. Comienza examinando la memoria de trabajo para determinar si los objetivos están presentes. De no ser así, el siguiente paso es buscar en la base de reglas aquellas que en sus conclusiones contengan los objetivos (representado en el nodo "Determinar posibles reglas a ejecutar"). En caso que haya varias, se utiliza **estrategia de resolución de conflictos** para seleccionar una. Luego se incorporan las premisas de la de la regla seleccionada como nuevos objetivos, dando comienzo a un nuevo ciclo. El proceso continúa hasta que se logran satisfacer los objetivos o no encuentran

más reglas que apliquen.

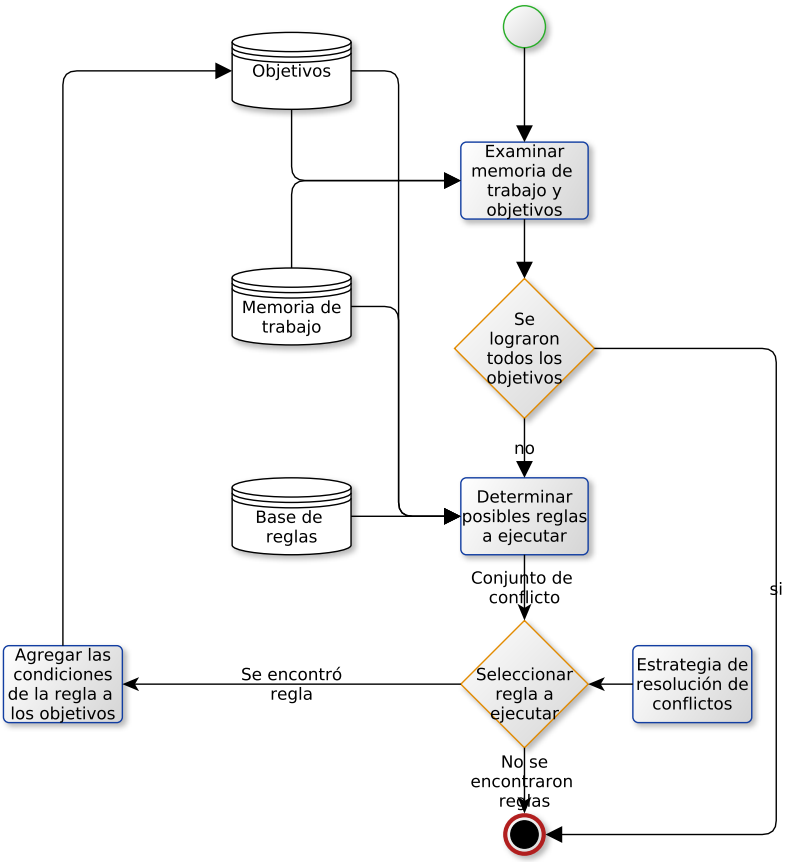

Figura 2.3: Algoritmo de encadenamiento hacia atrás

2.2.3. Marcos

Los marcos (*frames*) proporcionan una estructura para entender la representación del conocimiento asociado a situaciones estereotípicas.

Los elementos que utiliza esta técnica para representar conocimiento son:

marcos:
representan conceptos e instancias de los mismos.

relaciones:
expresan dependencias entre conceptos.

propiedades:
son los atributos de los conceptos.

facetas:
representan restricciones sobre los valores que pueden tomar las propiedades.

Los marcos que representan conceptos o situaciones genéricas se denominan **marcos clase** y se describen por un conjunto de propiedades comunes al concepto que modelan. Los marcos que representan elementos específicos de los marcos clase se denominan **marcos instancia** y sus propiedades se completan con información concreta de los individuos que representan.

Las **relaciones** son propiedades especiales cuyo valor se corresponde con otro marco. Pueden ser estructurales, que son fijadas por el sistema de representación, como ser *subclase-de*, *instancia-de* que permiten formar taxonomías. Otro tipo de relaciones son las *ad hoc*, que expresan dependencias particulares entre conceptos del dominio.

Las **propiedades** (*slots*) pueden ser de clase o de instancia. Las propiedades de clase representan características genéricas de un concepto y tienen el mismo valor en todas las instancias de esa clase y sus subclases. Las propiedades de instancia representan atributos con valores particulares de cada marco instancia.

Por último, las facetas (*facets*) son una forma de especificar restricciones a los valores de un *slot*. Las facetas permiten por ejemplo limitar la cardinalidad (cuantos valores puede contener), declarar el tipo de la propiedad (tipo de dato con el que se rellenará la propiedad: entero, cadena de caracteres, instancia de otra clase), el rango de valores (valores mínimos y máximos), o el valor por defecto (valor que debe tomar si no se asigna uno explícitamente) [33].

Todos los elementos mencionados permiten representar el conocimiento estático. El otro elemento que caracteriza una representación del conocimiento es el procedimiento de inferencia. En el esquema de marcos, el proceso de inferencia se utiliza para resolver consultas en la base de conocimiento, mantener la consistencia semántica de los valores introducidos en las propiedades o para determinar el marco clase a la que pertenece un marco instancia. En este último proceso se intenta equiparar los valores de las propiedades del marco instancia dado con los de los marcos clase de la base de conocimiento. Para esta comparación se utiliza una fórmula implementada en el motor de inferencias que indica el grado de coincidencia entre ambos; dicha fórmula debe tener en cuenta las propiedades heredadas según la estructura jerárquica.

2.2.4. Objetos

En el contexto de la Inteligencia Artificial la representación del conocimiento mediante objetos se refiere a una forma de modelar el mundo; comparte muchas características con los marcos.

La representación del dominio del problema se realiza construyendo una red de objetos que cooperan e interactúan entre sí enviándose mensajes para resolver una tarea.

Tomando la definición de [8], un objeto es una entidad compuesta por:

estado:
: se refiere al conjunto de los valores de sus atributos en un instante de tiempo dado.

comportamiento:
: se refiere a la funcionalidad del objeto, queda determinado por los mensajes a los que sabe responder dicho objeto, es decir, qué operaciones se pueden realizar con él.

identidad:
es la propiedad que permite diferenciar a un objeto y distinguirlo de otros.

Así, se puede decir que el estado de un objeto representa los resultados acumulados de su comportamiento y que aunque se modifique el estado, la identidad del objeto se preserva durante el tiempo de vida del mismo.

El otro concepto fundamental de este esquema es el de **clase**. Una clase define las propiedades y el comportamiento comunes a un conjunto de objetos. Así un objeto es una instancia de una clase.

Hay cuatro elementos a tener en cuenta al trabajar con objetos:

abstracción:
denota las características esenciales de un objeto que lo distinguen de los demás tipos de objetos, proporcionando límites conceptuales bien definidos, desde la perspectiva del observador.

encapsulamiento:
es el proceso de almacenar en un mismo compartimento los elementos que constituyen una abstracción (su estructura y su comportamiento). Permite separar la interfaz de una abstracción de su implementación.

modularidad:
es la propiedad de descomponer la representación en un conjunto de módulos con cohesión (es decir, que agrupan abstracciones relacionadas lógicamente) y débilmente acoplados (es decir, que minimizan las dependencias entre ellos).

jerarquía:
es una clasificación, categorización u ordenamiento de las abstracciones.

La abstracción se enfoca en el comportamiento observable del objeto, mientras que el encapsulamiento lo hace en la implementación que da lugar a dicho comportamiento. El encapsulamiento se consigue habitualmente mediante el **ocultamiento de información**, que es el proceso de ocultar los secretos del objeto que no forman parte de sus características esenciales (i.e., su estructura y la implementación de sus métodos).

Los tipos de jerarquías más importante son la **herencia** y la **agregación**.

La herencia define una relación entre clases que en la que una clase comparte las estructura y comportamiento definido en una o más clases. Típicamente, una subclase aumenta o redefine la estructura/comportamiento de sus superclases. Semánticamente, la herencia denota una relación de "es un".

La agregación describe una relación de "parte de". Permite el agrupamiento físico de estructuras relacionadas lógicamente; se utiliza tradicionalmente en el caso que una clase está compuesta por otra.

El paradigma de orientación a objetos apareció como un desarrollo evolutivo, no revolucionario; o sea no rompe con los avances del pasado sino que se basa en principios existentes (tales como abstracción, encapsulamiento, modularidad, jerarquía, entre otros). Lo importante del modelo de objetos es que ha logrado conjugar todos estos elementos en forma sinérgica.

2.2.5. Ontologías

En el campo de la Inteligencia Artificial, el término ontología ha sido ampliamente utilizado. De la gran cantidad de bibliografía existente sobre ontologías [20][21][22]; una de las definiciones más aceptadas es la de [32] que considera una ontología como una descripción formal de conceptos dentro de un dominio determinado.

Dichos conceptos poseen propiedades para describir sus características; a su vez estas propiedades pueden tener restricciones. Es habitual referirse a los conceptos como clases y a las propiedades como *slots*. Una forma de especificar restricciones a los valores de un *slot* es utilizando facetas (*facets*) tal como se explica en la Sección 2.2.3.

Además las ontologías pueden tener estar compuestas por otros elementos como relaciones, funciones y axiomas. Las relaciones representan un tipo de asociación entre conceptos; se definen formalmente como cualquier subconjunto del producto cartesiano de n conjuntos: $R \subset C_1 \times C_2 \times \ldots \times C_n$; sin embargo las ontologías contienen usualmente relaciones binarias donde el primer argumento se denomina dominio y el segundo rango. Las funciones son un tipo especial de relación en el que el elemento n-ésimo es único para los *n-1* anteriores. Los axiomas se utilizan para modelar sentencias que son siempre verdaderas; se pueden incluir en las ontologías para clarificar el significado que se pretende dar a los términos, definir restricciones complejas para los valores de los atributos o de los argumentos de las relaciones, para deducir nueva información o para verificar la consistencia de la ontología.

La presencia de estos elementos da lugar a una clasificación de las ontologías entre livianas y pesadas. Las livianas son aquellas que no poseen axiomas, generalmente constan de un conjunto de conceptos organizados para formar taxonomías, es decir están compuestas por conceptos, propiedades y relaciones entre ellos. En las pesadas los axiomas permiten modelar el dominio en forma más profunda ya que las restricciones semánticas que proveen otorgan mayor expresividad.

Un último componente de las ontologías son las instancias que se utilizan para representar elementos o individuos. Un conjunto de instancias de conceptos definidos en la ontología conforma la base de conocimientos.

Las ontologías definen un vocabulario de representación que permite a tanto a usuarios de sistemas como a agentes de software compartir y reusar conocimiento de un dominio.

2.3. Tipos de razonamiento

En esta sección se analizan distintos tipos de razonamiento que permiten generar nuevo conocimiento. En particular se analizan la deducción (razonamiento lógico en el que las conclusiones deben desprenderse de sus premisas), la inducción (razonamiento del caso específico al general) y la abducción (razonamiento hacia atrás desde una conclusión verdadera, hacia las premisas que habrían causado la conclusión) [19].

Hay distintas corrientes epistemológicas que cuestionan la validez de los razonamientos inductivos y abductivos, sin embargo estas consideraciones escapan al ámbito de este trabajo, se pueden ver más detalles en [25]. Aquí, en cambio

se adopta un enfoque más pragmático ya que los mismos resultan útiles en el sentido que garantizan cierto éxito en cuanto a la conservación de la verdad.

2.3.1. Razonamiento deductivo

El método de razonamiento deductivo es el único considerado válido desde el punto de vista lógico. Un razonamiento válido se denomina **deducción** y la lógica formal proporciona criterios para reconocer las deducciones.

Un razonamiento lógico consiste un conjunto de enunciados en el se afirma que el último está justificado por los anteriores de la cadena. Hay esquemas de razonamiento que garantizan la obtención de razonamientos válidos llamados **reglas de inferencia**. Una de las más utilizada es la llamada *Modus Ponens*:

$$\begin{array}{l} p \rightarrow q \\ \underline{p \qquad} \\ q \end{array}$$

La línea horizontal permite separar las **premisas** (ubicadas por encima) de la **conclusión** (el enunciado que aparece debajo de la línea). La característica esencial de la lógica deductiva es que una conclusión verdadera debe obtenerse de premisas verdaderas.

Generalizando, sean Γ el conjunto de fórmulas bien formadas de un sistema formal, $\{P_1, P_2, \ldots, P_n\} \subset \Gamma$ las premisas y $C \in \Gamma$ la conclusión, un razonamiento puede expresarse como: $P_1, P_2, \ldots, P_n \vdash C$. El símbolo $\vdash$ indica que lo que sigue es un teorema, es decir una conclusión de una cadena de inferencia válida.

Este tipo de razonamiento es el más utilizado en los sistemas expertos basados en reglas, en particular los que utilizan encadenamiento hacia adelante visto en la Sección 2.2.2.

2.3.2. Razonamiento inductivo

Es el tipo de razonamiento que a partir de numerosos casos (premisas) individuales busca proveer evidencia suficiente para respaldar la obtención de una regla.

Se lo puede esquematizar de la siguiente forma: supongamos que tenemos una serie de enunciados observacionales verdaderos, todos los cuales afirman que ciertos objetos (designados por A, B, C, etc) que pertenecen a una clase Q poseen cierta propiedad p.

A tiene la propiedad p
B tiene la propiedad p
C tiene la propiedad p
$\vdots$

Cuando el número de estas premisas es "suficientemente grande" (y no se dispone de ninguna premisa que afirme que cierto objeto M de la clase Q no tiene la propiedad p) el razonamiento inductivo permitiría afirmar la regla:

para todo x si x es de clase Q, entonces x tiene la propiedad p

A diferencia del razonamiento deductivo que garantiza la verdad de la conclusión, en el razonamiento inductivo no existe acuerdo sobre cuándo considerar un argumento como válido; sin embargo dado que permite expandir nuestro conocimiento sobre el mundo, es parte indispensable del método científico.

Vale la pena aclarar que a pesar del nombre, no debe confundirse con el principio de inducción matemático, ya que este último se basa en una regla de inferencia y es por lo tanto un ejemplo de razonamiento deductivo.

2.3.3. Razonamiento abductivo

Este método de razonamiento fue entrevisto por primera vez por Aristóteles, quien lo consideraba un tipo de silogismo en el que las premisas sólo brindan cierto grado de probabilidad de la conclusión. Mucho después fue descripto por el lógico Charles Peirce quien lo consideró una forma más de razonamiento junto a la deducción e inducción [16].

Inicialmente, Peirce no utilizó la palabra abducción para denominar este tipo de razonamiento, en su lugar habló de "hipótesis", "conjetura" o "suposición", ya que se refiere a fenómenos que no es posible observar directamente. Como en el caso de la inducción, esta inferencia no tiene carácter necesario sino meramente probable ya que podría haber otra explicación de la premisa, distinta de la que se establece en la conclusión.

El esquema es el siguiente:

$$\begin{array}{l} p \to q \\ \underline{q \qquad} \\ p \end{array}$$

En lógica este esquema se denomina falacia de afirmación del consecuente, lo que significa que los razonamientos de esta forma son inválidos, porque la verdad de las premisas no garantiza la verdad de la conclusión. Es importante destacar que el hecho que sea una falacia no implica que sus premisas o conclusión sean falsas.

Al igual que en la deducción, en la abducción a partir de un caso particular y de una regla general se obtiene un caso particular. Sin embargo, en la deducción el resultado es una consecuencia lógica de la regla general y por tanto verdadero; mientras que en la abducción el resultado es simplemente una "hipótesis" (una posible explicación al hecho observado) y no una conclusión definitivamente cierta [18].

Aunque la abducción no es un razonamiento deductivo válido, constituye un método útil de inferencia y se ha usado en sistemas expertos [19]. Además está ampliamente aceptado que el proceso de diagnóstico humano pertenece a la categoría de la inferencia abductiva y constituye el ejemplo más típico de la clase de problemas que pueden ser resueltos mediante este tipo de inferencia [18].

Un ejemplo sencillo de un razonamiento abductivo es al observar que el pasto está mojado, es razonable suponer que llovió. Claramente en este caso puede haber otras causas tales como que alguien lo regó.

Otro ejemplo conocido es el caso del descubrimiento del planeta Neptuno. Al observar los datos de la orbita de Urano, se notó que se desviaban de lo predicho por las leyes de Newton. Una posible explicación era que la teoría de

la gravitación universal fuera falsa; pero dado la gran cantidad de evidencia que la apoyaba esto no parecía muy probable. Dos astrónomos John Couch Adams y Urbain Le Verrier sugirieron independientemente y casi en simultaneo la existencia de otro planeta en el sistema solar. Esto proveía una mejor explicación a la desviación de la órbita de Urano.

En este último ejemplo se ve que es una técnica es útil en la reducción del espacio de búsqueda ya que permite generar hipótesis razonables que luego pueden verificarse.

La abducción es el proceso de razonamiento mediante el cual se engendran las nuevas ideas, las hipótesis explicativas y las teorías científicas. En este sentido se dice que es un tipo de razonamiento sintético (en el sentido que permite sintetizar nuevo conocimiento) o ampliativo (porque permite ampliar el conocimiento existente). Por todo esto constituye un instrumento de búsqueda de conocimiento.

2.4. Herramientas

En esta sección se presentan los motores de inferencia involucrados en este trabajo. El primero es Jess que se utilizó en la versión anterior de MAEOS y el segundo es Drools que se utilizó para reemplazarlo en la versión actual del sistema.

Además se muestra un ejemplo de *backward chaining* utilizando la sintaxis de cada uno de manera que se puedan apreciar las diferencias y similitudes de cada implementación. Esto servirá para comprender mejor la naturaleza de los problemas enfrentados durante el desarrollo de PAPAS que se detallan en la Sección 3.3.

2.4.1. Prolog

Si bien Prolog no se utiliza en el proyecto MAEOS, se incluye aquí para presentar el ejemplo que se utilizará en las secciones siguientes.

Prolog fue uno de los primeros lenguajes de programación en utilizar *backward chaining* para realizar inferencias por lo que ha servido como inspiración para las implementaciones posteriores [42]. Está basado en el paradigma de programación lógica, que utiliza fórmulas lógicas para representar el conocimiento y realizar inferencias sobre el mismo.

En el Listado 2.1 se muestra un ejemplo del mismo. Las líneas 1 a 3 muestran los predicados utilizados para describir el problema (i.e., la base de conocimientos). Las líneas 5 a 9 definen una regla que puede leerse como: `T1` está contenido en `T2` si `T1` está en ubicado en `T2` o si `T1` está contenido en `X` y `X` está ubicado en `T2`. Por último las líneas 11 y 12 muestran la ejecución de la consulta. Allí se aprecia que si bien no hay ningún hecho que indique `location(knife, house).`, el mecanismo de encadenamiento hacia atrás de Prolog permite detectar que `knife` se encuentra en `house`.

Listado 2.1: Ejemplo en Prolog

```
1 location(office, house).
2 location(kitchen, house).
3 location(knife, kitchen).
```

```
4
5  is_contained_in(T1,T2) :-
6    location(T1,T2).
7  is_contained_in(T1,T2) :-
8        location(X,T2),
9        is_contained_in(T1,X).
10
11 ?- is_contained_in(knife, house).
12 yes
```

2.4.2. Jess

Jess (Java Expert System Shell) es un motor de inferencias que además provee un lenguaje interpretado para desarrollar aplicaciones que pueden realizar inferencias a partir de conocimiento proporcionado en forma de reglas. Posee un API en Java que permite acceder a todas las características de este lenguaje. Se distribuye en forma gratuita para uso académico, pero requiere de una licencia paga para utilizarlo en aplicaciones comerciales.

Jess utiliza una versión del algoritmo RETE para procesar las reglas [17]. Este algoritmo proporciona una solución eficiente al problema de emparejamiento (*matching*) entre los hechos presentes en la memoria de trabajo y las reglas.

En el Listado 2.2 se muestra un ejemplo sencillo de encadenamiento hacia adelante en Jess. Las líneas 1 a 9 muestran las reglas; la línea 12 el agregado de un hecho a la memoria de trabajo y la 13 la ejecución. Finalmente la línea 14 muestra el resultado de la inferencia.

Inicialmente la agenda contiene sólo el hecho `(location knife kitchen)` por lo que sólo se puede disparar la regla `r1`. Como resultado, se agrega a la memoria de trabajo el hecho `(location knife house)` que satisface la condición de la regla `r2` produciendo en encadenamiento.

Listado 2.2: Ejemplo de forward chaining en Jess

```
1  (defrule r1
2      (location knife kitchen)
3      =>
4      (assert (location knife house)) )
5
6  (defrule r2
7      (location knife house)
8      =>
9      ( printout t " knife is in the house " crlf) )
10 ---------------------------------------------------
11
12 Jess> (assert (location knife kitchen))
13 Jess> (run)
14 knife is in the house
```

En [17] se explica que si bien Jess es estrictamente un motor de inferencias hacia adelante, soporta también encadenamiento hacia atrás mediante un mecanismo especial. Para que se active este mecanismo hay que indicar cuales reglas se van a utilizar en el proceso de encadenamiento hacia atrás (con

el comando `do-backward-chaining rule_name`); de esta forma el motor inserta en la memoria de trabajo hechos disparadores (*trigger facts*) con el nombre `need-rule_name`. La idea es que estos hechos disparen reglas que modifiquen la memoria de trabajo de manera que se satisfaga el LHS de la regla original.

Listado 2.3: Ejemplo de bacward chaining en Jess

```
(do-backward-chaining location)
(defrule go
  (check knife)
  (location knife house)
  =>
  (printout t "knife is in the house" crlf)
)

(defrule makelocation
  (need-location ?x ?y)
  =>
  (assert (location ?x ?y)))

---

Jess> (assert (check knife))
Jess> (run)
knife is in the house
```

En el Listado 2.3 se muestra que al insertar el hecho `(check knife)`, Jess detecta que solo falta un hecho para satisfacer el LHS de la regla `go` y que este hecho está marcado para ser utilizado por el encadenamiento hacia atrás, así que genera un `(need-location knife house)`. Ese hecho hace que se dispare la regla `makelocation` que agrega en la memoria el hecho que necesita la primera regla para ejecutarse.

Una de las desventajas de este mecanismo es que hay que tener en cuenta en el modelado del sistema estos hechos disparadores y generar reglas especiales que los utilicen.

2.4.3. Drools

Drools por su parte es un sistema de gestión de reglas de negocio (BRMS[3]) que se distribuye bajo **Apache Software License 2.0**.

En cuanto a funcionalidad, mientras que comenzó como un motor de inferencias, Drools ha ido evolucionando para convertirse en un BRMS; esto implica que además del motor de inferencias posee módulos para desplegar, ejecutar y monitorizar la ejecución de nuevas reglas.

Como parte de esta evolución es que pasó a formar parte de un conjunto de proyectos de software libre llamado KIE[4] (Knowledge Is Everything) que agrupa tecnologías relacionadas a gestión y automatización de sistemas de negocios. Dentro de este grupo hay otros proyectos interesantes como: **jBPM** (una *suite* para manejo de procesos de negocios) y **OptaPlanner** (que permite optimizar la

[3]Business Rule Management System
[4]http://kiegroup.org/

planificación de los recursos de negocios gracias a que implementa un algoritmo para resolver problemas de satisfacción de restricciones).

La utilización de Drools abre la puerta para que en el futuro se puedan incorporar más fácilmente otras herramientas de la *suite* KIE para aumentar las funcionalidades de MAEOS. Se darán más detalles en la Sección 5.2.

En lo que respecta concretamente al motor de inferencias, Drools inicialmente implementaba el algoritmo de inferencias RETE, al igual que Jess. Posteriormente fue mutando, primero a una variante llamada RETEOO que incorpora una serie de optimizaciones en el manejo de los nodos de la red y nuevas funcionalidades (como *backward chaining*, la posibilidad de agregar/quitar reglas dinámicamente entre otras) y finalmente a una nueva implementación llamada PHREAK. Si bien PHREAK se basa en las implementaciones anteriores, por sus características y las mejoras que incorpora ya no puede clasificarse como una implementación de RETE.

Otra de las ventajas de Drools es que permite insertar objetos Java en la memoria de trabajo, o dicho de otra manera, se pueden utilizar *beans* como *facts* directamente. Además, en la escritura de las reglas se puede utilizar cualquier propiedad declarada en el objeto.

Estas propiedades que permiten utilizar el mismo lenguaje para la representación del conocimiento y para describir las reglas son fundamentales para evitar procesos de traducción, como se verá en la Sección 2.5.2.

En el Listado 2.4 se muestra el mismo ejemplo de encadenamiento hacia adelante del Listado 2.2 pero ahora con la sintaxis de Drools. Las líneas 1 a 13 muestran la definición de las reglas, luego se muestra parte del programa Java donde se inicializa la memoria de trabajo y se dispara la inferencia (líneas 15-16). En la línea 18 se ve la salida del programa y por último se ve el contenido de la memoria de trabajo al finalizar la inferencia (líneas 20-21).

Listado 2.4: Ejemplo de forward chaining en Drools

```
1  rule "r1"
2  when
3    Location(source == "knife", target == "kitchen" );
4  then
5       insert(new Location("knife", "house"));
6  end
7
8  rule "r2"
9  when
10   Location(source == "knife", target == "house");
11 then
12   System.out.println( "knife is in the house" );
13 end
14 ---------------------------------------------------------
15 kSession.insert( new Location("knife", "kitchen") );
16 kSession.fireAllRules();
17 ---------------------------------------------------------
18 knife is in the house
19
20 Location (knife, kitchen)
21 Location (knife, house)
```

El *backward chaining* descripto en la documentación oficial de Drools [46] opera en forma similar al de Prolog. Es decir, a partir de hechos en la memoria de trabajo que modelan el problema, se pueden definir consultas (*query*) para determinar que hechos satisfacen ciertas condiciones. Retomando el ejemplo del Listado 2.1, se muestra la misma consulta, esta vez con la sintaxis de Drools, en el Listado 2.5. Los resultados de las consultas pueden obtenerse mediante un programa en Java o como en caso del ejemplo anterior se puede utilizar la consulta como parte del LHS de una regla y acceder a los resultados en su RHS.

En el Listado 2.6 se insertan tres hechos en la sesión para describir el problema (líneas 1 a 3), uno para activar la regla (línea 4), y por último se lanza la inferencia.

En el resultado de la ejecución (Listado 2.7) se puede observar (línea 1) que se dispara la regla `go`, o sea que si bien no hay un hecho que indique `Location("knife", "house")`, y por lo tanto no puede alcanzar el objetivo en primera instancia; el algoritmo de *backtracking* puede actuar recursivamente hasta lograrlo. Así, el LHS de la regla resulta verdadero y se ejecuta el RHS.

Listado 2.5: Ejemplo de query para backward chaining en Drools

```
1  query isContainedIn(String x, String y)
2    Location(x, y;)
3    or
4    ( Location(z, y;) and isContainedIn(x, z;) )
5  end
6
7  rule "go"
8  when
9    String( this == "go" )
10   isContainedIn("knife", "house";)
11 then
12   System.out.println( "knife is in the house" );
13 end
```

Listado 2.6: Ejemplo de backward chaining en Drools

```
1  ksession.insert( new Location("office", "house") );
2  ksession.insert( new Location("kitchen", "house") );
3  ksession.insert( new Location("knife", "kitchen") );
4  ksession.insert( "go" );
5  ksession.fireAllRules();
```

Listado 2.7: Resultado de la ejecución

```
1  knife is in the house
```

Los listados anteriores muestran un ejemplo del funcionamiento del *backward chaining* en Drools. Se reanudará este tema en la Sección 3.3.

2.5. Modificaciones incorporadas en MAEOS2

Esta sección describe las modificaciones realizadas al proyecto utilizando los conceptos presentados en las secciones precedentes.

En la Sección 2.3 se describen los tipos de razonamientos. En el proyecto MAEOS2 se utiliza razonamiento deductivo descripto en la Sección 2.3.1 en el módulo de análisis (MAMAS) y está implementado mediante la estrategia de encadenamiento hacia adelante (Sección 2.2.2). Por otro lado, el razonamiento abductivo explicado en la Sección 2.3.3 se utiliza en el módulo de diagnóstico (PAPAS) y está implementado mediante encadenamiento hacia atrás (Sección 2.2.2).

El razonamiento inductivo no se utiliza actualmente pero completa la exposición del conjunto de razonamientos más importantes.

2.5.1. Modificaciones en la representación del conocimiento

En la Sección 1.4.1 se dió una definición general de los estados de una PyME; ahora con los conceptos definidos en las secciones anteriores se puede precisar más la descripción.

Al momento de comenzar este trabajo, el proyecto MAEOS utilizaba una representación del conocimiento basada en ontologías (Sección 2.2.5) que estaban implementadas en el lenguaje para representar ontologías de Protégé-Frames. En esta representación el estado de la empresa se describía instanciando los conceptos de las distintas ontologías y asignando valores a sus *slots*.

En MAEOS2, la versión actual del sistema, se transformó la representación del conocimiento a un modelo basado en orientación a objetos (Sección 2.2.4). Es decir, que ahora el estado de la empresa se describe con una instanciación de clases, esto es creando objetos y asignando valores a sus propiedades. Hay que remarcar que este cambio tiene más que ver con una cuestión de implementación que con una cuestión conceptual, ya que ambas representaciones comparten los conceptos de jerarquías de clases, propiedades, etc. Esto es importante porque el experto, que es quién habitualmente manipula la base de conocimiento ya se encuentra familiarizado con estas nociones.

Las ontologías que componían la base de conocimiento utilizaban sólo clases y *slots* para modelar el dominio; en término de los conceptos expuestos en la Sección 2.2.5 podría decirse que eran ontologías livianas. Estos conceptos encuentran sus equivalentes en las clases y atributos del modelo orientado a objetos respectivamente, por lo que la traducción es bastante directa. Además dado que no utilizaban otros componentes de las ontologías tales como los *facets* mencionados en la Sección 2.2.5, no hay pérdida de información al realizar la migración.

A nivel de implementación, esta transformación implicó migrar las ontologías existentes del formato de Protégé-Frames a Java; los detalles del proceso se tratan en la Sección 3.1. La principal ventaja de esta transformación es que proporciona una representación del conocimiento uniforme para las herramientas de las siguientes etapas del proyecto, concretamente para el motor de inferencias. Se profundizará sobre esta cuestión en la Sección 2.5.2. Otra ventaja se debe a la popularidad del lenguaje Java; en la actualidad existen una gran cantidad de herramientas libres para trabajar con este lenguaje (entornos de

desarrollo, modelado de datos, generación de diagramas, etc). Así, tener la base de conocimiento implementada en este lenguaje abre la puerta para en el futuro incorporar funcionalidades al proyecto más fácilmente. Por ejemplo, se podrían persistir los resultados en una base de datos relacional (como MySQL) utilizando algún *framework* que realice el mapeo objeto-relacional (como Hibernate).

También hay que mencionar la perdida de vigencia del proyecto Protégé-Frames ya que su última actualización fue en abril del 2013 [7]. Además las nuevas versiones de Protégé no soportan el esquema de *frames*; en su lugar los desarrolladores se han orientado al lenguaje de representación de ontologías OWL.

Descartamos la migración del lenguaje de *frames* a OWL ya que este paradigma está basado en la suposición de mundo abierto (*open-world assumption*) lo que significa que todo enunciado que no se haya declarado explícitamente como verdadero se considera que tiene un valor desconocido (en lugar de falso). En oposición, tanto Protégé-Frames como los lenguajes orientados a objetos se basan en la suposición de mundo cerrado (*closed-world assumption*) que implica que si no hay información suficiente para probar una sentencia como verdadera, entonces se asume que es falsa [47].

2.5.2. Reemplazo del motor de inferencias

En esta sección se explican las razones que motivaron el cambio del motor de inferencias y las ventajas que presenta el actual respecto al anterior.

Regresando al proceso descripto en la Sección 1.4.2 y teniendo en cuenta lo explicado en la Sección 2.5.1, en MAEOS el estado de la empresa se representaba como instancias de conceptos de las distintas ontologías en formato de Protégé-Frames.

El sistema multi-agentes (MAMAS) que internamente utilizaba Jess como motor de inferencias transformaba estas instancias en hechos (*facts*) y luego los resultados se volvían a convertir en instancias de Protégé-Frames para presentarlos al usuario.

En Protégé-Frames el conocimiento se representa con clases e instancias mientras que su correspondiente en Jess serían los templates y *facts* respectivamente. Por lo que parte del trabajo realizado por MAMAS era la conversión entre estos dos formatos.

Aquí observamos dos inconvenientes. Primero en términos de eficiencia computacional ya que había que convertir los datos ingresados al formato de Jess y posteriormente los resultados obtenidos al formato de Protégé-Frames.

El segundo de los inconvenientes es que el lenguaje de Protégé-Frames es más expresivo que el de Jess, es decir que hay características y funcionalidades del primero que no pueden ser representadas en el segundo. Este tipo de problema es lo que en [6] denominan **adecuación representacional** y la definen como la capacidad de representar y tratar con todas las clases de conocimiento que se precisan manipular en un dominio dado. La adecuación representacional es una propiedad deseable de un sistema basado en conocimiento, es decir debe existir una correspondencia entre la representación en el nivel del conocimiento y en el simbólico.

Para salvar estas diferencias en [2] se tomaron varias decisiones de diseño, entre las cuales una de las más controversiales es la forma de representar los *slots* de tipo *Instance* de Protégé-Frames. Este tipo de *slot* permite modelar que

una instancia está compuesta por otras instancias de su base de conocimiento. Utilizando los conceptos definidos en la Sección 2.2.4 se podría decir permite expresar relaciones de **agregación**.

Para representar este tipo en el lenguaje de Jess en [2], agregaron a cada hecho (*fact*) un *slot* numérico llamado `ID` y un mecanismo para asignar un valor diferente de ese slot a cada *fact* que se agregaba a la memoria de trabajo.

Por ejemplo la regla del Listado 2.8 expresa que en la *elaboración de la estrategia* debe haber una instancia de *actor* que sea *dirigente* (notar el uso de la variable `?x` en la línea 5). La desventaja de esta solución es que requiere de gran cuidado por parte del experto que escribe las reglas ya que no se puede garantizar que se cumpla el dominio de clases aceptado definido en Protégé-Frames sobre el *slot* de tipo *Instance*.

Listado 2.8: Regla Jess

```
1  (defrule regle_1 "boissin_et_al.,_2003_p4"
2    (dirigeant (ID ?x))
3    (elaboration_de_la_strategie
4           (Ssregle obs)
5           (Sacteur ?x))
6    (not (croissance_de_l_entreprise
7           (Name regle_1_boissin_2003)
8           (Ssregle const100)
9           (Sevolution_future croissant)))
10   =>
11   (assert (croissance_de_l_entreprise
12          (Name regle_1_boissin_2003)
13          (Ssregle const100)
14          (Sevolution_future croissant)
15          (ID (getid)) )
16   )
17 )
```

Como se observa por ejemplo en las líneas 2 y 15 del Listado 2.8 los problemas de representación impactan en la redacción de las reglas influyendo directamente en la eficacia y eficiencia de la solución lograda.

Se resolvieron estos problemas reemplazando el motor de inferencias Jess por Drools que permite utilizar instancias de objetos Java como *facts* que pueden insertarse directamente en la memoria de trabajo. De esta forma se logra una representación del conocimiento uniforme, evitando procesos de traducción. Dicho de otra manera, se evitan los problemas de adecuación representacional.

Obviamente hubo que traducir las reglas a la sintaxis de Drools. Se darán más detalles sobre este proceso en la Sección 3.2.

La transformación de la base de conocimientos a un lenguaje orientado a objetos conlleva varias virtudes. Por un lado, representar la **agregación** no presenta ningún inconveniente. Puesto que Java es un lenguaje estáticamente tipado[5] se pueden detectar los errores al asignar un objeto a una propiedad de otro en el momento de la compilación. De hecho, durante la realización de este trabajo se detectaron varias inconsistencias en las reglas que habían pasado

[5] cada variable debe ser declarada con un tipo que no puede alterarse durante su ciclo de vida

inadvertidas.

Además por ser fuertemente tipado[6] se simplifica el trabajo del experto ya que incluso el mismo IDE de desarrollo puede sugerir el tipo de objeto cuando se escribe una regla.

Otra cuestión que motivó el cambio del motor de inferencias es la licencia. Como se mencionó en la Sección 2.4.2, Jess es un motor de inferencias que se distribuye con licencia privativa, lo cual genera varios problemas.

Por un lado si bien permiten su uso para fines educativos y gubernamentales, no está permitido su uso para fines comerciales. Esto es importante si en el futuro el proyecto MAEOS pretende ser utilizado en la industria.

Por otro lado se nos planteaba un dilema moral, ya que sostenemos que no deberían desarrollarse trabajos en el ámbito académico basados en librerías que no posean licencias libres. Tal como se define en [45] un programa que no posee una licencia libre, impide la investigación por violar las libertades 0 (de ejecutar el programa como se desea, con cualquier propósito) y 1 (de estudiar cómo funciona el programa, y cambiarlo para que haga lo que se quiera; el acceso al código fuente es una condición necesaria para ello).

Drools por su parte, tal como se expuso en 2.4.3 se distribuye bajo **Apache Software License 2.0** que una licencia de software libre y como tal otorga al usuario las cuatro libertades mencionadas en [45].

2.6. Resumen

En la siguiente tabla se resumen las modificaciones realizadas al proyecto MAEOS en los distintos componentes del sistema.

	MAEOS	MAEOS2
Representación del conocimiento	Ontologías	Diseño orientado a objetos
Lenguaje de la KB	Protégé-Frames	Java
Motor de inferencias	Jess	Drools
Razonamiento deductivo	MAMAS	MAMAS2
Razonamiento abductivo	-	PAPAS

[6] dada una variable de un tipo concreto, no se puede usar como si fuera de otro tipo distinto a menos que se haga una conversión explicita

Capítulo 3

Implementación

Programs must be written for people to read, and only incidentally for machines to execute.

- Harold Abelson

En este capítulo se exponen en detalle los aspectos técnicos de la implementación de los cambios mencionados en la Sección 1.5. Está estructurado de la siguiente forma, por cada modificación primero se describe el estado del arte y en la sección siguiente las modificaciones realizadas junto con las ventajas que presenta sobre la implementación anterior.

En la Sección 3.3 se detalla el desarrollo del módulo PAPAS que aumenta la funcionalidad existente en MAEOS y por lo tanto no tiene equivalente en la versión anterior del sistema.

3.1. Migración de las ontologías

En la Sección 2.5.1 se explicaron las razones que motivaron el cambio de representación de los datos. A nivel de implementación este cambio implicó migrar las ontologías existentes del formato Protégé-Frames a Java.

Se transformó cada concepto de las ontologías en un *Bean*. Un *Bean* de Java es una clase que por cada propiedad tiene un par de métodos para acceder a ella, uno para obtener el valor (generalmente llamado get*NombreDeLaPropidedad*) y otro para modificar el valor de la misma (generalmente llamado set*NombreDeLaPropidedad*).

El proyecto MAEOS cuenta con diferentes ontologías, donde cada una representa un área específica en el dominio de gestión empresarial [2].

Se utilizó el nombre de cada ontología como nombre de un paquete para generar una estructura de clases, conservando así la independencia de los conceptos que es uno de los objetivos de diseño de MAEOS: mantener la pluralidad y poder expresar así los diferentes puntos de vista. En la Figura 3.1 se muestra la jerarquía de paquetes resultante y el contenido de dos de los mismos.

Otra de las decisiones de diseño que se tomaron fue que el código de los *beans* generados tenía que ser "legible", es decir debía verse "como si" lo hubiera escrito una persona. Esta decisión no obedece sólo a una cuestión estética, sino que al ser independientes de cualquier *framework* permite que sean reutilizados por

Figura 3.1: Jerarquía de clases obtenida

cualquier herramienta en el futuro. A raíz de esta decisión quedaron descartadas varias herramientas de generación de código que en general producen clases que extienden de alguna del propio *framework*.

Por ejemplo, una de las herramientas analizadas fue JSave[1] que es un *plugin* de Protégé-Frames que permite generar clases Java a partir de ontologías en formato Protégé-Frames (su última actualización fue en 2008). Sin embargo, no se obtuvieron resultados satisfactorios por lo que finalmente se optó por desarrollar una solución *ad hoc*.

El programa desarrollado se ajustó para realizar la migración de las ontologías existentes; no se pretendió hacer una herramienta de uso general ya que esta traducción es un proceso que se ejecuta una sola vez.

El programa está contenido en la clase `Ontology2Bean` y su funcionamiento consiste en *parsear* el archivo que contiene la ontología (de extensión *.pont*) utilizando el API de Protégé. Para almacenar los resultados de este proceso se crearon unas estructuras intermedias que contienen los datos de la clase y sus atributos. Cuando el *parser* detecta un concepto de una ontología que puede ser modelado como un *bean* de java crea una de estas estructuras llamada `BeanModel` para almacenar los atributos, tipos de datos, etc. En la Figura 3.2 se puede ver el diagrama del proceso descripto.

Además, si durante este proceso detecta un tipo de dato que tiene valores constantes crea otra estructura diferente llamada `EnumModel`. Ésta dará lugar a un tipo especial de clase java que se indica como `enum`.

Una vez obtenidos los BeanModel que representan las clases de la ontología, se pasan como parámetros a la siguiente etapa del proceso que es la generación de los archivos java que tienen el código fuente del bean. Para esto se evaluaron numerosas librerías que permiten generar código fuente a partir de un *template* (por ejemplo: Stratego/XT[2], StringTemplate[3]).

Finalmente se decidió utilzar una llamada JET[4] (Java Emitter Templates) por su facilidad de uso. A diferencia de las otras librerías mencionadas que tienen sus propios dialectos para definir los *templates*, JET utiliza la misma sintaxis de las páginas JSP (JavaServer Pages) con la que ya estábamos familiarizados.

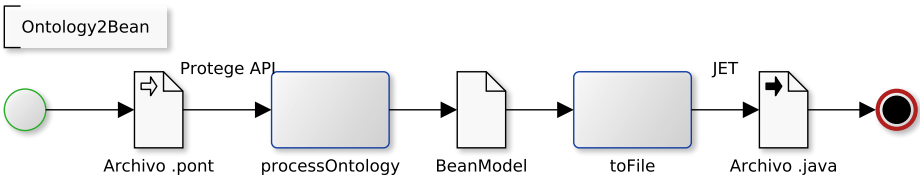

Figura 3.2: Proceso de conversión de ontologías a beans

[1] http://protegewiki.stanford.edu/wiki/JSave
[2] http://strategoxt.org/
[3] http://www.stringtemplate.org/
[4] https://eclipse.org/modeling/m2t/?project=jet y https://eclipse.org/articles/Article-JET/jet_tutorial1.html

Como se ve en el Listado 3.1, un *template* de JET consiste en un archivo de texto con código java intercalado. El *template* recibe los parámetros en la variable `argument` que en este caso es de tipo `BeanModel` (línea 5). Además se pueden usar instrucciones de control de flujo, como en la línea 19 para iterar la lista de propiedades del bean y así generar los *getter* y *setters* correspondientes.

Listado 3.1: Parte del template de JET para generar un bean

```
1  <%@ jet   package="ar.edu.unr.fceia.lcc.core"
2       imports="java.util.* ar.edu.unr.fceia.lcc.model.ontology
            .* org.apache.commons.lang3.text.WordUtils"
3       class="BeanTemplate" %>
4  <%
5       BeanModel bm = (BeanModel) argument;
6   %>
7   package ar.edu.unr.fceia.lcc.model.ontology.<%= bm.
        getOntologyName() %>;
8
9   import ar.edu.unr.fceia.lcc.model.ontology.*;
10
11  public class <%= bm.getClassName() %>
12       <%= bm.getSuperClassName().equals("") ? "" : "extends " +
             bm.getSuperClassName() %> {
13
14       private String name;
15
16  <%
17       Set<String> keys = bm.getProperties().keySet();
18
19       for (String property : keys) {
20       String type = bm.getProperties().get(property);
21       String methodName =  WordUtils.capitalize(property);
22   %>
23
24       private <%= type %> <%= property %>;
25
26       public <%= type %> get<%= methodName %>() {
27            return <%= property %>;
28       }
29
30       public void set<%= methodName %>(<%= type %> <%= property
             %>) {
31            this.<%= property %> = <%= property %>;
32       }
33
34                                  .
35                                  .
36                                  .
```

A partir del *template* del Listado 3.1 se obtiene un *bean* como el del Listado 3.2.

Listado 3.2: Parte de un bean generado a partir del template

```
1  package ar.edu.unr.fceia.lcc.model.ontology.boissin;
2
3  import ar.edu.unr.fceia.lcc.model.ontology.*;
4
5  public class ChefDEntreprise extends Acteur {
6
7       private String name;
8
9       private Boolean remplir;
10
11      public Boolean getRemplir() {
12           return remplir;
13      }
14
15      public void setRemplir(Boolean remplir) {
16           this.remplir = remplir;
17      }
18
19      private Sregle sregle;
20
21      public Sregle getSregle() {
22           return sregle;
23      }
24
25      public void setSregle(Sregle sregle) {
26           this.sregle = sregle;
27      }
28
29                                  .
30                                  .
31                                  .
```

Como los archivos generados se graban dentro del proyecto, el entorno de desarrollo (Eclipse) los detecta y compila automáticamente permitiendo descubrir inmediatamente cualquier error en el proceso de traducción.

3.2. Migración del motor de inferencias

En la Sección 2.5.2 se explicaron las razones que motivaron el cambio del motor de inferencias. En forma concisa son:

- unificar el formato de representación del conocimiento para evitar traducciones entre distintas implementaciones.
- resolver problema de representación de la agregación debido a la diferencia de expresividad de los lenguaje utilizados.

Como consecuencia del reemplazo del motor de inferencias hubo que reimplementar MAMAS.

El primer paso fue traducir toda la base de reglas de la sintaxis de Jess a la de Drools. Puesto que, al igual que en caso de la traducción de las ontologías, es un proceso que se ejecuta una sola vez, en este caso también se desarrolló un programa *ad hoc* para esta tarea.

Como se puede ver en la Figura 3.3, el diseño de este programa es similar al anterior, es decir, primero se *parsean* los archivos que contienen las reglas (con extensión *.rules*) utilizando el API de MAMAS, que a su vez utiliza el API de Jess. Este proceso genera una estructura intermedia llamada `RuleModel` para almacenar los elementos constituyentes de la regla. Finalmente esta estructura se pasa como parámetro al método que produce el archivo de reglas de Drools (con extensión *.drl*).

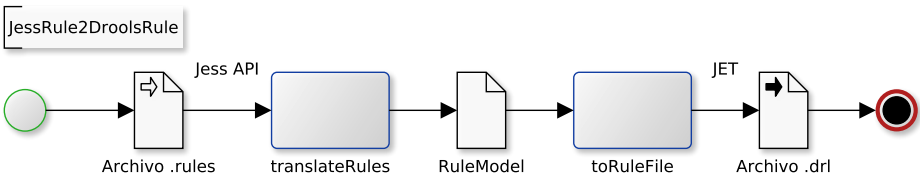

Figura 3.3: Proceso de conversión de reglas Jess a reglas Drools

Aquí también se utilizó JET para generar las reglas a partir de un *template*. Parte del mismo se puede ver en el Listado 3.3.

Listado 3.3: Parte del template de JET para generar una regla de Drools

```
1  <%@ jet package="ar.edu.unr.fceia.lcc.core" imports="java.util
       .* ar.edu.unr.fceia.lcc.model.ontology.*" class="
       RuleOntoTemplate" %>
2
3  package ar.edu.unr.fceia.lcc.rules;
4
5  import ar.edu.unr.fceia.lcc.model.ontology.*;
6  <%
7        ArrayList<RuleModel> lrm = (ArrayList<RuleModel>)
             argument;
8   %>
9  import ar.edu.unr.fceia.lcc.model.ontology.<%= lrm.get(0).
       getOntologyName() %>.*;
10 <%
11       for (RuleModel rm : lrm) {
12  %>
13
14  rule "<%= rm.getRuleName() %>"
15       when
16 <%
17       for (BeanModel bm : rm.getLhs()) {
18            String variableName = bm.getVariableName();
19  %>
20            $<%= variableName %>      : ar.edu.unr.fceia.lcc.
                  model.ontology.<%= bm.getOntologyName() %><%=
                  bm.getClassName() %>(
21 <%
```

```
22              Set<String> keys = bm.getProperties().keySet();
23              int i = 0;
24              for (String k :keys) {
25                   String v = bm.getProperties().get(k);
26                   String [] vs = v.split(" ");
27   %>
28                  <%= (vs.length > 1) ? (vs[1] + " " + vs[0] + "
                         " + vs[2]) :
29                                (k + " == " + v) %><%= ((++i) ==
                                     keys.size() ? "" : ", ")%>
30   <%
31              }    //   for keys
32   %>
33              );
34   <%
35         }    //   for lhs
36   %>
37         then
38
39
40                                          .
41                                          .
42                                          .
```

Por ejemplo, en el Listado 3.4 se puede ver la la regla Drools generada para la regla Jess mostrada en el Listado 2.8. Como se puede observar se resuelve el inconveniente de la representación de los *slots* de tipo *Instance* mencionado en la Sección 2.5.2. En la línea 3 queda explícitamente representado el tipo de dato de la variable `dirigeant` que luego se compara en la línea 5 con la propiedad `acteur` del objeto `ElaborationDeLaStrategie`. De esta forma se obtiene una mayor claridad en la regla y se elimina la necesidad de utilizar el artificio mencionado.

Listado 3.4: Regla en formato Drools

```
1  rule "regle_1 boissin_et_al.,_2003_p4"
2    when
3       $dirigeant      : Dirigeant();
4       $edls           : ElaborationDeLaStrategie(
5                          acteur == $dirigeant,
6                          sregle == Sregle.OBS
7                       );
8    then
9       CroissanceDeLEntreprise cdle =
10                     new CroissanceDeLEntreprise();
11      cdle.setEvolutionFuture(EvolutionFuture.CROISSANT);
12      cdle.setName("Regle1Boissin2003");
13      cdle.setSregle(Sregle.CONST100);
14      cdle.setRemplir(true);
15
16         insert(cdle);
17 end
```

Al comparar las dos reglas queda en evidencia otra de las ventajas de utilizar Drools. Todas las reglas de la base de reglas en formato Jess tienen una cláusula *not* en el **LHS** que es igual al hecho que se agrega en el **RHS**. Esto se hacía para evitar que la misma regla se disparara reiteradamente, pero tiene la desventaja de hacer las reglas mas difíciles de mantener ya que si lo que está en la cláusula *not* no es exactamente igual al hecho del RHS no se logra el resultado esperado.

Drools posee un parámetro de configuración que permite definir cuando dos hechos de la memoria de trabajo se consideran iguales, pudiendo utilizarse dos criterios:

***identity*:**
: que distingue entre objetos comparando sus direcciones de memoria.

***equality*:**
: que utiliza los métodos *equals* y *hashCode* del objeto para determinar si son iguales.

Configurando el proyecto con el parámetro `equalsBehavior="equality"` se pudo eliminar la cláusula *not* de las reglas, ganando en legibilidad y evitando una posible fuente de errores. Esto muestra otra de las ventajas de utilizar un motor de inferencias orientado a objetos, ya que tiene en cuenta la propiedad **identidad** mencionada en la Sección 2.2.4.

3.3. Módulo de razonamiento abductivo

Este módulo permite generar recomendaciones para orientar al titular de la PyME hacia las condiciones que debe crear en la empresa para alcanzar un **estado final** deseado.

Notar que se hace referencia a "un" estado ideal y no a "el" estado ideal, ya que el razonamiento abductivo permite conjeturar el estado más probable pero no garantiza que sea el único.

Como se explica en la Sección 2.2.2 el encadenamiento hacia atrás parte de un objetivo (en este caso el **estado final**) a partir del cual intenta determinar las condiciones para alcanzarlo. Al finalizar este proceso se obtiene un conjunto de condiciones que denominamos **estado ideal** que permiten suponer que si la evolución es la predicha por las reglas, se alcanzará el estado final.

Para desarrollar este módulo se concibió una manera de aprovechar el compendio de conocimiento ya disponible en forma de reglas y las posibilidades del motor de inferencias de realizar encadenamiento hacia atrás (*backward chaining*).

Esta idea tiene varias ventajas: por un lado permite sacarle el máximo provecho a la formalización de conocimiento existente al reutilizar las mismas reglas que se habían utilizado en la etapa de inferencia hacia adelante; por otro lado, mantiene la integridad conceptual del proyecto MAEOS al utilizar el mismo enfoque en las distintas etapas lo que termina simplificando la arquitectura del sistema.

El funcionamiento esperado era poder ingresar el **estado final** para que Drools reconozca en las conclusiones de las reglas los valores ingresados, agregue como nuevos objetivos las condiciones de esas reglas y continúe con el proceso recursivamente hasta que ya no pueda inferir más.

Sin embargo, al momento de implementar el proceso descripto surgieron varios inconvenientes. El primero tiene que ver con la forma en que está implementado el *backward chaining* en Drools, la cual se describió en la Sección 2.4.3.

El problema es que esta operatoria no es compatible con el formato de las reglas existentes en la base de reglas. Una regla típica del proyecto MAEOS es por ejemplo la del Listado 3.4 que tiene una o más condiciones en su LHS, e inserta uno o más hechos en la memoria de trabajo en su RHS.

En este punto resultó evidente que si bien conceptualmente era correcto aplicar *backward chaining*, en la práctica no sería posible hacerlo en forma directa.

Aquí se evidenció otra de las ventajas de utilizar *software* de código abierto: afortunadamente el proyecto Drools cuenta con una comunidad de desarrollo muy activa, a la que se acudió en busca de consejos acerca de como proceder en esta situación. A los pocos días se obtuvo una respuesta[5] de uno de los lideres del proyecto que confirmó que no se iban a poder obtener los resultados esperados por este camino y sugirió investigar la variante de razonamiento abductivo del encadenamiento hacia atrás. Esta variante no se menciona en la documentación oficial porque aún es una característica experimental que está en desarrollo, pero se puede acceder al código fuente del proyecto para estudiarla.

En esta forma de encadenamiento hacia atrás de Drools se pueden definir consultas que inserten objetos en la memoria de trabajo cuando sus condiciones se satisfacen. Es decir, opera en forma similar al encadenamiento hacia atrás de Jess comentado en la Sección 2.2.2, pero está mejor integrado y no requiere ningún artilugio para su utilización.

En el Listado 3.5 mostramos un ejemplo de una consulta abductiva.

Listado 3.5: Consulta abductiva en Drools

```
1  query make( String $room1, String $room2 )
2       @Abductive( target=Location.class )
3       not Location(source == $room1, target == $room2)
4  end
5
6  rule "go"
7  when
8    String( this == "go" )
9    make("knife", "house";)
10 then
11   System.out.println( "knife is in the house" );
12 end
```

La semántica de esta consulta es la siguiente: si se satisface el cuerpo de la consulta de la línea 3 (o sea si no hay una ubicación con esos parámetros), se inserta en la memoria de trabajo el objeto especificado (línea 2). El objeto se crea usando un constructor con la misma firma (i.e., con la misma cantidad y tipos de argumentos) que la de la consulta, en este caso, esto implica que la clase `Location` deberá tener un constructor de la forma `Location(String x, String y)`. El resultado de la ejecución es similar al del ejemplo anterior, con la diferencia que en la memoria de trabajo hay un objeto

[5] http://stackoverflow.com/q/28010926/1399001

más: `Location ("knife", "house")` insertado por la consulta abductiva `make`.

Como puede verse en el Listado 3.5 la forma de utilizar la consulta abductiva es diferente a la de las reglas existentes; por lo que para poder utilizar esta variante del encadenamiento hacia atrás hubo que generar reglas que incluyan consultas abductivas en sus LHS.

Se desarrolló entonces un proceso que por cada regla utilizada por la inferencia hacia adelante, genera **automáticamente** una para ser utilizada por la inferencia hacia atrás. En la Figura 3.4 se puede notar que para este proceso se utilizó la misma arquitectura que en las traducciones anteriores. Es decir, se generan estructuras intermedias (las clases `AbductiveQueryModel`) que se utilizan como entrada para generar las reglas abductivas a partir de un *template* realizado con JET. De esta forma, se mantuvo el objetivo mencionado en las secciones anteriores de que las reglas resultantes debían ser "legibles", es decir debían verse "como si" las hubiera escrito un humano.

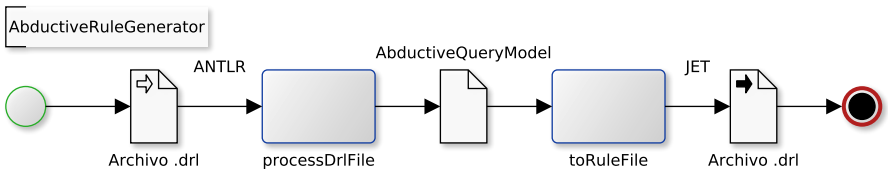

Figura 3.4: Proceso de generación de reglas abductivas

Para implementar este proceso hubo que obtener los objetos utilizados tanto en el LHS como en el RHS de cada regla. Si bien la primer parte es posible, se tropezó con el inconveniente que no hay forma de acceder mediante el API pública de Drools al RHS de las reglas (vale la pena mencionar que esto también fue validado realizando otra consulta[6] a los desarrolladores de Drools).

Para salvar este escollo no quedó más remedio que *parsear* el código fuente de las mismas. Este parseo es de índole distinta a los descriptos en las Secciones 3.1 y 3.2 que sólo se ejecutan una vez y con datos existentes. En este caso se tuvo en cuenta que el hecho que la base de reglas puede evolucionar, con lo cual por un lado debe ser lo suficientemente general para poder analizar nuevas reglas y por otro debe formar parte de PAPAS ya que hay que ejecutarlo cada vez que se produzca una modificación.

Para que el *parser* sea lo suficientemente general se construyó una gramática para las reglas y luego se utilizó la herramienta ANTLR[7] para generar un *parser* que reconoce esa gramática. ANTLR es *software* libre y es ampliamente usado tanto en el ambiente académico como en la industria. En el Listado 3.6 mostramos algunas reglas de la misma.

Listado 3.6: Parte de la gramática desarrollada para generar las reglas abductivas

```
1 droolRule
2       : 'rule' ruleName 'when' ruleWhen 'then' ruleThen 'end'
3       ;
```

[6] http://stackoverflow.com/q/28596820/1399001
[7] http://www.antlr.org/

```
4  ruleName
5        : StringLiteral
6        ;
7  ruleWhen
8        :  lhsVariableDeclaration+
9        ;
10 lhsVariableDeclaration
11       : (variableReference ':')? qualifiedName '('
             expressionList? ')' ';'
12       ;
13 ruleThen
14       :    insertBlock+
15       ;
16 insertBlock
17       :    rhsVariableDeclaration (setStatement)*
             insertStatement
18       ;
19
20 rhsVariableDeclaration
21       :    qualifiedName Identifier '=' 'new' qualifiedName '('
             ')' ';'
22       ;
23 setStatement
24       :    Identifier '.set' Identifier '(' primary ')' ';'
25       ;
26 insertStatement
27       : 'insert' '(' Identifier ')' ';'
28       ;
```

A partir de un archivo con la gramática que especifica el lenguaje a reconocer, en este caso el de las reglas de inferencia, ANTLR genera automáticamente programas que permiten recorrer el árbol sintáctico resultante basados en los patrones de diseño *Visitor* y *Listener* [36]. Extendiendo estos programas se puede ejecutar código específico de la aplicación; en este caso para generar las reglas abductivas.

En el Listado 3.7 muestra parte de la implementación que genera las reglas abductivas donde se puede ver que está basada en la versión que extiende del patrón *Listener* (lo que cambia respecto al patrón *Visitor* es esencialmente la manera en que se recorre el árbol). Con este patrón ANTLR genera métodos *enter* y *exit* para cada regla de la gramática y sólo es necesario implementar los métodos de la super clase a los que se le quiere agregar comportamiento.

Una ventaja de este diseño es que se aísla el código de la aplicación en el *Listener* lo que hace a la gramática reusable para otras aplicaciones.

Listado 3.7: Parte de la clase que genera las reglas abductivas

```
1  public class AbductiveQueryGenerator extends
       DroolsRulesBaseListener {
2
3        public void enterLhsVariableDeclaration(
             LhsVariableDeclarationContext ctx) {
4            String className = ctx.qualifiedName().getText();
5
```

```
6             if (ctx.variableReference() != null)
7                   ruleVariables.put(ctx.variableReference().
                        getText(), className);
8
9             BeanModel bm = new BeanModel();
10            bm.setClassName(className);
11            current.getAbducibles().add(bm);
12      }
13
14      public void enterRhsVariableDeclaration(
            RhsVariableDeclarationContext ctx) {
15            String fullClassName = ctx.qualifiedName().get(0).
                  getText();
16
17            BeanModel bm = new BeanModel();
18            bm.setClassName(fullClassName);
19            current.getConditions().add(bm);
20      }
21
22      public void enterSetStatement(SetStatementContext ctx) {
23            List<BeanModel> conditions = current.getConditions()
                  ;
24            BeanModel bm = conditions.get(conditions.size()-1);
25
26            String field = WordUtils.uncapitalize(ctx.Identifier
                  (1).getText());
27            String value = ctx.primary().getText();
28
29            VariableReferenceContext vrc = ctx.primary().
                  variableReference();
30
31            if ( vrc != null ) {
32                  bm.getProperties().put(value, ": " + field);
33            } else {
34                  bm.getProperties().put(field, value);
35            }
36
37      }
```

El Listado 3.8 muestra el *template* utilizado para generar las reglas abductivas. Recibe como entrada una lista de `AbductiveQueryModel`, donde cada uno contiene información de la regla a generar en forma de dos listas:

- `conditions` con los objetos y los valores de sus propiedades que formaban parte del RHS de la regla original.
- `abducibles` con los nuevos objetivos y los valores de sus propiedades que eran parte del LHS de la regla original.

Listado 3.8: Template que genera las reglas abductivas

```
1 <%@ jet package="ar.edu.unr.fceia.lcc.abductive" imports="java
      .util.* ar.edu.unr.fceia.lcc.model.*" class="
      AbductiveRuleTemplate" %>
```

```
2
3  package ar.edu.unr.fceia.lcc.rules;
4
5  import ar.edu.unr.fceia.lcc.model.ontology.*;
6  import ar.edu.unr.fceia.lcc.model.Goal;
7  import java.util.HashMap;
8
9  <%
10       ArrayList<AbductiveQueryModel> lqm = (ArrayList<
             AbductiveQueryModel>) argument;
11 %>
12
13 <%
14       for (AbductiveQueryModel qm : lqm) {
15 %>
16
17 rule "<%= qm.getRuleName() %>"
18       when
19 <%
20       for (BeanModel bm : qm.getConditions()) {
21            String variableName = bm.getVariableName();
22 %>
23            $<%= variableName %> : <%= bm.getClassName() %>(
24                 <%
25                 Set<String> keys = bm.getProperties().keySet();
26                 int i = 0;
27                 for (String k :keys) {
28                      String v = bm.getProperties().get(k);
29                 %>
30                 <%= v.startsWith(":") ? (k + " : " + v.
                       substring(1)) : (k + " == " + v) %><%= ((++
                       i) == keys.size() ? "" : ", ")%>
31            <%
32                 }     //    for keys
33            %>
34            );
35 <%
36       }     //    for conditions
37 %>
38
39 <%
40            for (BeanModel bm : qm.getAbducibles()) {
41            String variableName = bm.getVariableName();
42 %>
43            $args_<%= variableName %> : HashMap() from [
44            <%
45                 Set<String> keys = bm.getProperties().keySet();
46                 int i = 0;
47                 for (String k : keys) {
48                      String v = bm.getProperties().get(k);
49            %>
50                      "<%= k %>" : <%= v %><%= ((++i) == keys.
                            size() ? "" : ", ")%>
51            <%
```

```
52                  }     //    for keys
53              %>
54              ]
55              $a<%= variableName %> : goal(<%= bm.getClassName()
                    %>.class, $args_<%= variableName %>;)
56  <%
57          }     //    for abducibles
58   %>
59          then
60  <%
61              for (BeanModel bm : qm.getAbducibles()) {
62              String variableName = bm.getVariableName();
63   %>
64              insert($a<%= variableName %>.build());
65  <%
66              }     //    for abducibles
67   %>
68   end
69  <%
70          }     //    for lqm
71   %>
```

En el Listado 3.9 se muestra la regla abductiva obtenida a partir del template del Listado 3.8 para la regla del Listado 3.4. La regla puede interpretarse de la siguiente manera: si hay en la memoria de trabajo un dirigente (línea 7) y se busca una evolución a futuro con crecimiento de la empresa (líneas 8 a 11), entonces agregar como subobjetivo el hecho que en la elaboración de la estrategia (línea 18) el dirigente debe ser un actor principal (línea 15).

Listado 3.9: Regla abductiva correspondiente a la mostrada en el Listado 3.4

```
1   query goal(Class $t, HashMap $params)
2       @Abductive( target=Goal.class )
3   end
4
5   rule "Abductive regle_1 boissin_et_al.,_2003_p4"
6       when
7           $dirigeant : Dirigeant();
8           $cdle : CroissanceDeLEntreprise(
9                   remplir == true,
10                  sregle == Sregle.CONST100,
11                  evolutionFuture == EvolutionFuture.CROISSANT
12          );
13
14          $args_edls : HashMap() from [
15                  "acteur" : $dirigeant,
16                  "sregle" : Sregle.OBS
17          ]
18          $aedls : goal(ElaborationDeLaStrategie.class,
                $args_edls;)
19      then
20          insert($aedls.build());
21  end
```

En las líneas 1 a 3 del Listado 3.9 se puede ver la *query* abductiva genérica que se utiliza para todas las reglas. En la regla mostrada se utiliza en la línea 18 para construir una instancia de `Goal` que en la línea 20 termina por insertar en la memoria una instancia de `ElaborationDeLaStrategie` con sus propiedades `acteur` y `sregle` inicializadas en `$dirigeant` y `Sregle.OBS` respectivamente.

Si no se hubiese utilizado este recurso, hubiera sido necesario crear una *query* abductiva por cada tipo abducido. Para el ejemplo sería una como la del Listado 3.10

Listado 3.10: Implementación deficiente de una query abductiva

```
1 query makeEDLS(Dirigeant $acteur, Sregle $sregle)
2      @Abductive( target=ElaborationDeLaStrategie.class )
3
4                                   ⋮
```

Esta sería una solución muy poco general ya que como se explicó anteriormente para crear el objeto especificado por `target` se utiliza un constructor con la misma firma que la de la consulta. En el ejemplo anterior, la clase `ElaborationDeLaStrategie` debería tener un constructor que reciba como argumentos `acteur` y `sregle`. Esto impondría restricciones innecesarias a los *beans*, y haría que sea muy difícil mantener las reglas.

En lugar de eso, se implementó una única *query* abductiva llamada `goal` que crea una clase `Goal` para representar un objetivo genérico. `Goal` posee un constructor que recibe como primer parámetro el tipo de objeto a crear y como segundo los valores de sus propiedades, es decir con la misma firma que la *query* `goal`. Con estos datos, mediante *reflection* puede construir una instancia del *bean* especificado e invocar a sus métodos *set* para asignar valores a las propiedades correspondientes.

Esta clase tiene un parámetro de tipo e implementa el patrón de diseño *factory* como se observa en el Listado 3.11. Tiene un método *build* (línea 14) que devuelve el objeto construido. En la línea 17 se crea una instancia de dicho objeto, a continuación en la línea 21 se crea el método *set* para luego aplicarselo al objeto.

Listado 3.11: Parte del código fuente de la clase Goal

```
1  public class Goal<T> {
2
3    private Class<T> type;
4
5    private HashMap<String, Object> params;
6
7
8    public Goal(Class<T> type, HashMap<String, Object> params)
9    {
10     this.type = type;
11     this.params = params;
12   }
13
14   public T build() {
15     T t = null;
```

```
16      try {
17      t = type.newInstance();
18      for (Entry<String, Object> property : params.entrySet()) {
19          Method method;
20          if (property.getValue() != null) {
21            method = type.getMethod( "set" + WordUtils.
                  capitalize(property.getKey()),
22                      property.getValue().getClass() );
23            method.invoke(t, property.getValue());
24          }
25      }
26
27                                      ⋮
28      return t;
29    }
30
31                                      ⋮
```

La ventaja de usar esta clase genérica es que independiza a los objetos abducidos del uso que se les de en las reglas. Esto quiere decir que una modificación en la base de reglas no impacta en la representación de los datos.

El resultado de este proceso es un conjunto de archivos con las reglas abuctivas. Este conjunto constituye la base de conocimiento de PAPAS. La otra parte del sistema experto, es el mecanismo descripto de inferencia hacia atrás de Drools.

Con estos elementos se completa el desarrollo de PAPAS que permite lograr la funcionalidad esperada mencionada al comienzo de esta sección. Concretamente, permite obtener un posible *estado ideal* a partir de un **estado final** ingresado por el usuario.

La siguiente etapa consiste en utilizarlo en un caso real para validar su funcionamiento. Este es el tema del Capítulo 4.

3.4. Resumen

En la siguiente tabla se muestran algunas cifras resultantes de los procesos de conversión descriptos en las Secciones 3.1, 3.2 y 3.3.

Cantidad de	
clases	341
reglas de MAMAS 2	263
reglas de PAPAS	263

Capítulo 4

Resultados

En este capítulo se detallan los resultados obtenidos al aplicar los módulos desarrollados a caso real. Los datos utilizados fueron suministrados por el consultor y corresponden a una empresa con sede en Estrasburgo que desarrolla su actividad en el rubro de servicios eléctricos.

El consultor concurrió a la empresa "Electricité Services", realizó un relevamiento, y obtuvo un documento en el que relaciona sus observaciones con los conceptos del modelo de datos. Este proceso se describe en detalle en [2]. A partir de esto se realizó la instanciación del *estado inicial* que son los datos que necesita el módulo MAMAS.

4.1. MAMAS

Como consecuencia del cambio en la representación de los datos (explicado en la Sección 2.5.1) y del reemplazo del motor de inferencias por Drools (explicado en la Sección 2.5.2) se debió reimplementar el módulo de análisis (MAMAS).

Para comprobar el funcionamiento de los procesos de traducción (de las ontologías a *java beans* y de las reglas en formato Jess al de Drools) y del nuevo motor de inferencias; se construyó el mismo caso de prueba para ambas versiones de la aplicación.

Esto significa que se representó el *estado inicial* mediante una instanciación de clases *java* (para la nueva versión) y mediante una instanciación de ontologías *frames* (para la versión anterior). Luego se ejecutaron las inferencias en los respectivos motores obteniendose así el *estado actual* en sus dos representaciones. En la Tabla 4.1 se contrastan los resultados producidos.

Cantidad de instancias del estado	MAMAS	MAMAS 2
inicial	16	16
actual	145	100

Al inspeccionar exhaustivamente los resultados se pudo observar que los objetos, junto con los valores de sus propiedades, obtenidos por la inferencia hacia adelante de Drools coincidían con los obtenidos en la versión de Jess.

Esta concordancia nos permitió verificar que para este caso la traducción de la base de reglas fue correcta, el nuevo modelo orientado a objetos también y que la inferencia realizada por Drools era equivalente a la de Jess.

La diferencia en la cantidad de instancias que conforman el *estado actual* que se observa en la Tabla 4.1 evidencia en realidad una de las mejoras introducidas por la nueva versión. Es decir, si bien hay menos instancias, pero no hay perdida de información ya que las instancias adicionales eran causadas por la implementación anterior. Uno de los problemas de la versión anterior era justamente que el volumen de instancias en la salida que dificultaba la interpretación de los resultados.

Parte de la salida de los programas se puede ver en los Listados 4.1 y 4.2 donde se puede observar la redundancia mencionada.

Listado 4.1: Parte de la salida producida por MAMAS 2

```
1  boissin.ChefDEntreprise [name=PasBoisFichSe3, remplir=true,
       sregle=OBS, nom=Carle]
2  boissin.CroissanceDeLEntreprise [name=Regle18Boissin2003,
       remplir=true, sregle=CONST100, acteur=boissin.
       ChefDEntreprise [name=PasBoisFichSe3, remplir=true, sregle
       =OBS, nom=Carle]]
3  boissin.ElaborationDeLaStrategie [name=Regle17Boissin2003,
       remplir=true, sregle=CONST100, degreDeCentralisation=<null
       >, acteur=boissin.ChefDEntreprise [name=PasBoisFichSe3,
       remplir=true, sregle=OBS, nom=Carle]]
4
5  boissin.DeveloppementDeLEntreprise [name=PasboisboisSe1,
       sregle=CONST100]
6  boissin.Entreprise [name=BoisPijoSe5, remplir=true, sregle=OBS
       ]
7
8  reyes.AdaptationAuChangement [name=PasreyereyeSe11, sregle=
       CONST80, difficile=<null>]
9
10 reyes.Anticipation [name=Regle54Reyes2004, degre=FAIBLE,
       sregle=CONST80]
11
12 reyes.Dirigeant [name=PasReyeFichSe2, acteur=<null>, remplir=
       true, sregle=OBS]
13
14 reyes.MethodeDeDecision [name=Regle95Reyes2004, acteur=reyes.
       Dirigeant [name=PasReyeFichSe2, acteur=<null>, remplir=
       true, sregle=OBS], sregle=CONST100]
15
16 reyes.Proximite [name=Regle40Reyes2004, sregle=CONST80,
       evolutionFuture=<null>]
```

Listado 4.2: Parte de la salida producida por MAMAS

```
1  (Boissin_2003::chef_d_entreprise (Name PAS_BOIS_FICH_se3) (ID
       19) (remplir TRUE) (Snom "CARLE") (Ssregle obs))
2  (Boissin_2003::croissance_de_l_entreprise (Name
       regle_18_boissin_2003) (ID 128) (remplir TRUE) (Sacteur
       19) (Sevolution_future nil) (Ssregle const100))
3  (Boissin_2003::elaboration_de_la_strategie (Name
```

```
    regle_17_boissin_2003) (ID 129) (remplir TRUE) (Sacteur
    19) (Sdegre_de_centralisation nil) (Ssregle const100))
4   (Boissin_2003::developpement_de_l_entreprise (Name
    pasboisbois_se1) (ID 127) (Ssregle const100))
5   (Boissin_2003::entreprise (Name BOIS_PIJO_se5) (ID 44) (
    remplir TRUE) (Ssregle obs))
6
7   (Reyes_2004::adaptation_au_changement (Name pasreyereye_se11)
    (ID 70) (Sdifficile nil) (Ssregle const80))
8   (Reyes_2004::adaptation_au_changement (Name REYE_PIJO_se7) (ID
     143) (Sdifficile nil) (Ssregle const80))
9
10  (Reyes_2004::anticipation (Name regle_54.1_reyes_2004) (ID 65)
     (Ssregle const80) (Sdegre faible))
11  (Reyes_2004::anticipation (Name regle_54_reyes_2004) (ID 88) (
    Ssregle const80) (Sdegre faible))
12
13  (Reyes_2004::dirigeant (Name PAS_REYE_FICH_se2) (ID 27) (
    Sacteur -1) (Ssregle obs) (remplir TRUE))
14  (Reyes_2004::dirigeant (Name PAS_REYE_BOIS_se7) (ID 36) (
    Sacteur -1) (Ssregle obs) (remplir TRUE))
15  (Reyes_2004::dirigeant (Name PAS_REYE_BOIS_se1) (ID 38) (
    Sacteur -1) (Ssregle obs) (remplir TRUE))
16
17  (Reyes_2004::methode_de_decision (Name regle_95_reyes_2004) (
    ID 107) (Sacteur 27) (Ssregle const100))
18  (Reyes_2004::methode_de_decision (Name regle_95_reyes_2004) (
    ID 116) (Sacteur 36) (Ssregle const100))
19  (Reyes_2004::methode_de_decision (Name regle_95_reyes_2004) (
    ID 124) (Sacteur 38) (Ssregle const100))
20
21  (Reyes_2004::proximite (Name regle_38_reyes_2004) (ID 82) (
    Ssregle const80) (Sevolution_future nil))
22  (Reyes_2004::proximite (Name regle_40_reyes_2004) (ID 68) (
    Ssregle const80) (Sevolution_future nil))
```

En las líneas 7 y 8 del Listado 4.2 se evidencia el problema mencionado. Hay dos instancias de `adaptation_au_changement` que sólo difieren en el *slot* ficticio `ID`, lo mismo ocurre en las líneas 10 y 11 con `anticipation`. El problema se agrava cuando el objeto inferido forma parte de otros (agregación), como en el caso de las líneas 13, 14 y 15, donde aparecen tres instancias de `dirigeant` que terminan generando tres instancias de `methode_de_decision` (notar por ejemplo en la línea 17 el valor `(Sacteur 27)` hace referencia al `ID` del `dirigeant` de la línea 13).

En el Listado 4.1 se puede ver que hay una sola instancia de las clases mencionadas: `AdaptationAuChangement` (línea 7), `Anticipation` (línea 9), `Dirigeant` (línea 11) y `MethodeDeDecision` (línea 13).

El inconveniente ocurre porque la versión anterior del sistema interpreta como dos instancias distintas a aquellas que poseen algún valor distinto en sus *slots* y todas las instancias difieren en el valor del *slot* ficticio `ID` (explicado en la Sección 2.5.2). En la nueva versión se evitó este problema aprovechando la configuración del motor de inferencias (`equalsBehavior="equality"`) explicada en 3.2 y definiendo el método `equals` de las clases de forma que sólo se tengan en

cuenta los atributos relevantes. Esta característica de Drools que brinda mayor control en la comparación de dos objetos resulta clave para evitar la explosión de instancias en la salida.

Por ejemplo las reglas `reyes_38` y `reyes_40` agregan a la memoria de trabajo una instancia de `Proximite` (línea 16 del Listado 4.1). En el caso de Drools al ejecutarse la segunda regla, detecta que ya hay un objeto de este tipo con esos valores en la memoria de trabajo y no lo inserta. En el caso de Jess, no tiene forma de distinguir estas instancias por lo cual aparecen repetidas en la salida (líneas 21 y 22 del Listado 4.2).

En conclusión podemos decir que con MAMAS2 se obtienen resultados superadores ya que para el caso tratado tener un 30 % menos de instancias en la salida representa un avance importante para la posterior interpretación de los resultados.

4.2. PAPAS

Continuando con el ciclo de asesoramiento a la empresa mostrado en la Figura 1.2, el consultor definió unos objetivos apropiados para el caso, es decir definió un *estado final*. De acuerdo a su práctica habitual y en base a su experiencia, definió también una serie de recomendaciones para alcanzar dicho estado, es decir un *estado ideal*. Para comprobar el funcionamiento de PAPAS comparamos este *estado ideal* definido por el consultor con el arrojado por la ejecución.

En esta etapa aprovechamos una característica interesante de Drools que permite configurar sesiones independientes, cada una con su conjunto de reglas. De esta forma la ejecución de las reglas abductivas no interfiere con la ejecución de las reglas de la fase de análisis.

Los objetivos planteados en el *estado final* permitieron instanciar los siguientes conceptos:

- aumentar la delegación
  `reyes.Delegation [evolutionFuture=CROISSANT, sregle=CONST50]`
- el empresario dueño de la empresa debe estar presente
  `boissin.ChefDEntreprise [sregle=OBS]`
- aumentar el crecimiento de la empresa
  `boissin.CroissanceDeLEntreprise [evolutionFuture=CROISSANT, sregle=CONST100, remplir=true, acteur=boissin.ChefDEntreprise [sregle=OBS]]`
- aumentar la velocidad de respuesta
  `reyes.Reactivite [sregle=CONST80]`
- mantener la flexibilidad
  `reyes.Flexibilite [sregle=OBS]`
- formalizar la estrategia
  `reyes.Strategie [elaboration=DEMARCHE_FORMALISEE, sregle=CONST50]`

Por otro lado, la instanciación correspondiente al *estado ideal* recomendado por el consultor es la siguiente:

- el dirigente debe ser el artífice principal de la elaboración de la estrategia
  `boissin.ElaborationDeLaStrategie [acteur=boissin.Dirigeant [sregle=OBS],sregle=OBS]`
- mejorar la comunicación dentro de la empresa, principalmente mediante su formalización
  `reyes.CommunicationFormelle [sregle=CONST50]`
- aumentar los efectivos (cantidad de empleados)
  `reyes.Effectif [sregle=OBS,evolutionFuture=CROISSANT]`
- mantener el compromiso del personal
  `reyes.Implication [sregle=CONST80]`
- mantener la motivación del personal
  `reyes.Motivation [sregle=CONST80]`
- debe consolidarse como una empresa mediana
  `reyes.MoyenneEntreprise [sregle=OBS]`

En el Listado 4.3 se muestra el resultado de la ejecución de PAPAS; las líneas resaltadas (3, 29, 30, 32, 36, 38, 39) muestran las coincidencias entre las recomendaciones inferidas por el sistema y las sugeridas por el consultor. Es decir, que prácticamente todas las recomendaciones hechas por el consultor están presentes en la salida de PAPAS.

Es importante destacar que al analizar en detalle el *estado ideal* definido por el consultor pudimos observar que se puede obtener con un paso de encadenamiento hacia atrás; mientras que el módulo de razonamiento abductivo va más allá y sólo está limitado por la estructura de la base de reglas. En el *estado ideal* producido por PAPAS, hay otras instancias (como por ejemplo `pijouret.SatisfactionAuTravail`), que se obtienen a partir de un encadenamiento hacia atrás de hasta cuatro pasos.

Listado 4.3: Salida producida por PAPAS

```
1  boissin.Brevet [sregle=OBS,evolutionFuture=CROISSANT]
2  boissin.CashFlow [sregle=OBS,evolutionFuture=CROISSANT]
3  boissin.ChefDEntreprise [sregle=OBS]
4  boissin.ChiffreDAffaires [sregle=OBS,evolutionFuture=CROISSANT
       ]
5  boissin.DeveloppementDeLEntreprise [name=Abductive
       PasboisboisSe1,sregle=CONST100]
6  boissin.Emploi [sregle=OBS,evolutionFuture=CROISSANT]
7  boissin.Entreprise [name=Abductive PasReyeBoisSe3,sregle=OBS]
8  boissin.FondsPropres [sregle=OBS,evolutionFuture=CROISSANT]
9  boissin.OutputPhysiqueDeLEntreprise [sregle=OBS,
       evolutionFuture=CROISSANT]
10 boissin.PartDeMarche [sregle=OBS,evolutionFuture=CROISSANT]
11 boissin.Profit [sregle=OBS,evolutionFuture=CROISSANT]
12 boissin.ReseauFormel [sregle=OBS,evolutionFuture=CROISSANT]
13 boissin.SoldeIntermediaireDeGestion [sregle=OBS,
       evolutionFuture=CROISSANT]
14 boissin.ValeurDeLActifNet [sregle=OBS,evolutionFuture=
       CROISSANT]
15 boissin.Vente [sregle=OBS,evolutionFuture=CROISSANT]
```

```
16 fiche.Entreprise [name=Abductive PasReyeFichSe4,sregle=OBS]
17 pijouret.CultureDeLEntreprise [sregle=CONST80]
18 pijouret.Emploi [name=Abductive BoisPijoSe3,sregle=OBS]
19 pijouret.Entreprise [name=Abductive ReyePijoSe5,sregle=OBS]
20 pijouret.Flexibilite [name=Abductive ReyePijoSe9,sregle=OBS]
21 pijouret.Gpec [name=Abductive ReyePijoSe11,sregle=CONST100]
22 pijouret.Implication [name=Abductive ReyePijoSe13,sregle=
       CONST80]
23 pijouret.Motivation [name=Abductive ReyePijoSe15,sregle=
       CONST80]
24 pijouret.SatisfactionAuTravail [sregle=OBS]
25 pijouret.Strategie [name=Abductive ReyePijoSe17,sregle=CONST50
       ]
26 reyes.Actif [name=Abductive PasReyeBoisSe21,sregle=OBS]
27 reyes.AdaptationAuChangement [name=Abductive ReyePijoSe7,
       sregle=OBS]
28 reyes.Capital [name=Abductive PasReyeBoisSe11,sregle=OBS]
29 reyes.CommunicationFormelle [sregle=CONST50]
30 reyes.Effectif [sregle=OBS,evolutionFuture=CROISSANT]
31 reyes.Entreprise [name=Abductive PasreyereyeSe1,sregle=OBS]
32 reyes.Flexibilite [sregle=OBS]
33 reyes.FormeOrganisationnelle [name=Abductive PasreyereyeSe9,
       sregle=OBS]
34 reyes.Gpec [sregle=CONST100]
35 reyes.GrandeEntreprise [name=Abductive PasreyereyeSe3,sregle=
       OBS]
36 reyes.Implication [sregle=CONST80]
37 reyes.ModeDOrganisation [name=Abductive PasreyereyeSe9,sregle=
       OBS]
38 reyes.Motivation [sregle=CONST80]
39 reyes.MoyenneEntreprise [sregle=OBS]
40 reyes.PetiteEntreprise [name=Abductive PasreyereyeSe5,sregle=
       OBS]
41 reyes.StructureFormelle [sregle=CONST100]
42 reyes.TresPetiteEntreprise [name=Abductive PasreyereyeSe7,
       sregle=OBS]
```

Todo esto nos permite concluir que para este caso el sistema funciona correctamente y constituye una herramienta útil para el desarrollo de la actividad del experto. Estos resultados son alentadores e invitan a utilizar el sistema en otros casos para continuar validando los beneficios que aporta.

Capítulo 5

Comentarios finales

5.1. Conclusiones

En este trabajo se avanzó en el desarrollo del proyecto MAEOS. Más precisamente, se agregó un módulo de diagnóstico (PAPAS) que junto con el módulo de análisis (MAMAS) posibilitan la evaluación completa de la PyME. Para lograrlo se reimplementó gran parte del sistema, logrando así una mayor uniformidad conceptual y resolviendo algunos problemas de representación que limitaban la expresividad de la base de conocimientos.

Todo esto permitió cumplir con uno de los objetivos del proyecto: mejorar la eficiencia y rendimiento de las recomendaciones de negocios a PyMEs en los aspectos en los que humano es menos eficiente. Vale la pena recordar que la intención no fue reemplazar la tarea de consultor por un software, sino facilitar el hecho que pueda aplicar su criterio basado tanto en sus conocimientos teóricos como en su experiencia previa para discriminar que resultados tener en cuenta y cuales descartar.

La posibilidad de contar con datos de una empresa permitió comprobar el funcionamiento de las herramientas desarrolladas en un caso real. Además los aportes del consultor permitieron validar los resultados obtenidos que han sido promisorios e invitan a la realización de más pruebas.

También hay que destacar que para el desarrollo de PAPAS se programaron varios componentes de software que si bien se aplicaron a la generación automática de reglas abductivas para las PyMEs; en realidad son independiente del dominio. En particular la gramática para las reglas Drools y los programas que la procesan son lo suficientemente generales para ser reutilizados en otros proyectos.

Relacionado con lo anterior, para llevar a cabo la implementación fue fundamental la interacción con la comunidad de desarrollo de Drools que aportó ideas sobre el camino a seguir. Vale la pena mencionar que durante la investigación realizada como parte de este trabajo, también se realizaron contribuciones al código del proyecto Drools.

5.2. Trabajos futuros

El proyecto MAEOS ha ido creciendo con los años e incorporando distintas herramientas de software: una aplicación web (DONNA) para administrar la base de reglas, una aplicación de escritorio (Protégé-Frames) para administrar la base de conocimientos, otra aplicación de escritorio (DISKO) para ingresar los datos de la PyME y realizar el análisis de los mismos.

Para mantener actualizadas sus bases de conocimientos DISKO debe ejecutar un proceso de importación que accede remotamente a la base de datos MySQL de DONNA y realizar un proceso de traducción del formato relacional al de Jess y Protégé-Frames.

Este trabajo representa un primer paso en la reestructuración del proyecto MAEOS tendiente a lograr una implementación más homogénea. La próxima etapa sería reemplazar DISKO por un sistema web para que acceda directamente a la misma base de datos que DONNA. Esto además facilitaría la tarea del consultor ya que no necesitaría tener la herramienta instalada sino que podría acceder a la misma estando en la empresa desde cualquier dispositivo que disponga de conectividad a internet.

Para esto se puede explorar la suite de jBPM (Business Process Management) que permite modelar, ejecutar y monitorizar las distintas fases del proceso de análisis y diagnóstico. Además puede ser combinado con Drools ya que provee un editor de reglas que podría reemplazar parte de la funcionalidad provista actualmente por DONNA.

Otra posibilidad interesante es la incorporación de razonamiento difuso (*fuzzy*). Actualmente, la mayoría de las clases poseen el atributo *sregle* que indica el nivel de certeza con el que aparece una instancia. Hay un subproyecto llamado *Drools Chance* que agrega al motor de inferencias soporte para el manejo de incertidumbre.

También se puede avanzar en la implementación de la capa de experiencia de la arquitectura presentada en [10]. Durante la evolución de este trabajo se investigó la posibilidad de utilizar Decisional DNA (DDNA) [44] y Set of Experience Knowledge Structure (SOEKS) [43] para almacenar formalmente los eventos de toma de decisión y poder así capitalizar la experiencia del consultor. Al momento de comenzar este trabajo estos *frameworks* estaban en un estado incipiente de su desarrollo, sin embargo lucen como una alternativa prometerdora. Otra opción evaluada para capitalizar el conocimiento del experto fue Case Based Reasoning (CBR) [1] [26], que permitiría generar recomendaciones basadas en la adaptación de experiencias pasadas. El inconveniente de esta propuesta fue que para comprobar su efectividad era necesaria una base con numerosos casos que no estaba disponible en su momento.

Una última dirección de investigación tiene que ver con la reducción automática de la cantidad de instancias obtenidas como resultado de las inferencias. Tanto la inferencia hacia adelante como la inferencia hacia atrás generan gran cantidad de objetos, por lo que sería de utilidad (ya que facilitaría la tarea del consultor) incorporar un proceso de filtrado de los mismos. Esto también ayudaría por ejemplo a una vez obtenido el *estado ideal* compararlo con el *estado actual* para determinar cuales son los valores a ajustar.

Apéndice A

Documentación de las aplicaciones

Para desarrollar las aplicaciones que conforman el proyecto se utilizó el IDE Eclipse. A la instalación básica se le agregaron los *plugins* de JET (para que se generen las clases a partir de los templates), ANTLR (para que se genere el *listener* a partir de la gramática), Drools (para el chequeo automático de la sintaxis de las reglas).

A.1. Organización del código fuente

A continuación se enumera la distribución de los archivos del proyecto.

`/src`

`/main`

`/java`

`/ar.edu.unr.fceia.lcc.abductive`
clases desarrolladas para generar las reglas abductivas.

`/ar.edu.unr.fceia.lcc.es`
clases utilizadas en el Capítulo 4 para testear el caso de la empresa Electricité Services.

`/ar.edu.unr.fceia.lcc.mamas`
clases generadas automáticamente por JET a partir de los templates.

`/ar.edu.unr.fceia.lcc.model`
clases auxiliares utilizadas para almacenar los datos obtenidos al parsear ontologías, reglas, etc.

`/ar.edu.unr.fceia.lcc.model.ontology`
enumeraciones utilizadas por las clases del modelo para hacer referencia a valores constantes.

`/ar.edu.unr.fceia.lcc.model.ontology.(boissin, fiche, pijouret, reyes, torres, triquere)`
cada paquete contiene el modelo de datos de un área particular del dominio. Se mantuvieron los nombres de las ontologías originales.

`/antlr4`
archivo DroolsRules.g4 que contiene la gramática desarrollada pa-

ra parsear las reglas de Drools.

`/resources`

`/rules.goal`
archivo .drl con la regla abductiva genérica.

`/rules.mamas`
archivos .drl con las reglas de análisis.

`/rules.papas`
archivos .drl con las reglas de diagnóstico.

`/target/generated-sources/antlr4`
clases base generadas automáticamente por ANTLR a partir de la gramática. Estas son las que hay que extender para generar las reglas abductivas.

`/fichiers`
archivos de las ontologías y reglas originales (no se utilizan, corresponden a la versión anterior del sistema).

`/preco`
archivos con las reglas de recomendaciones originales (no se utilizan, corresponden a la versión anterior del sistema).

`/templates`
archivos .jet para generar las clases, reglas de MAMAS y de PAPAS.

Bibliografía

[1] Agnar Aamodt y Enric Plaza. «Case-based reasoning; Foundational issues, methodological variations, and system approaches». En: *AI COMMUNICATIONS* 7.1 (1994), págs. 39-59 (vid. pág. 53).

[2] Santiago Almirón. «Gestión de Múltiples Puntos de Vista en Análisis y Diagnóstico de PyMEs». Tesis de lic. Universidad Nacional de Rosario, 2011 (vid. págs. 7, 9, 27, 28, 30, 46).

[3] Douglas E. Appelt y Martha E. Pollack. «Weighted Abduction for Plan Ascription». En: *Technical Note 491, SRI International, Menlo Park*. 1992, págs. 1-25 (vid. pág. 6).

[4] Husam Arman, Allan Hodgson y Nabil NZ Gindy. «An ontology-based knowledge management system to support technology intelligence». En: *International Journal of Industrial and Systems Engineering* 5.3 (2010), págs. 377-389 (vid. pág. 6).

[5] RJ Bennett. «Expectations-based evaluation of SME advice and consultancy: an example of business link services». En: *Journal of small business and enterprise development* 14.3 (2007), págs. 435-457 (vid. pág. 5).

[6] A.A. Betanzos. *Ingeniería del conocimiento: aspectos metodológicos*. Pearson Educación, 2004. ISBN: 9788420541921 (vid. págs. 12, 27).

[7] Stanford Center for Biomedical Informatics Research. *Protege Desktop Older Versions*. 2015. URL: `http://protegewiki.stanford.edu/wiki/Protege_Desktop_Old_Versions` (vid. pág. 27).

[8] Grady Booch. *Object-Oriented Analysis and Design with Applications (3rd Edition)*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 2004. ISBN: 020189551X (vid. pág. 16).

[9] Philippe Bouché, Nathalie Gartiser y Cecilia Zanni-Merk. «Managing Knowledge in the Framework of the Organizational Evolution of SMEs». English. En: *Innovation through Knowledge Transfer 2010*. Ed. por RobertJ. Howlett. Vol. 9. Smart Innovation, Systems and Technologies. Springer Berlin Heidelberg, 2011, págs. 63-72. ISBN: 978-3-642-20507-1 (vid. pág. 7).

[10] Lucas Boullosa, Cecilia Zanni-Merk, Nathalie Gartiser y Ana Casali. «Formalizing Experiential Knowledge for Analysis and Diagnosis of SME». En: *WAMS 2013 - The International Workshop on Applied Modeling and Simulation*. Université dÁix Marseille (France), 2013, págs. 42-48 (vid. págs. 7, 10, 53).

[11] Ronald Brachman y Hector Levesque. *Knowledge Representation and Reasoning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004. ISBN: 1558609326 (vid. pág. 12).

[12] Richard Dazeley. «An Expert System Methodology for SMEs and NPOs». En: *11th Annual Australian Conference on Knowledge Management and Intelligent Decision Support-ACKMIDS*. 2008 (vid. pág. 6).

[13] F. De Carlo, O. Borgia, M. Tucci y M. Rapaccini. «A rule based expert system for maintenance as a competitive advantage». En: *Reliability and Maintainability Symposium, 2009. RAMS 2009. Annual*. Ene. de 2009, págs. 448-453 (vid. pág. 6).

[14] Sylvain Delisle y Josée St-Pierre. «Two Expert Diagnosis Systems for SMEs: From Database-Only Technologies to the Unavoidable Addition of AI Techniques». English. En: *Knowledge-Based Intelligent Information and Engineering Systems*. Ed. por Vasile Palade, RobertJ. Howlett y Lakhmi Jain. Vol. 2773. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2003, págs. 111-125. ISBN: 978-3-540-40803-1 (vid. pág. 6).

[15] Marc Denecker y Antonis Kakas. «Abduction in Logic Programming». English. En: *Computational Logic: Logic Programming and Beyond*. Ed. por AntonisC. Kakas y Fariba Sadri. Vol. 2407. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, págs. 402-436. ISBN: 978-3-540-43959-2 (vid. pág. 6).

[16] Igor Douven. «Abduction». En: *The Stanford Encyclopedia of Philosophy*. Ed. por Edward N. Zalta. Spring 2011. 2011 (vid. pág. 20).

[17] Ernest Friedman-Hill. *Jess in Action: Java Rule-based Systems*. Greenwich, CT: Manning, 2003. ISBN: 1-930-11089-8 (vid. pág. 22).

[18] José A Gámez. «Abducción en modelos gráficos». En: (1998) (vid. pág. 20).

[19] Joseph C. Giarratano y Gary D. Riley. *Expert Systems: Principles and Programming*. Pacific Grove, CA, USA: Brooks/Cole Publishing Co., 2005. ISBN: 0534384471 (vid. págs. 11, 13, 18, 20).

[20] Thomas R. Gruber. «A translation approach to portable ontology specifications». En: *KNOWLEDGE ACQUISITION* 5 (1993), págs. 199-220 (vid. pág. 18).

[21] Thomas R. Gruber. «Toward Principles for the Design of Ontologies Used for Knowledge Sharing». En: *IN FORMAL ONTOLOGY IN CONCEPTUAL ANALYSIS AND KNOWLEDGE REPRESENTATION, KLUWER ACADEMIC PUBLISHERS, IN PRESS. SUBSTANTIAL REVISION OF PAPER PRESENTED AT THE INTERNATIONAL WORKSHOP ON FORMAL ONTOLOGY*. Kluwer Academic Publishers, 1993 (vid. pág. 18).

[22] Nicola Guarino. «Formal Ontology and Information Systems». En: IOS Press, 1998, págs. 3-15 (vid. pág. 18).

[23] Kouichi Hirata. «A classification of abduction: abduction for logic programming.» En: *Machine intelligence 14*. Citeseer. 1993, pág. 405 (vid. pág. 6).

[24] J.R. Josephson, B. Chandrasekaran, Jr. Smith J.W. y M.C. Tanner. «A Mechanism for Forming Composite Explanatory Hypotheses». En: *Systems, Man and Cybernetics, IEEE Transactions on* 17.3 (mayo de 1987), págs. 445-454. ISSN: 0018-9472 (vid. pág. 6).

[25] Gregorio Klimovsky. *Las Desventuras del conocimiento científico: una introducción a la epistemología*. La Ciencia y la gente. A-Z Editora, 1994. ISBN: 9789505342754 (vid. pág. 18).

[26] Janet L. Kolodner. «An introduction to case-based reasoning». English. En: *Artificial Intelligence Review* 6.1 (1992), págs. 3-34. ISSN: 0269-2821 (vid. pág. 53).

[27] David B Leake. «Focusing construction and selection of abductive hypotheses». En: *IJCAI*. Vol. 93. 1993, págs. 24-29 (vid. pág. 6).

[28] Henry. Mintzberg. *Mintzberg on Management: Inside Our Strange World of Organizations*. English. Free Press, 1989. ISBN: 9780029213711 (vid. pág. 5).

[29] Henry Mintzberg. *The structuring of organizations: A synthesis of the research*. English. Englewood Cliffs, NJ: Prentice-Hall, 1979, xvi, 512 p. : ISBN: 0138552703 (vid. pág. 5).

[30] Kevin Mole. «Tacit knowledge, heuristics, consistency and error signals : how do business advisers diagnose their SME clients?» En: *Journal of Small Business and Enterprise Development* 14.4 (2007), págs. 582-601 (vid. pág. 5).

[31] Charles G. Morgan. «Hypothesis generation by machine». En: *Artificial Intelligence* 2.2 (1971), págs. 179 -187. ISSN: 0004-3702 (vid. pág. 6).

[32] Natalya F. Noy y Deborah L. Mcguinness. *Ontology Development 101: A Guide to Creating Your First Ontology*. Inf. téc. 2001 (vid. pág. 18).

[33] Natalya Fridman Noy, Ray W. Fergerson y Mark A. Musen. «The Knowledge Model of ProtéGé-2000: Combining Interoperability and Flexibility». En: *Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management*. EKAW '00. London, UK, UK: Springer-Verlag, 2000, págs. 17-32. ISBN: 3-540-41119-4 (vid. pág. 16).

[34] Doris Gomezelj Omerzel y Bostjan Antoncic. «Critical entrepreneur knowledge dimensions for the SME performance». En: *Industrial Management & Data Systems* 108.9 (2008), págs. 1182-1199. eprint: `http://dx.doi.org/10.1108/02635570810914883` (vid. pág. 5).

[35] P.V. O'Rorke, American Association for Artificial Intelligence y United States. Office of Naval Research. *Automated Abduction: Working Notes, 1990 Spring Symposium Series, March 27-29, 1990, Stanford University*. Technical report (University of California, Irvine. Department of Information and Computer Science). Information y Computer Science, University of California, Irvine, 1990 (vid. pág. 6).

[36] Terence Parr. *The Definitive ANTLR 4 Reference*. 2nd. Pragmatic Bookshelf, 2013. ISBN: 1934356999, 9781934356999 (vid. pág. 40).

[37] Yun Peng y James A. Reggia. *Abductive Inference Models for Diagnostic Problem-solving*. New York, NY, USA: Springer-Verlag New York, Inc., 1990. ISBN: 0-387-97343-5 (vid. pág. 6).

[38] David Poole. «A methodology for using a default and abductive reasoning system». En: *International Journal of Intelligent Systems* 5.5 (1990), págs. 521-548. ISSN: 1098-111X (vid. pág. 6).

[39] Harry E. Pople. «On the Mechanization of Abductive Logic». En: *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*. IJCAI'73. Stanford, USA: Morgan Kaufmann Publishers Inc., 1973, págs. 147-152 (vid. pág. 6).

[40] Dominique Renaud, Philippe Bouché, Nathalie Gartiser, Cecilia Zanni-Merk y Henri-Pierre Michaud. «Knowledge Transfer for Supporting the Organizational Evolution of SMEs: A Case Study». English. En: *Innovation through Knowledge Transfer*. Ed. por RobertJames Howlett. Vol. 5. Smart Innovation, Systems and Technologies. Springer Berlin Heidelberg, 2010, págs. 293-302. ISBN: 978-3-642-14593-3 (vid. pág. 5).

[41] PaulJ.A. Robson y RobertJ. Bennett. «SME Growth: The Relationship with Business Advice and External Collaboration». English. En: *Small Business Economics* 15.3 (2000), págs. 193-208. ISSN: 0921-898X (vid. pág. 5).

[42] Stuart J. Russell y Peter Norvig. *Artificial Intelligence: A Modern Approach*. 2.ª ed. Pearson Education, 2003. ISBN: 0137903952 (vid. pág. 21).

[43] Cesar Sanín y Edward Szczerbicki. «Experience-Based Knowledge Representation: Soeks». En: *Cybernetics and Systems* 40.2 (2009), págs. 99-122 (vid. pág. 53).

[44] Cesar Sanín, Carlos Toro, Haoxi Zhang, Eider Sanchez, Edward Szczerbicki, Eduardo Carrasco, Peng Wang y Leonardo Mancilla-Amaya. «Decisional DNA: A multi-technology shareable knowledge structure for decisional experience». En: *Neurocomputing* 88 (2012), págs. 42-53 (vid. pág. 53).

[45] Richard Stallman. *¿Qué es el software libre?* 2015. URL: `http://www.gnu.org/philosophy/free-sw.es.html` (vid. pág. 29).

[46] The JBoss Drools team. *Drools Documentation*. 2015. URL: `http://docs.jboss.org/drools/release/6.2.0.Final/drools-docs/html_single/index.html` (vid. pág. 25).

[47] Hai H. Wang y col. *Frames and OWL Side by Side*. Inf. téc. 2006 (vid. pág. 27).