Universidad Nacional del Sur

Tesis de Doctor en Ciencias de la Computación

Grupos de Servicios de Razonamiento para el Procesamiento de Consultas Contextuales en Paralelo

Mariano Tucat

Bahía Blanca Argentina

2011

Prefacio

Esta Tesis se presenta como parte de los requisitos para optar al grado Académico de Doctor en Ciencias de la Computación, de la Universidad Nacional del Sur y no ha sido presentada previamente para la obtención de otro título en esta Universidad u otra. La misma contiene los resultados obtenidos en investigaciones llevadas a cabo en el ámbito del Departamento de Ciencias e Ingeniería de la Computación durante el período comprendido entre el 1 de abril de 2005 y el 15 de julio de 2011, bajo la dirección del Dr. Alejandro J. García, Profesor Asociado del Departamento de Ciencias e Ingeniería de la Computación.

.....

Mariano Tucat
mt@cs.uns.edu.ar
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Bahía Blanca, 15 de julio de 2011

UNIVERSIDAD NACIONAL DEL SUR
Secretaría General de Posgrado y Educación Continua

La presente tesis ha sido aprobada el .../.../..., mereciendo la calificación de .....(.....)

Agradecimientos

He tenido mucha suerte.

Sobre el final de mis estudios de grado, tuve la suerte de encontrarme con el Dr. Alejandro García, quien me sugirió y me ayudó a que iniciara los estudios de posgrado. Gracias, Ale, una gran parte de lo que aprendí durante estos años, no sólo relacionado con la investigación, sino también con la docencia, lo aprendí gracias a vos.

También tuve (y tengo) la suerte de ser parte del LIDIA y del DCIC, lo cual facilitó el desarrollo de mis estudios, gracias a todos los que forman el laboratorio y también, a los que son parte del departamento. Este agradecimiento es extensivo al CONICET, organismo que me dio apoyo económico e institucional.

A lo largo de estos años en la universidad, tuve la suerte de conocer a muchas personas, no sólo durante el desarrollo de esta tesis, como el grupo de la salita, sino también, a algunos de ellos los conocí durante la carrera de grado, como Nico y Gotti. Gracias, amigos, por tantas charlas y por compartir tan buenos momentos.

Sin lugar a dudas, nada de esto hubiera sido posible sin la ayuda de mis padres, Ruben y Graciela, y mi hermano, Seba. En este caso no creo que haya sido suerte, sino el esfuerzo de ellos, que me permitieron estudiar sin tener que preocuparme por otra cosa. Gracias, viejos, por enseñarme con el ejemplo lo que es la responsabilidad, el sacrificio, entre otras tantas cosas.

Tuve, también, la suerte de conocer, en estos años en la uni, a quien hoy es mi compañera, Juli. Gracias, Juli, por ayudarme, por escucharme, por alentarme, por estar, siempre. Te Amo.

Gracias, a todos, Tucky.

Resumen

Esta tesis se enfoca en el desarrollo de una clase particular de *Servicios de Razonamiento* para entornos con múltiples agentes, y de *Grupos de Servicios de Razonamiento* los cuales permiten el tratamiento de diferentes tipos de consultas en paralelo. Se introduce el concepto de *Consulta Contextual*, que permite a los clientes incluir en sus consultas conocimiento privado y especificar qué tratamiento debe darse a dicho conocimiento, al momento de integrarse con el conocimiento público almacenado en el Servicio de Razonamiento. Se abordan tanto la definición abstracta, análisis y propiedades de estos conceptos, como así también se muestran aplicaciones particulares en dominios de aplicación concretos con diferentes lenguajes de representación de conocimiento.

Los Servicios de Razonamiento introducidos en esta tesis permiten representar información o conocimiento público y responder consultas de clientes utilizando este conocimiento público junto con información privada provista por el cliente, la cuál es denominada *contexto*. Las Consultas Contextuales constarán de dos elementos: la consulta particular que el cliente pretende hacer al Servicio de Razonamiento, y un *contexto* dentro del cual debe procesarse dicha consulta particular.

Se definen diferentes tipos de Consultas Contextuales, mostrando las propiedades y relaciones entre ellas. El objetivo de definir estas nuevas alternativas es lograr eficiencia en el cálculo de las respuestas, buscando reducir la sobrecarga en el intercambio de mensajes y las actualizaciones del conocimiento en el Servicio de Razonamiento; y además permitir el procesamiento paralelo de Consultas Contextuales.

Se introduce el concepto de *Grupo de Razonamiento*, que generaliza el concepto de Servicio de Razonamiento y permite: simplificar la interacción de un agente frente a un grupo de Servicios de Razonamiento y cuando sea posible, distribuir automáticamente el cómputo de la respuesta entre los Servicios de Razonamiento del grupo y calcular la respuesta en paralelo en forma automática.

Abstract

This thesis presents a particular kind of *Reasoning Services* for multiagent environments, and *Groups of Reasoning Services* allowing the parallel processing of different kinds of queries. The concept of *Contextual Query* allows the clients to include in their queries, private knowledge and the way in which this information should be handled by the Reasoning Service at the moment of using it with its public knowledge. This theses presents an abstract definition of these concepts, including their analysis and properties, and also specific examples on different application domains using distinct languages for knowledge representation.

The Reasoning Services introduced in this thesis allow the representation of information (public knowledge) and answer client queries using this information together with the private knowledge provided by the clients, also known as *context*. A Contextual Query contains the query, that a client want to ask to the Reasoning Service, and the *context* in which this query should be processed.

Different kinds of Contextual Queries are presented, including their properties and the relationships among them. The main goal of these Contextual Queries is to be efficient when answering them, trying to reduce the number of messages exchanged and the temporal updates of the knowledge at the Reasoning Service; also allowing parallel processing of these Contextual Queries.

A *Group of Reasoning Services* generalizes the concept of Reasoning Service, simplifying the interaction of an agent with a group of Reasoning Services, and allowing the automatic distribution of the queries among Reasoning Services and also their processing in parallel, whenever this is possible.

Índice general

1. Introducción **1**

1.1. Contribuciones 5

1.2. Publicaciones 6

1.3. Organización de la tesis 7

2. Servicio de Razonamiento con Consultas Contextuales **9**

2.1. Servicio de Razonamiento 10

2.2. Consultas Contextuales 16

2.3. Ejemplos de Aplicación 20

2.3.1. Servicio de Recomendación de Noticias 20

2.3.2. Servicio de Razonamiento DeLP 28

2.3.3. Servicio DeLP de Recomendación de Inmuebles 36

2.4. Conclusiones 40

3. Consultas Contextuales Extendidas: Eficiencia y Distribución de carga **42**

3.1. Configuraciones con Múltiples Clientes y *SRs* 43

3.2. Consulta Contextual Múltiple 45

3.3. Consulta Contextual Múltiple Factorizada 48

3.4. Interrogación Contextual 51

3.5. Análisis y Propiedades 56

3.6. Ejemplo de Aplicación de Consultas Extendidas: Servicio de Recomendación de Noticias 59
3.7. Consultas Contextuales Extendidas en DeLP 62
3.8. Conclusiones 65

4. Grupos de Razonamiento **67**

4.1. Grupos de *SR* y Función de Selección 68
4.2. Grupos y Consultas Contextuales Extendidas: Oportunidad de Paralelismo 73
4.2.1. Selección Automática de *SR* 74
4.2.2. Descomposición y Selección Automática 74
4.2.3. Reagrupación y Selección Automática 76
4.3. Opciones para Descomposición y Distribución 81
4.3.1. Descomposición de Consultas Contextuales Extendidas 82
4.3.2. Distribución de Consultas Contextuales 84
4.4. Grupos de Grupos 85
4.5. Conclusiones 86

5. Una Aplicación para *GR*: Intérprete DeLP Paralelo **87**

5.1. Motivación 88
5.2. Servicio de Razonamiento Lógico Argumentativo 90
5.3. Consultas Contextuales para DeLP en paralelo 95
5.4. Intérprete DeLP Paralelo 99
5.5. Algoritmo DeLP Paralelo 103
5.6. Análisis del Algoritmo 105
5.7. Conclusiones 106

6. Propuesta de Implementación **107**

6.1. Motivación 108

6.2. Implementación de *SRs* utilizando Agentes 109

6.2.1. Arquitectura de Agente para Implementar un *SR* 110

6.2.2. Implementación utilizando primitivas de mensajes 115

6.3. Sistemas Multi-Agente para Implementar *GRs* 117

6.3.1. Coordinación entre Agentes 118

6.3.2. Implementación de *SMAs* 122

6.4. Conclusiones 125

7. Conclusiones **127**

A. Programación Lógica **131**

A.1. Sintaxis 132

A.2. Semántica 135

B. Programación Lógica Rebatible (DeLP) **139**

B.1. Sintaxis 140

B.2. Argumentación Rebatible 141

Capítulo 1

Introducción

Esta tesis se enfoca en el desarrollo de una clase particular de *Servicios de Razonamiento* para entornos con múltiples agentes, y de *Grupos de Servicios de Razonamiento* los cuales permiten el tratamiento de diferentes tipos de consultas en paralelo. En la tesis se introduce el concepto de *Consulta Contextual*, que permite a los clientes incluir en sus consultas conocimiento privado y especificar qué tratamiento debe darse a dicho conocimiento, al momento de integrarse con el conocimiento público almacenado en el Servicio de Razonamiento. En la tesis se abordarán tanto la definición abstracta, análisis y propiedades de estos conceptos, como así también se mostrarán aplicaciones particulares en dominios de aplicación concretos con diferentes lenguajes de representación de conocimiento.

Por ejemplo, un Servicio de Razonamiento podría implementar un Servicio de Recomendación de Noticias[1], y este servicio podría ser consultado por diferentes clientes para obtener sugerencias sobre qué publicaciones leer en la Web. Está claro que una consulta como *"¿Qué noticia me recomienda?"* puede ser demasiado general para que el servicio genere una respuesta satisfactoria para el cliente. En cambio, *"Estoy interesado en la Copa América 2011, preferentemente publicaciones deportivas, ¿qué noticia me recomienda?"* provee, además de la consulta, un "contexto", el cual puede ser utilizado por el servicio para realizar una sugerencia que sea satisfactoria.

Los Servicios de Razonamiento (SR) introducidos en esta tesis constarán de tres componentes: conocimiento público, un mecanismo de inferencia, y un conjunto de operadores

[1]Un servicio particular de este dominio de aplicación será desarrollado en el capítulo 2

que definen cómo integrar el conocimiento enviado por los clientes. Las Consultas Contextuales (CC) constarán de dos elementos: la consulta particular que el cliente pretende hacer al SR, y un *contexto* dentro del cual debe procesarse dicha consulta particular. Este contexto podrá contener conocimiento privado incluido por el cliente para ser utilizado en el cálculo de la respuesta, y operadores (conocidos por el SR) a través de los cuales se especificará qué tratamiento darle a dicho conocimiento privado. Por ejemplo, un SR puede disponer de un operador para dar prioridad al conocimiento privado sobre el público, otro operador para excluir conocimiento público, o un operador para dar prioridad al conocimiento público. De esta manera, utilizando los operadores que son indicados en el contexto, el SR podrá integrar adecuadamente el conocimiento privado enviado por el cliente al conocimiento público; y podrá calcular la respuesta, utilizando tanto el conocimiento público almacenado, como el contexto en el cual el cliente quiere que se trate su consulta. Como se mostrará más adelante, una vez generada la respuesta a una CC el contexto utilizado no afectará a futuras consultas del mismo agente, o a otras consultas simultáneas de otros agentes. De esta manera, la misma consulta pero con diferente contexto podrá tener una respuesta diferente.

Siguiendo con el ejemplo introducido anteriormente, si el Servicio de Recomendación de Noticias tiene operadores para preferir o excluir publicaciones específicas, el cliente puede, en el contexto, especificar cómo quiere que se trate el conocimiento particular enviado. Por ejemplo, *"Estoy interesado en la Copa América 2011, preferentemente publicaciones deportivas de Argentina. Excluir las de la editorial Atlantis. ¿Qué noticia me recomienda?"*. En un dominio de diagnóstico, un cliente podría enviar en el contexto de la consulta los síntomas que quiere que se tengan en cuenta para procesar su consulta.

Las Consultas Contextuales y los Servicios de Razonamiento se definirán en forma abstracta, asumiendo un *marco de representación de conocimiento* que incluya un lenguaje de representación que permita realizar inferencias, y un lenguaje para representación de consultas y respuestas acorde con el lenguaje de representación de conocimiento elegido. De esta manera, estos conceptos pueden aplicarse a una gran variedad de lenguajes de representación. A lo largo de la tesis se mostrarán ejemplos de aplicación donde se utilizarán formalismos concretos para representar conocimiento, consultas y respuestas. Por ejemplo, en la Sección 2.3 se mostrará cómo implementar un Servicio de Recomendación de Noticias utilizando Programación Lógica, y otro SR que utiliza Lógica Rebatible para representar conocimiento, y Argumentación como mecanismo de inferencia.

Los *SRs* y las *CCs* introducen una manera particular de realizar una interacción entre agentes en entornos de múltiples agentes. En aquellos escenarios donde existan gran cantidad de clientes y de *SRs*, y además el nivel de interacción entre ellos sea alto, generalmente se requerirá del intercambio de grandes cantidades de información. Por este motivo, en esta tesis se estudiarán y definirán diferentes tipos de consultas contextuales, mostrando las propiedades y relaciones entre ellas. El objetivo de definir estas nuevas alternativas es lograr eficiencia en el cálculo de las respuestas, buscando reducir la sobrecarga en el intercambio de mensajes y las actualizaciones del conocimiento en el *SR*; y además permitir el procesamiento paralelo de consultas contextuales. Por ejemplo, entre las *CCs* que se introducirán, se encuentra la *consulta contextual múltiple* que permite enviar en un solo mensaje varias consultas individuales y recibir las respuestas a cada una de ellas en un único mensaje. Como se verá, esto permite disminuir la sobrecarga sin afectar la semántica de las *CCs* y ofrecerá una oportunidad de paralelismo.

Siguiendo con el ejemplo introducido previamente, el cliente puede estar interesado en obtener noticias relacionadas a temas específicos dentro de un tópico general. En particular, puede ocurrir que el contexto de las distintas noticias en las cuales tiene interés varíe levemente. Por ejemplo, *"Estoy interesado en la Copa América 2011, preferentemente publicaciones deportivas relacionadas al Seleccionado Argentino, ¿qué noticia me recomienda? Además, estoy interesado en obtener información de los distintos estadios en los cuales se jugarán los partidos de la Copa, ¿qué noticia me recomienda?"*. Este tipo de consultas múltiples ofrecerá la oportunidad de reducir la sobrecarga de comunicación y realizar cómputo en paralelo.

En un entorno donde existan varios *SRs* disponibles para ser consultados, los clientes deberían decidir a cuales *SRs* enviar su consulta, lo cual puede resultar en una sobrecarga adicional e innecesaria. Para facilitar la interacción con los clientes y tener más oportunidades de paralelismo en forma automática, en la tesis se introduce el concepto de *Grupo de Razonamiento* (*GR*), que generaliza el concepto de *SR* y permite: (a) simplificar la interacción de un agente frente a un grupo de *SRs*, (b) distribuir automáticamente el cómputo de la respuesta entre los *SRs* del grupo cuando sea posible, y (c) calcular la respuesta en paralelo en forma automática, cuando esto sea posible. Desde el punto de vista de la interacción, el cliente no notará la diferencia entre consultar a un *SR* o a un *GR*.

Continuando con el ejemplo mencionado antes, considere un *GR* formado por múltiples

Servicios de Recomendación de Noticias cada uno de ellos especializado en distintos temas. Por ejemplo, podría existir un *SR* especializado en política, otro en noticias policiales, otro en noticias de selecciones nacionales de fútbol, y otro especializado en noticias de la Copa América 2011. Un cliente puede entonces realizar su consulta directamente al *GR* que agrupa estos servicios especializados. Al momento de recibir la consulta el *GR* puede utilizar el contexto para decidir los *SRs* más adecuados, y además distribuir el cómputo entre los elegidos.

Disponer del concepto Grupo de Razonamiento permite definir *GRs* heterogéneos, donde se puede utilizar el contexto enviado en las consultas para elegir al *SR* más adecuado para calcular la respuesta. También, permite definir *GRs* homogéneos, donde utilizando un grupo de *SRs* con las mismas características se podrá distribuir o paralelizar el cómputo de los diferentes tipos de consultas contextuales definidas.

Como ejemplo de aplicación de un *GR* homogéneo, se desarrolló un intérprete Paralelo para la Programación Lógica Rebatible (DeLP por sus siglas en Inglés). El intérprete DeLP Paralelo utiliza un *GR* formado por múltiples *SRs* específicos con la capacidad de responder a las distintas *CCs*. La manera de consultar a estos *SRs* específicos utilizando las *CCs* permite al *GR* distribuir y paralelizar automáticamente el cómputo del estado de garantía de consultas DeLP.

Como se mostrará en los capítulos siguientes, no hay una única manera de implementar *GRs* y *SRs*. En esta tesis se propondrá una alternativa general de implementación utilizando agentes y sistemas multiagente. La principal ventaja de esta propuesta de implementación corresponde a la flexibilidad en cuanto a su concreción respecto de los dominios de aplicación y los marcos de representación de conocimiento posibles.

Finalmente es importante indicar que en la literatura del área de Inteligencia Artificial es posible encontrar diferentes nociones de contexto relacionados con el razonamiento [BBM95, BM93, Giu93, Guh91]. Por ejemplo, en [Giu93] se expresa que el razonamiento es normalmente llevado a cabo sobre un subconjunto de la base de conocimiento global. Es decir, nunca es considerado todo el conocimiento, sino solo un pequeño subconjunto de él. La noción de contexto es utilizada como una manera de formalizar estas ideas de localización.

Esta tesis utilizará como significado de la palabra *contexto* al conocimiento contextual definido en [Bre05], estando asociado a aquel conocimiento de los clientes que ellos consideren necesario en la resolución de sus consultas. En una visión más pragmática, un

contexto corresponde a un conjunto de hechos utilizados localmente para probar una determinada consulta junto con una parte de las reglas de inferencia utilizadas para razonar acerca del conocimiento.

1.1. Contribuciones

En primer lugar, se define formalmente el concepto de Servicio de Razonamiento (SR). Un SR corresponde a una entidad autónoma con la capacidad de representar conocimiento, el cual será utilizado para responder consultas de clientes. Estos *SRs* permiten a los clientes modificar temporalmente este conocimiento con el objetivo de considerar información específica de los clientes al momento de resolver sus consultas.

Se introduce también el concepto de Consulta Contextual (CC). Una CC corresponde a la herramienta de interacción que poseen los clientes para consultar a un SR. La característica más importante de las *CCs* es que incluyen, además de la consulta propiamente dicha, información privada que los clientes desean sea utilizada en el cómputo de la respuesta y operadores para integrar esa información adecuadamente. Esta información es denominada contexto y tiene como objetivo representar qué información debe tener en cuenta el SR para la consulta y qué tratamiento darle a dicha información.

La CC corresponde a un esquema de interacción entre un cliente y un SR. Como parte del trabajo realizado para el desarrollo de esta tesis se estudiaron y analizaron diferentes formas de interacción. Se definieron varios tipos de *CCs* con el objetivo de abarcar los escenarios que se presentan con mayor frecuencia. Estos escenarios contemplan los requerimientos de los clientes, como por ejemplo, la necesidad de ejecutar múltiples consultas simultáneamente, la posibilidad de realizar diversas consultas utilizando un mismo contexto, o incluso, la alternativa de construir gradualmente el contexto para la consulta, permitiendo intercalar consultas en dicho proceso. En la tesis se presentan relaciones y propiedades entre los diferentes tipos de consultas definidos.

Se introduce el concepto de Grupo de Razonamiento (GR) como una generalización de los *SRs*. Un GR está compuesto por múltiples *SRs* y permite a los clientes abstraerse del problema de decisión respecto de qué SR debe resolver la consulta. La responsabilidad de la decisión recae sobre el GR, el cual deberá determinar cómo resolver cada CC a partir de los *SRs* disponibles. Este concepto permite paralelizar el cómputo de la respuesta a una

consulta en multiples *SRs* de manera transparente para el cliente. Para ejemplificar los conceptos introducidos se incluyen varios ejemplos de aplicación en diferentes dominios. En particular se realizó una aplicación utilizando un *GR* Homogéneo que permite implementar un intérprete de la Programación en Lógica Rebatible que computa los argumentos en paralelo.

1.2. Publicaciones

A continuación se incluyen los artículos publicados como resultado de los trabajos llevados a cabo en la realización de esta tesis. Para cada uno de estos trabajos se detalla brevemente su contribución y se indica el capítulo en el cual fueron incluidos los resultados de los mismos.

- En el trabajo "*An argumentative reasoning service for deliberative agents*" publicado en *Lecture Notes in Artificial Intelligence v. 4798* , [GRTS07], se introduce el concepto de Servicio de Razonamiento y consulta contextual simple. Se realiza una aplicación para un formalismo argumentativo para el cual se definieron tres operadores de integración de conocimiento. Los resultados de este artículo se incluyen en el Capítulo 2.

- En el trabajo "*Using Defeasible Logic Programming with Contextual Queries for Developing Recommender Servers*", publicado en *the Uses of Computational Argumentation, AAAI Fall Symposium 2009* [TGS09], se introducen por primera vez las consultas contextuales múltiples para Servicios de Razonamiento basados en DeLP. Los resultados de este artículo se incluyen en el Capítulo 3.

- En los trabajos "*Interaction Primitives for Implementing Multi-agent Systems*" [GTS05], publicado en *the VII Argentine Symposium on Artificial Intelligence (ASAI 2005)* y "*An Extended Set of Interaction Primitives for Multi-agent Systems Develpment*" [TG05], publicado en el *7mo Workshop de Investigadores en Ciencias de la Computación (WICC 2005)*, se propone un conjunto de primitivas de interacción que incluyen, como características principales, la posibilidad de crear distintos sistemas multi-agente independientes y, por otro lado, la posibilidad de asociar a la recepción de mensajes específicos la ejecución de predicados Prolog. Los resultados de este trabajo se incluyen en el Capítulo 6.

- En el trabajo "*Design and Implementation of a FIPA based Agent Communication Model for a Logic Programming Framework*" [TG07], publicado en el *XIII Congreso Argentino de Ciencias de la Computación (CACiC 2007)*, se presenta un modelo de comunicación entre agentes basado en la arquitectura FIPA y en los Protocolos de Interacción, siguiendo el espíritu de la Programación Lógica. Los resultados de este trabajo se incluyen en el Capítulo 6.

1.3. Organización de la tesis

Esta tesis está organizada de la siguiente manera:

Capítulo 2: Se definen los *Servicios de Razonamiento* y las *Consultas Contextuales*. En principio de introducen los componentes que forman un Servicio de Razonamiento. Luego, se presentan las Consultas Contextuales, el mecanismo que permite a los clientes consultar a estos Servicios de Razonamiento. Se muestran, además, algunas carencias y oportunidades en la interacción cliente-servicio de razonamiento.

Capítulo 3: Se introducen las *Consultas Contextuales Extendidas*, las cuales permiten mejorar la eficiencia en la interacción y facilitan la distribución del cómputo de las respuestas. Se presentan propiedades que permiten relacionar los distintos tipos de consultas definidos y entender sus diferencias y ventajas comparativas.

Capítulo 4: Se define el concepto de *Grupo de Razonamiento*, cuyo principal objetivo es permitir distribuir el cómputo de la respuesta. Además, brinda la posibilidad de seleccionar el servicio de razonamiento más indicado para responder la consulta, facilitando, además, la resolución de la misma en paralelo.

Capítulo 5: Se presenta una *aplicación concreta* de propósito general donde se combinan los conceptos introducidos en los capítulos anteriores. Se muestra como es posible lograr un *intérprete de DeLP Paralelo* utilizando los distintos tipos de consultas contextuales y un grupo de razonamiento formado por servicios de razonamiento especialmente diseñados para la aplicación.

Capítulo 6: Se describe una de las alternativas existentes a la hora de *implementar* los conceptos de servicio de razonamiento y grupo de razonamiento. La alternativa

mostrada utiliza *agentes* para implementar servicios de razonamiento, *sistemas multiagente* para los grupos y *protocolos de interacción* para describir las interacciones entre clientes y servicios de razonamiento.

Capítulo 7: Finalmente se detallan las conclusiones obtenidas.

Apéndice A: Se incluyen las principales definiciones del formalismo de la Programación Lógica, utilizado en los ejemplos introducidos a lo largo de la tesis. Se muestra cómo es la sintaxis de los programas lógicos válidos y se definen las consultas que se pueden realizar sobre estos programas lógicos, junto con su significado. Por último, se introduce la semántica utilizada para los programas lógicos y las derivaciones que se pueden obtener a partir de estos, para finalmente definir cómo es una respuesta válida para una consulta lógica determinada.

Apéndice B: Se incluyen las principales definiciones de la Programación Lógica Rebatible (DeLP por su sigla en inglés). Se definen formalmente los conceptos de programa lógico rebatible, consulta lógica rebatible y respuesta para una consulta lógica rebatible. Luego, se introducen las derivaciones rebatibles, las cuales permitirán realizar una análisis dialéctico, siendo éste la base para la definición de la respuesta a una consulta lógica rebatible.

Capítulo 2

Servicio de Razonamiento con Consultas Contextuales

En el presente capítulo se introducirán los conceptos de *Servicio de Razonamiento* (*SR*) y *Consulta Contextual* (*CC*). Un *SR* puede contener conocimiento público, el cuál será empleado para responder consultas de clientes. La *CC* será el mecanismo que utilizarán los clientes para consultar un *SR*. Este tipo de consulta no sólo incluirá la consulta que el cliente pretende hacer al *SR*, sino que agregará un *contexto* dentro del cual debe procesarse la consulta. Dicho contexto corresponderá al conocimiento privado incluido por el cliente para ser utilizado en el cálculo de la respuesta y operadores que especifiquen qué tratamiento darle a dicho conocimiento privado. Un *SR* utilizará, en el cálculo de las respuestas a las consultas recibidas, el conocimiento público almacenado junto con el conocimiento individual que los clientes pueden incluir en la *CC*.

Como se explicará en este capítulo, al recibir una *CC*, el *SR* modificará temporalmente el conocimiento que tiene almacenado, utilizando los operadores indicados por el cliente en el contexto de la consulta. Esta modificación será temporaria y solo tendrá efecto a fin de calcular la respuesta a la consulta efectuada. Las modificaciones que se realicen para responder una consulta no tendrán efecto para consultas posteriores o simultáneas de otros agentes. Los conceptos de *SR* y *CC* serán definidos en forma abstracta. Esto permitirá que estos conceptos puedan utilizarse con un amplia gama de formalismos, incluyendo a la mayoría de las variantes de la programación lógica (ASP [Bar03, MR01, FL05], DyLP [ALP+94], disjunctive LP [LMR92], etc), Description Logics [BCM+03] y lenguajes de bases de datos deductivas [Liu99], entre otros.

A lo largo de este capítulo y de toda la tesis se utilizará el formalismo de la Programación Lógica para ejemplificar los conceptos introducidos. Esta decisión se basa, fundamentalmente, en que la Programación Lógica corresponde a un formalismo para la representación de conocimiento ampliamente conocido. No obstante, en el Apéndice A se incluyen las principales definiciones de dicho formalismo con el objetivo de lograr una tesis autocontenida.

2.1. Servicio de Razonamiento

Un *Servicio de Razonamiento* (*SR*) constará de tres componentes (ver Figura 2.1). Su definición formal (Definición 2.3) será introducida luego de definir a cada uno de sus componentes:

Figura 2.1: Servicio de Razonamiento

1. el *conocimiento público* que será utilizado como base para la resolución de las consultas de los clientes;

2. un *mecanismo de inferencia* que utilizará ese conocimiento para computar las respuestas a consultas de los clientes; y

3. un *conjunto de operadores de tratamiento de contexto* mediante los cuales se podrá, por ejemplo, agregar en forma temporal información para crear el contexto en el cual se procesará la consulta del cliente.

Como se explicó previamente, en este capítulo se introducirán los conceptos de *SR* y las *CC* de la forma más abstracta posible, ya que estos conceptos podrán utilizarse luego con diferentes formalismos de representación de conocimiento y razonamiento. Para ello se asumirá que el formalismo concreto que se utilice para una aplicación, incluirá un lenguaje o formalismo para la representación de conocimiento público, una forma de expresar consultas y un lenguaje en el cual se expresarán las respuestas del *SR*. Estos tres elementos conformarán el *marco de representación de conocimiento* en el cual estarán definidos los conceptos de *SR*, *CC*, el mecanismo de inferencia del *SR* y los operadores que permitirán que el contexto sea utilizado adecuadamente para el cómputo de las respuestas.

De esta manera, un *marco de representación de conocimiento* será una terna $(\mathcal{D}p, \mathcal{D}c, \mathcal{D}r)$ formado por un dominio de programas $\mathcal{D}p$ y su correspondiente dominio de consultas $\mathcal{D}c$ y respuestas $\mathcal{D}r$. El dominio de programas $\mathcal{D}p$ corresponderá al lenguaje de representación. Por ejemplo, para la Programación Lógica, el dominio de programas es el conjunto de todos los posibles programas lógicos que pueden construirse (Definición A.8). En el caso del dominio de consultas, denotado con $\mathcal{D}c$, representará el conjunto de todas las consultas sintácticamente válidas que pueden construirse para el dominio de programas $\mathcal{D}p$ correspondiente. Por lo tanto, existe una relación directa entre el dominio de programas y el dominio de consultas utilizado. Por ejemplo, en la programación lógica, el dominio de las consultas corresponde al conjunto de todas las posibles consultas lógicas (Definición A.7).

En este marco, el dominio de las respuestas, denotado $\mathcal{D}r$, representa el conjunto de respuestas que se pueden obtener al hacerle una consulta (del dominio de consultas) a un programa (del dominio de programas asociado). Por ejemplo, en la programación lógica, el dominio de respuestas corresponderá al conjunto de posibles conjuntos de respuestas para consultas lógicas (Definición A.15).

El mecanismo de inferencia y los operadores de tratamiento de contexto de un *SR* dependerán del formalismo en el cual se represente el conocimiento, es decir, del *marco de representación de conocimiento* utilizado. En un *SR*, el conocimiento público será el conocimiento de base a partir del cual se obtendrán las respuestas. Los operadores permitirán la modificación temporal del conocimiento en el *SR*. En el caso del mecanismo de

inferencia, éste permitirá calcular las respuestas a las consultas a partir del conocimiento en el *SR* y ciertas modificaciones realizadas utilizando los operadores correspondientes. Por lo tanto, el mecanismo de inferencia de un *SR* será encargado de procesar y responder las consultas de los clientes.

Como se definirá a continuación, este mecanismo de inferencia, denominado intérprete, será representado en forma general como una función que dado un programa y una consulta, retorna la respuesta correspondiente.

Definición 2.1 (Intérprete)**.** *Sea* $\mathcal{M} = (\mathcal{D}p, \mathcal{D}c, \mathcal{D}r)$ *un marco de representación de conocimiento. Un* intérprete *en* $\mathcal{M}$ *es una función* $\mathcal{I} : \mathcal{D}p \times \mathcal{D}c \mapsto \mathcal{D}r$*, que dado un programa* $\mathcal{P}$ *y una consulta* Q *devuelve una respuesta* $\mathcal{R}$ *del conjunto de respuestas* $\mathcal{D}r$*.*

Ejemplo 2.1 (Intérprete de Programación Lógica). *En particular, si el marco de representación de conocimiento es la programación lógica, este marco será identificado con la terna* $\mathcal{M}_{lp} = (\mathcal{D}_{lp}, \mathcal{D}_{lq}, \mathcal{D}_{la})$*. En este contexto, un* intérprete de programación lógica *recibe un programa lógico* $\mathcal{P}_{lp}$ *y una consulta lógica* Q_{lp}*, y devuelve como respuesta un conjunto de substituciones* $\{\theta_1, \ldots, \theta_n\}$ *tal que cada* θ_i *(*$i = 1, \ldots, n$*) es una respuesta para* Q_{lp} *en* $\mathcal{P}_{lp}$*.* □

Una característica distintiva de los *SRs* que se definen en este capítulo corresponde a la posibilidad de modificar temporalmente el conocimiento público almacenado. Es decir, estos servicios proveerán operadores específicos que permitirán a consultas especiales modificar temporalmente el conocimiento que poseen almacenado para el cálculo de las respuestas. El conocimiento público en un *SR* estará representado por un programa en el dominio de programas $\mathcal{D}p$ dentro de un marco de representación de conocimiento. A continuación, se definirá el concepto de *operador de programa* como una función que dados dos programas retorna un nuevo programa obtenido luego de operar al primer programa con el segundo.

Definición 2.2 (Operador de Programa)**.** *Sea* $\mathcal{D}p$ *un dominio de programas. Un* operador de programa *es una función* $O : \mathcal{D}p \times \mathcal{D}p \mapsto \mathcal{D}p$*. Se denotará con* $\mathcal{O}_{\mathcal{D}p}$ *al conjunto de todos los operadores* O *cuyo dominios y rango son el dominio* $\mathcal{D}p$*.*

Ejemplo 2.2 (Operadores de Programa Lógico). *Considerando el marco de representación de conocimiento* $\mathcal{M}_{lp} = (\mathcal{D}_{lp}, \mathcal{D}_{lq}, \mathcal{D}_{la})$ *introducido en el Ejemplo 2.1, a con-*

tinuación se mostrarán dos posibles operadores de programa para el dominio de programa $\mathcal{D}_{lp}$.

El primero, identificado con el símbolo $\oplus$*, agrega cláusulas al programa almacenado en el SR y se puede definir de la siguiente manera: sean* $\mathcal{P}_{lp1}$ *y* $\mathcal{P}_{lp2}$ *dos programas lógicos,* $\mathcal{P}_{lp1} \oplus \mathcal{P}_{lp2} = \mathcal{P}_{lp1} \cup \mathcal{P}_{lp2}$.

El segundo operador, identificado con el operador $\ominus$*, quita cláusulas de programa enviadas como parte del contexto, en el caso que éstas sean parte del programa almacenado en el SR y se define a continuación: sean* $\mathcal{P}_{lp1}$ *y* $\mathcal{P}_{lp2}$ *dos programas lógicos,* $\mathcal{P}_{lp1} \ominus \mathcal{P}_{lp2} = \mathcal{P}_{lp1} \setminus \mathcal{P}_{lp2}$.

Considerando a $\mathcal{P}_1 = \{b, a \leftarrow b\}$, $\mathcal{P}_2 = \{b, c \leftarrow a\}$ *y* $\mathcal{P}_3 = \{a, c \leftarrow a\}$ *programas del dominio de programa* $\mathcal{D}_{lp}$*. El resultado de operar* $\mathcal{P}_1$ *con* $\mathcal{P}_2$ *utilizando el operador* $\oplus$ *es* $\mathcal{P}_1 \oplus \mathcal{P}_2 = \{b, a \leftarrow b, c \leftarrow a\}$*. Por otro lado, el resultado de operar* $\mathcal{P}_3$ *con* $\mathcal{P}_2$ *utilizando el operador* $\ominus$ *es* $\mathcal{P}_3 \ominus \mathcal{P}_2 = \{a\}$. □

El principal objetivo de los operadores de programa es proporcionar la posibilidad de modificar el conocimiento del *SR*, manteniendo el control de las modificaciones en el propio *SR*. Es decir, el *SR*, al determinar qué operadores provee, está controlando la forma en la cuál los clientes pueden modificar (temporalmente) el programa almacenado utilizando las consultas contextuales, las cuales serán definidas más adelante.

Observación 2.1. *Es importante notar que el orden de los operandos en un operador no es indistinto, ya que puede darse el caso en el cual obtengan distintos resultados las operaciones* $O(\mathcal{P}_A, \mathcal{P}_B)$ *y* $O(\mathcal{P}_B, \mathcal{P}_A)$*. En los operadores de programa lógico introducidos en el Ejemplo 2.1 puede verse que si bien no importa el orden de los operandos en el operador* $\oplus$*, esto sí ocurre en el caso del operador* $\ominus$*. En particular, no tiene el mismo resultado la operación* $\mathcal{P}_1 \ominus \mathcal{P}_2$ *que la operación* $\mathcal{P}_2 \ominus \mathcal{P}_1$.

A continuación, se define el concepto de Servicio de Razonamiento (*SR*), el cuál será el encargado de responder las consultas de los clientes. Un *SR*, mostrado en la Figura 2.2, estará compuesto por un intérprete y un programa, ambos asociados al dominio de programas $\mathcal{D}p$, y un conjunto de operadores de programa incluidos en $\mathcal{O}_{\mathcal{D}p}$.

Definición 2.3 (Servicio de Razonamiento (*SR*))**.** *Sea* $\mathcal{M}$ *un marco de representación de conocimiento y* $\mathcal{D}p$ *el dominio de programas perteneciente al marco* $\mathcal{M}$*. Un servicio de razonamiento para el marco* $\mathcal{M}$ *es una terna* $\langle \mathcal{I}, \mathbf{O}, \mathcal{P} \rangle$ *donde* $\mathcal{I}$ *es un intérprete cuyo*

marco es $\mathcal{M}$, $\mathbf{O}$ es un subconjunto de los posibles operadores $\mathcal{O}_{\mathcal{D}p}$ y $\mathcal{P}$ es un programa del dominio de programas $\mathcal{D}p$.

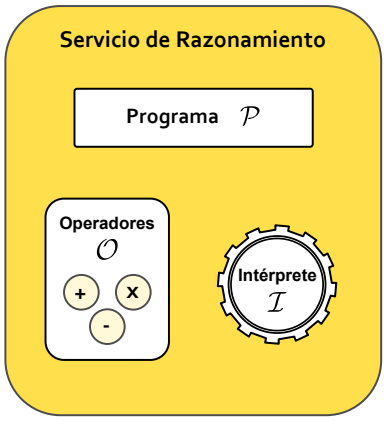

Figura 2.2: Servicio de Razonamiento

En un servicio de razonamiento $\langle \mathcal{I}, \mathbf{O}, \mathcal{P} \rangle$, el programa $\mathcal{P}$ representará el conocimiento público que será utilizado para responder las consultas de los clientes, los operadores de programa establecerán las alternativas que poseen los mismos al momento de modificar el programa $\mathcal{P}$, mientras que el intérprete será el encargado de calcular las respuestas a las consultas una vez realizadas las modificaciones temporales sobre el conocimiento público almacenado.

Ejemplo 2.3. *Considere un servicio de razonamiento $\mathcal{SR}_{tr}$ con el objetivo de sugerir tratamientos para enfermedades de acuerdo a los síntomas de los pacientes. A continuación se incluye el programa $\mathcal{P}_{tr}$, el cuál representa conocimiento público que podría poseer el servicio de razonamiento $\mathcal{SR}_{tr}$:*

$$\mathcal{P}_{tr} = \left\{ \begin{array}{l} \textit{enfermedad(e1) :- síntoma(s1), síntoma(s2), síntoma(s3).} \\ \textit{enfermedad(e2) :- síntoma(s1), síntoma(s4).} \\ \textit{enfermedad(e3) :- síntoma(s2), síntoma(s4).} \\ \textit{tratamiento(t1) :- enfermedad(e1).} \\ \textit{tratamiento(t2) :- enfermedad(e2), not(alérgico(t2)).} \\ \textit{tratamiento(t3) :- enfermedad(e2), not(enfermedad(e3)).} \\ \textit{tratamiento(t4) :- enfermedad(e2), enfermedad(e3).} \\ \textit{tratamiento(t5) :- enfermedad(e3), not(enfermedad(e2)).} \end{array} \right\}$$

Teniendo en cuenta el operador para programas lógicos $\oplus$ introducido en el Ejemplo 2.1, el cuál permite agregar información a un programa lógico, y el intérprete de programación lógica del Ejemplo 2.1, es posible definir un ejemplo de servicio de razonamiento $\mathcal{SR}_{tr} = \langle \mathcal{I}_{lp}, \{\oplus\}, \mathcal{P}_{tr} \rangle$. Como se mostrará en los siguientes ejemplos, este SR en particular, a través del intérprete $\mathcal{I}_{lp}$, podrá responder consultas sugiriendo tratamientos específicos para pacientes utilizando el programa $\mathcal{P}_{tr}$ e información particular de los pacientes agregada utilizando el operador de programa $\oplus$.

En general, es posible definir servicios de razonamiento para el marco de representación de conocimiento lógico $\mathcal{M}_{lp}$ y compuesto por la terna $\langle \mathcal{I}_{lp}, O, \mathcal{P}_{lp} \rangle$ donde $\mathcal{I}_{lp}$ es el intérprete de programación lógica, O es un conjunto de operadores de programa en $\mathcal{D}_{lp}$, como por ejemplo $\{\oplus, \ominus\}$, y $\mathcal{P}_{lp}$ es un programa lógico de $\mathcal{D}_{lp}$ representando el conocimiento del SR. □

Es importante resaltar que los operadores definidos previamente en el Ejemplo 2.1, tanto el operador $\oplus$ como $\ominus$, son sintácticos. Esto significa que al modificar el programa del Ejemplo 2.1, lo que se está modificando corresponde ni mas ni menos que a las cláusulas del programa. De todas formas, existe la posibilidad de definir nuevos operadores, los cuales podrían tener otro tipo de comportamiento. Es decir, es posible definir, por ejemplo, operadores que aseguren que ciertos hechos obtengan respuestas positivas o negativas, dependiendo de lo que se busque.

En la siguiente sección se introducen las consultas contextuales, las cuales permiten incluir junto con las consultas de los clientes un contexto en el cual responder dichas consultas. Este contexto representa información con la cuál se modificará temporalmente el conocimiento público almacenado en el *SR*, a través de los operadores disponibles, para computar las respuestas.

2.2. Consultas Contextuales

En esta sección se define el concepto de *Consulta Contextual* (CC). Este tipo de consulta tiene la particularidad de incluir información que será tenida en cuenta a la hora de calcular la respuesta. Esta información es lo que se denomina *contexto* de la consulta y, en general, representa conocimiento propio de los clientes.

El *contexto* consiste de información que modificará el conocimiento utilizado por el *SR* al momento de obtener la respuesta correspondiente. Más precisamente, un contexto está compuesto por uno o más programas que serán utilizados para modificar el conocimiento público en el *SR* utilizando los operadores provistos por el propio servicio. A continuación se define formalmente el concepto de *contexto* de la siguiente manera:

Definición 2.4 (Contexto)**.** *Sea $\mathcal{M}$ un marco de representación de conocimiento, $\mathcal{D}p$ el dominio de programas en $\mathcal{M}$ y $\mathcal{O}_{\mathcal{D}p}$ el conjunto de operadores para $\mathcal{D}p$. Un* contexto *$\mathcal{C}o$ en el dominio $\mathcal{D}p$ es una secuencia de pares $\langle(\mathcal{P}_1, O_1), (\mathcal{P}_2, O_2), \ldots, (\mathcal{P}_n, O_n)\rangle$ donde cada $\mathcal{P}_i$ es un programa del dominio $\mathcal{D}p$ y cada $O_i \in \mathcal{O}_{\mathcal{D}p}$ $(1 \leq i \leq n)$.*

Definición 2.5 (Contexto Adecuado)**.** *Sea $\mathcal{M}$ un marco de representación de conocimiento, $\mathcal{D}p$ el dominio de programas en $\mathcal{M}$ y $\mathcal{O}_{\mathcal{D}p}$ el conjunto de operadores para $\mathcal{D}p$. Sea $\mathcal{SR} = \langle \mathcal{I}, \mathbf{O}, \mathcal{P}\rangle$ un SR para $\mathcal{M}$, un contexto $\mathcal{C}o = \langle(\mathcal{P}_1, O_1), (\mathcal{P}_2, O_2), \ldots, (\mathcal{P}_n, O_n)\rangle$ en el dominio $\mathcal{D}p$ es* adecuado *para $\mathcal{SR}$ si todo $O_i \in \mathbf{O}$ $(1 \leq i \leq n)$.*

Ejemplo 2.4. *Continuando con el SR introducido en el Ejemplo 2.1, considere un paciente con los síntomas* s1 *y* s4*, y alérgico al medicamento* t1*. Este conocimiento particular puede ser representado con*

$$\mathcal{P}_{lp1} = \left\{ \begin{array}{l} sintoma(s1). \\ sintoma(s4). \\ alergico(t1). \end{array} \right\}$$

En este escenario, el contexto que representa la información que el paciente necesita sea considerada a la hora de obtener un tratamiento corresponde a la secuencia $\langle(\mathcal{P}_{lp1}, \oplus)\rangle$. Dicho contexto es adecuado para el SR introducido en el Ejemplo 2.1, ya que el único operador que incluye corresponde al operador provisto por el SR antes mencionado. □

La noción de contexto tiene como objetivo representar qué información debe tener en cuenta el *SR* para la consulta y qué tratamiento darle a dicha información. El uso de

contextos permite al cliente modificar el conocimiento que utiliza el servicio al momento de resolver las consultas, permitiendo de esta manera alterar temporalmente el programa que será utilizado por el *SR* para computar las respuestas.

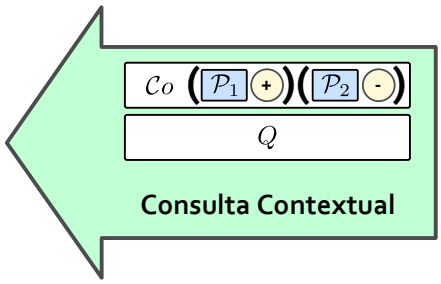

Figura 2.3: Consulta Contextual

Con el objetivo de facilitar la formalización de definiciones posteriores, a continuación se introducirá una definición que establece notación respecto de la noción de contexto. En este caso en particular, se definirá notación para especificar la actualización de un programa por un contexto, agrupando todas las modificaciones a realizar en el programa de acuerdo al contexto:

Definición 2.6 (Actualización de Programa a partir de un Contexto)**.** *Sea $\mathcal{P}$ un programa de un dominio $\mathcal{D}p$ y $\mathcal{C}o = \langle(\mathcal{P}_1, O_1), (\mathcal{P}_2, O_2), \ldots, (\mathcal{P}_n, O_n)\rangle$ un contexto en el dominio de programas $\mathcal{D}p$. La actualización de $\mathcal{P}$ con $\mathcal{C}o$, denotada $\mathcal{P} \diamond \mathcal{C}o$, se define como la composición de operadores $(O_n^{\mathcal{P}_n} \circ O_{n-1}^{\mathcal{P}_{n-1}} \circ \ldots \circ O_2^{\mathcal{P}_2} \circ O_1^{\mathcal{P}_1})(\mathcal{P})$, donde la notación $O_i^{\mathcal{P}_j}(\mathcal{P}_k)$ es una simplificación de $O_i(\mathcal{P}_j, \mathcal{P}_k)$.*

A continuación se definirán las Consultas Contextuales (*CCs*) y la forma en la cuál son resueltas por los *SRs*. Es decir, se definirá cómo es una consulta para un *SR* y cómo reacciona dicho servicio para resolver la consulta. Una Consulta Contextual (*CC*) está compuesta por un contexto y una consulta, perteneciente al dominio de consultas correspondiente (ver Figura 2.3). A continuación se introducen las definiciones de *CC* y de respuesta a una *CC* por un *SR*.

Definición 2.7 (Consulta Contextual (*CC*))**.** *Sea $\mathcal{D}p$ un dominio de programas y $\mathcal{D}c$ un dominio de consultas ambos pertenecientes al marco de representación de conocimiento*

$\mathcal{M}$. *Una* consulta contextual *en el marco* $\mathcal{M}$ *es un par* $[\mathcal{C}o, Q]$, *donde* $\mathcal{C}o$ *es un contexto en* $\mathcal{D}p$ *y* Q *es una consulta perteneciente al dominio* $\mathcal{D}c$. *La consulta contextual* $[\mathcal{C}o, Q]$ *será* adecuada *para un servicio de razonamiento* $\mathcal{SR} = \langle \mathcal{I}, \mathbf{O}, \mathcal{P} \rangle$ *en el marco* $\mathcal{M}$, *si el contexto* $\mathcal{C}o$ *es adecuado para* $\mathcal{SR}$.

Definición 2.8 (Respuesta para una *CC*)**.** *Sea* $\mathcal{M}$ *un marco de representación de conocimiento,* $\mathcal{SR} = \langle \mathcal{I}, \mathbf{O}, \mathcal{P} \rangle$ *un SR para* $\mathcal{M}$ *y* $\mathcal{CC} = [\mathcal{C}o, Q]$ *una consulta contextual adecuada para* $\mathcal{SR}$ *en* $\mathcal{M}$. *La respuesta para* $\mathcal{CC}$ *en* $\mathcal{SR}$, *denotada* $\mathcal{A}ns(\mathcal{SR}, \mathcal{CC})$, *corresponde al resultado de la función* $\mathcal{I}(\mathcal{P} \diamond \mathcal{C}o, Q)$.

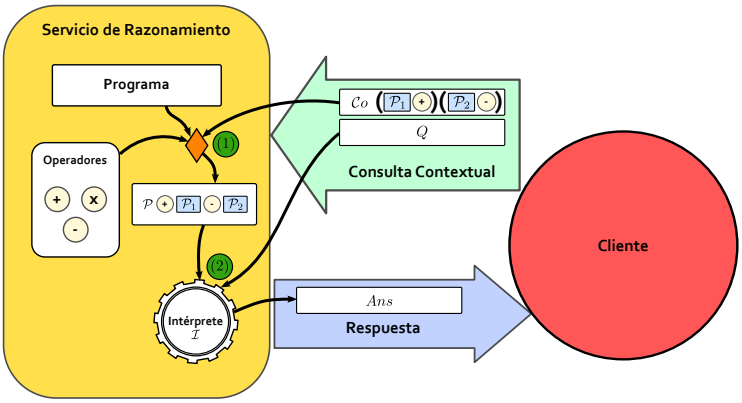

Figura 2.4: Respuesta a una Consulta Contextual

De esta manera, una *CC* provee tanto una consulta adecuada para un *SR*, como de un contexto en el cual responder a esa consulta. Para obtener dicha respuesta, el *SR* primero modifica temporalmente el programa $\mathcal{P}$ a través de aplicaciones sucesivas de las operaciones correspondientes a los programas incluidos en el contexto $\mathcal{C}o$, en el orden en que aparecen en la secuencia (ver (1) en Figura 2.4). Luego, utilizando el intérprete $\mathcal{I}$, el programa generado $(\mathcal{P} \diamond \mathcal{C}o)$ y la consulta correspondiente (Q), $\mathcal{SR}$ calcula la respuesta para $\mathcal{CC}$, siendo $\mathcal{A}ns(\mathcal{SR}, \mathcal{CC}) = \mathcal{I}(\mathcal{P} \diamond \mathcal{C}o, Q)$ (ver (2) en Figura 2.4).

Ejemplo 2.5. *En el Ejemplo 2.1 se introdujo el* $\mathcal{SR}_{tr}$ *que sugiere tratamientos a pacientes. En el Ejemplo 2.2 se mostró el contexto de un paciente con determinados síntomas*

y condiciones particulares. A continuación se mostrará cuál sería la CC que debería realizar el paciente antes mencionado para obtener una sugerencia acerca de qué tratamiento debería realizar.

En este escenario, el paciente debería enviar a $\mathcal{SR}_{tr}$ *una CC incluyendo el contexto* $\langle(\mathcal{P}_{lp1}, \oplus)\rangle$ *introducido en el Ejemplo 2.2 junto con la consulta* tratamiento(Tr)*. Es decir, la consulta contextual sería* $\mathcal{CC}_{tr} = [\langle(\mathcal{P}_{lp1}, \oplus)\rangle, tratamiento(Tr)]$.

El SR, al recibir la consulta contextual $\mathcal{CC}_{tr}$*, modificará el programa almacenado utilizando el contexto* $\langle(\mathcal{P}_{lp1}, \oplus)\rangle$*. En particular, agregará el programa* $\mathcal{P}_{lp1}$ *al programa almacenado, para luego utilizar el intérprete de programación lógica con el programa generado para calcular la respuesta a la consulta* tratamiento(Tr)*.* □

En general, una *CC* en el marco de representación de conocimiento lógico $\mathcal{M}_{lp}$, será un par $[\mathcal{C}o, Q]$, compuesto por un contexto en $\mathcal{M}_{lp}$ y su correspondiente consulta lógica, perteneciente al dominio de consultas $\mathcal{D}_{lq}$. El proceso de computar la respuesta para este tipo de consultas es el explicado previamente, sólo que en este caso, utilizando el intérprete de programación lógica.

En esta sección se mostró cómo los *SRs* responden *CCs* utilizando el conocimiento propio, modificado por el contexto de la consulta, es decir, tomando en cuenta dicho contexto a la hora de calcular la respuesta. Una de las características más interesantes de los *SRs* es que permiten que el conocimiento privado incluido en el contexto de la *CC*, sea no solamente información o datos, sino también reglas o relaciones entre la información. El ejemplo utilizado a lo largo de la sección muestra como la programación lógica permite la definición de *SRs* conteniendo programas desarrollados en dicho lenguaje y como las *CCs* extienden la alternativas a la hora de consultar a un programa lógico.

Un *SR* representa una clase especial de servicio que puede responder *CCs* de clientes, estableciendo una relación típica del estilo cliente-servidor. Por lo tanto, esto permite la existencia de múltiples clientes consultando a un mismo *SR*, así como también que un mismo cliente pueda consultar diversos *SRs*, entre otras configuraciones posibles que serán analizadas en detalle en el Capítulo 3. En la siguiente sección se muestran distintos ejemplos concretos donde se pueden aplicar los conceptos de *SR* y *CC* introducidos previamente.

2.3. Ejemplos de Aplicación

El primer ejemplo de aplicación utiliza el marco de representación de conocimiento de la programación en lógica, introducido previamente para ejemplificar los conceptos definidos en este capítulo. Se mostrarán, para este ejemplo, distintos tipos de consultas que se pueden realizar en este marco. Luego, se utilizará un marco de representación de conocimiento más elaborado, que permitirá la definición de operadores de programa más complejos. Por lo tanto, se introducirá un *SR* en este nuevo marco junto con los operadores correspondientes, para luego presentar un ejemplo de aplicación concreto utilizando dicho *SR*.

2.3.1. Servicio de Recomendación de Noticias

Los *SRs* introducidos en este capítulo y las *CCs* pueden aplicarse a la implementación de Servicios de Recomendación (Recommender Services) utilizando Programación Lógica. En esta sección se mostrará una aplicación concreta que describe un *Servicio de Recomendación de Noticias* (*SRN*) utilizando Programación Lógica. En el artículo [TGS09], se introdujo y describió esta aplicación en forma general. Un servicio de estas características podría contener noticias propiamente dichas y, a su vez, podría poseer mecanismos para extraer información de las mismas con el objetivo de recomendar a clientes, las noticias que considere más cercanas a sus preferencias.

En este contexto, el *SRN* mantiene un conjunto de noticias de actualidad y los clientes le realizan consultas en busca de noticias de su interés. Para poder realizar recomendaciones personalizadas, el servicio de recomendación debe obtener información específica de los clientes. Las *CCs* permiten a los clientes proveer de esta información al *SR*. El objetivo es doble, por un lado, mostrar una aplicación concreta, y por otro, ejemplificar los conceptos introducidos previamente. Es por este último motivo que el contenido del *SRN* será lo más simple posible.

El conocimiento público de un *SRN* específico tendrá como principal objetivo recomendar noticias deportivas. Para el desarrollo del programa almacenado en el *SR* se utilizará el dominio de programas lógico $\mathcal{D}_{lp}$ introducido en la sección anterior. Además, se utilizará un *SR* en el marco de la programación lógica, con los operadores $\oplus$ y $\ominus$ (Ejemplo 2.1), y el intérprete de programación lógica (ver Figura 2.5). Todos estos conceptos fueron introducidos en los ejemplos de la sección anterior.

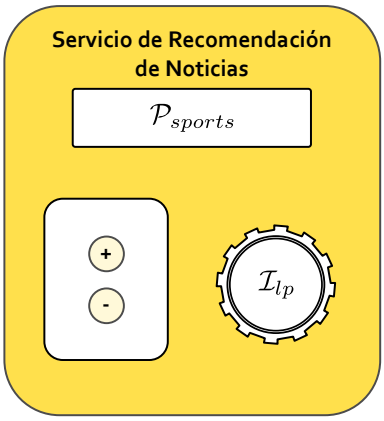

Figura 2.5: Servicio de Recomendación de Noticias

El conocimiento público del *SRN* será denominado $\mathcal{P}_{sports}$ e incluirá información específica de las noticias representada con el predicado *noticia*/5 el cuál incluye el identificador de la noticia, el deporte y el equipo al cuál se refiere la misma y el autor y el periódico que la publican (ver Figura 2.6). A continuación se mostrarán solamente las porciones de código asociadas al conocimiento público del *SRN* que son relevantes para responder las *CCs*. Es claro que el *SRN* real incluye más código, pero su inclusión no resulta necesaria para entender cómo se computan las respuestas a las *CCs*.

```
noticia (1, fútbol,    river,              autor2,   periódico1).
noticia (2, fútbol,    boca,               autor3,   periódico2).
noticia (3, básquet,   river,              autor1,   periódico1).
noticia (4, tenis,     david_nalbandian,   autor2,   periódico1).
noticia (5, fútbol,    river,              autor1,   periódico3).
noticia (6, básquet,   boca,               autor1,   periódico2).
noticia (7, fútbol,    river,              autor3,   periódico2).
```

Figura 2.6: Representación de las Noticias en el Conocimiento Público en $\mathcal{P}_{sports}$

El *SRN* incluirá, además, reglas para asociar las noticias a distintos conceptos, mostradas en la Figura 2.7. Estas reglas permiten, por ejemplo, determinar a qué deporte

pertenece una noticia, con qué equipo esta asociada la misma, entre otras características. Además, incluye hechos indicando los autores y las fuentes que serán consideradas en la búsqueda de la noticia para los clientes. Estas reglas y hechos serán utilizados para determinar si las noticias incluidas en el programa pueden ser de interés para los distintos clientes que realicen consultas.

deporte(Id,Deporte) ← *noticia(Id,Deporte ,_,_,_).*
equipo(Id,Equipo) ← *noticia(Id,_,Equipo ,_,_).*
autor(Id,Autor) ← *noticia(Id,_,_,Autor,_).*
fuente(Id,Fuente) ← *noticia(Id,_,_,_,Fuente).*

relacionada_con(Id1,Id2) ←
menciona(Id1,Palabra),
menciona(Id2,Palabra).
menciona(Id,Palabra) ←
texto_noticia(Id,Texto),
buscar_palabra_en_texto(Palabra,Texto),
relevante(Palabra).

considera(autor,autor1). *considera(fuente,periodico1).*
considera(autor,autor2). *considera(fuente,periodico2).*
considera(autor,autor3). *considera(fuente,periodico3).*

busca*(noticia(Id,Deporte,Equipo,Autor,Fuente))* ←
noticia(Id,Deporte,Equipo,Autor,Fuente),
considera(autor,Autor),
considera(fuente,Fuente),
interesado_en(Id).

Figura 2.7: Reglas de $\mathcal{P}_{sports}$ que extraen información de las noticias

Por último, el conocimiento público del *SRN* incluye el predicado *busca*, que será el encargado de determinar cuando una noticia dada resulta de interés para un cliente. A partir de este predicado, el intérprete de programación lógica intentará construir pruebas

en base a las noticias incluidas en el programa y a las preferencias del cliente. Como puede observarse en la Figura 2.7, el predicado *busca* utiliza una regla denominada *interesado_en* que no se encuentra entre las reglas del programa. Cabe destacar que esta regla deberá ser agregada por el cliente al momento de realizar la consulta, como parte del contexto, indicando qué tipo de noticias le resultan interesantes.

A partir del conocimiento público mostrado en las Figuras 2.6 y 2.7, denominado globalmente $\mathcal{P}_{sports}$, es posible definir un *SR* que recomienda noticias deportivas. Este *SR* será identificado con la terna $\langle \mathcal{I}_{lp}, \{\oplus, \ominus\}, \mathcal{P}_{sports} \rangle$. Es decir, el *SR* contiene como programa almacenado al programa $\mathcal{P}_{sports}$ incluyendo la información de las noticias y las reglas de inferencia que le permitirán al *SR* determinar si una noticia específica le puede interesar a un cliente. Además, provee los operadores $\oplus$ y $\ominus$ para modificar dicho programa, junto con el intérprete de programación lógica que nos permitirá obtener las respuestas a nuestras consultas.

Los clientes interesados en obtener recomendaciones sobre noticias deportivas deberán enviar *CCs* al *SRN* para que tenga en cuenta sus preferencias, ya sea agregando o ignorando el conocimiento privado enviado en el contexto. Con el objetivo de mostrar posibles *CCs* que se le pueden realizar al *SRN*, se asumirá la existencia de tres clientes (*A1*, *A2* y *A3*) mostrados en la Figura 2.8. A continuación se introducirán *CCs* específicas que pueden realizar estos clientes de acuerdo a sus preferencias. Se mostrarán varias consultas con el objetivo de abarcar la mayor cantidad de casos posibles que ilustren las alternativas que brindan las *CCs*.

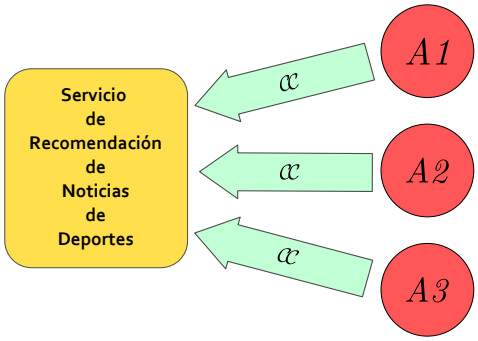

Figura 2.8: El *SRN* de deportes recibiendo consultas de múltiples clientes

En particular, el cliente *A1* está interesado solamente en noticias relacionadas con el fútbol y no quiere recibir noticias del (*periódico*1). En este caso, resulta necesario agregar al programa en el *SR* un nuevo programa lógico que contenga la regla de inferencia $interesado\_en(Id) \leftarrow deporte(Id, fútbol)$. Además, es necesario quitar del programa en el *SR* el programa que contiene al hecho $considera(fuente, periódico1)$. Observesé que *interesado_en* se representó como una regla, lo cual permite una gran flexibilidad a la hora de establecer dicha relación. Luego, el contexto lógico $\langle(\mathcal{P}_1^+, \oplus), (\mathcal{P}_1^-, \ominus)\rangle$ incluye las preferencias de este cliente, siendo:

$$\mathcal{P}_1^+ = \{interesado\_en(Id) \leftarrow deporte(Id, fútbol)\}, \text{ y}$$
$$\mathcal{P}_1^- = \{considera(fuente, periódico1)\}$$

Por otro lado, el cliente *A2* está interesado en noticias relacionadas al fútbol y, además, noticias de básquet. En este caso en particular, el nuevo programa que habría que agregar al programa en el *SR*, representando estas preferencias del cliente *A2* sería:

$$\mathcal{P}_2^+ = \left\{ \begin{array}{l} interesado\_en(Id) \leftarrow deporte(Id, fútbol), \\ interesado\_en(Id) \leftarrow deporte(Id, básquet) \end{array} \right\}$$

Teniendo en cuenta estas preferencias, el cliente *A1* podría desear obtener una noticia del Club Atlético River Plate. Es ese caso, la *CC* correspondiente sería:

$$[\langle(\mathcal{P}_1^+, \oplus), (\mathcal{P}_1^-, \ominus)\rangle, busca(noticia(Id, Deporte, river, Autor, Periódico))]$$

En el caso del cliente *A2*, éste podría desear obtener una noticia del Club Atlético Boca Juniors, en cuyo caso, la consulta contextual lógica sería la siguiente:

$$[\langle(\mathcal{P}_2^+, \oplus)\rangle, busca(noticia(Id, Deporte, boca, Autor, Periódico))]$$

Al momento de responder una *CC*, el *SRN* $\langle\mathcal{I}_{lp}, \{\oplus, \ominus\}, \mathcal{P}_{sports}\rangle$ primero computará el programa modificado, para luego calcular la respuesta a la consulta utilizando el intérprete de programa lógico. En el caso particular de la *CC* del cliente *A1*, el *SRN* primero modificará el programa lógico $\mathcal{P}_{sports}$ agregándole el programa $\mathcal{P}_1^+$ y quitándole $\mathcal{P}_1^-$, y sobre este programa calculará con el intérprete de programación lógica la respuesta a la consulta enviada, obteniendo como respuesta el par:

$$\left(SI, \left\{ \begin{array}{l} busca(noticia(5, fútbol, river, autor1, periódico3)), \\ busca(noticia(7, fútbol, river, autor3, periódico2)) \end{array} \right\}\right)$$

En cambio, para responder la *CC* del cliente *A2*, el *SRN* modificará el programa lógico $\mathcal{P}_{sports}$ agregándole el programa $\mathcal{P}_2^+$ y sobre este programa calculará la respuesta utilizando el intérprete, obteniendo el siguiente par:

$$(SI, \{busca(noticia(2, fútbol, boca, autor3, periódico2))\}).$$

Es importante mencionar que ambas consultas pueden ser procesadas simultáneamente por un *SR*, sin que las modificaciones temporales realizadas en el cálculo de la respuesta de una consulta afecte al cómputo de la respuesta de la otra consulta.

Por otro lado, el cliente *A3* puede estar interesado en noticias de *tenis*. En especial, en aquellas que tengan a *David Nalbandian* como protagonista, donde se mencione la *Copa Davis* y que no hayan sido publicadas por *periódico2*. En este caso, el cliente *A3* podría ejecutar la *CC* $[\langle(\mathcal{P}_3^+, \oplus), (\mathcal{P}_3^-, \ominus)\rangle, Q_3]$ siendo:

$$\begin{array}{l} \mathcal{P}_3^+ = \{interesado\_en(Id) \leftarrow menciona(Id, copa\_davis) \\ \mathcal{P}_3^- = \{considera(fuente, periódico2)\} \\ Q_3 = busca(noticia(Id, tenis, david\_nalbandian, Autor, Periódico)) \end{array}$$

En este caso, al igual que en los ejemplos anteriores, el *SRN* $\langle \mathcal{I}_{lp}, \{\oplus, \ominus\}, \mathcal{P}_{sports}\rangle$ primero computará el programa modificado, para luego calcular la respuesta. Cabe destacar que todas las modificaciones realizadas en el *SRN* para responder las consultas anteriores han sido descartadas y se iniciará el cálculo de la nueva respuesta utilizando el programa $\mathcal{P}_{sports}$ originalmente almacenado en el *SRN*. Dicho *SRN* agrega al programa almacenado el programa $\mathcal{P}_3^+$ y luego quita el programa $\mathcal{P}_3^-$, y sobre este nuevo programa ejecuta el intérprete para computar la respuesta a Q_3. Finalmente, el *SRN* devolverá la siguiente respuesta al cliente *A3*:

$$(SI, \{busca(noticia(4, tenis, david\_nalbandian, autor2, periódico1))\})$$

Considere el caso en el cual tanto el cliente *A1* como el cliente *A2* están interesados en noticias de los clubes *River Plate* o *Boca Juniors* publicadas por *periódico2* y que no hayan sido escritas por *autor2*. En particular, el cliente *A1* desea obtener noticias

relacionadas con el básquet, mientras que el cliente *A2* está buscando noticias de fútbol. Luego, el cliente *A1* ejecutará la CC $[\langle(\mathcal{P}_4^+, \oplus), (\mathcal{P}_4^-, \ominus)\rangle, Q_{4,1}]$, mientras que el cliente *A2* ejecutará la CC $[\langle(\mathcal{P}_4^+, \oplus), (\mathcal{P}_4^-, \ominus)\rangle, Q_{4,2}]$, siendo:

$$\mathcal{P}_4^+ = \left\{ \begin{array}{l} interesado\_en(Id) \leftarrow equipo(Id, river), \\ interesado\_en(Id) \leftarrow equipo(Id, boca) \end{array} \right\}$$
$$\mathcal{P}_4^- = \{considera(autor, autor2)\}$$
$$Q_{4,1} = busca(noticia(Id, básquet, Equipo, Autor, periódico2))$$
$$Q_{4,2} = busca(noticia(Id, fútbol, Equipo, Autor, periódico2))$$

Ante cualquiera de estas CCs, el SRN $\langle \mathcal{I}_{lp}, \{\oplus, \ominus\}, \mathcal{P}_{sports}\rangle$ primero computará el programa modificado, para luego calcular la respuesta. En este caso en particular, donde los contextos de ambas consultas son iguales, las modificaciones necesarias en el SRN para calcular las respuestas a las consultas deberán ser realizadas sobre el conocimiento público, de manera repetida, para cada una de las consultas enviadas. Por lo tanto, el SRN agrega al programa almacenado el programa $\mathcal{P}_4^+$ y luego quita el programa $\mathcal{P}_4^-$, y sobre este nuevo programa ejecuta el intérprete para computar la respuesta para la consulta de, por ejemplo, *A2*, obteniendo:

$$\left(SI, \left\{ \begin{array}{l} busca(noticia(2, fútbol, boca, autor3, periódico2)), \\ busca(noticia(7, fútbol, river, autor3, periódico2)) \end{array} \right\}\right)$$

Luego, en el caso de la consulta enviada por el cliente *A1*, nuevamente el SRN modifica el programa almacenado agregando el programa $\mathcal{P}_4^+$ y luego quita el programa $\mathcal{P}_4^-$ y obtiene, utilizando el intérprete, la siguiente respuesta para $Q_{4,1}$:

$$(SI, \{busca(noticia(6, básquet, boca, autor1, periódico2))\})$$

Otra posibilidad es que los tres clientes (*A1*, *A2* y *A3*) estén interesados en noticias de *fútbol* publicadas por *periódico2*. En el caso del cliente *A1*, éste se encuentra interesado en aquellas noticias que mencionen el *clásico* entre los clubes River Plate y Boca Juniors. El cliente *A2*, en cambio, está buscando noticias que hablen del jugador *Ortega*. Finalmente, el cliente *A3* desea encontrar noticias relacionadas con el *clásico* y donde se mencione al jugador *Riquelme*.

En este escenario, el cliente *A1* ejecutará la CC $[\langle(\mathcal{P}_{5,1}^+, \oplus)\rangle, Q_5]$, el cliente *A2* ejecutará la CC $[\langle(\mathcal{P}_{5,2}^+, \oplus)\rangle, Q_5]$, y finalmente, el cliente *A3* ejecutará la CC $[\langle(\mathcal{P}_{5,3}^+, \oplus)\rangle, Q_5]$, siendo:

$$\mathcal{P}^{+}_{5,1} = \left\{ \begin{array}{l} interesado\_en(Id) \leftarrow menciona(Id, clásico), equipo(Id, river), \\ interesado\_en(Id) \leftarrow menciona(Id, clásico), equipo(Id, boca) \end{array} \right\}$$
$$\mathcal{P}^{+}_{5,2} = \{interesado\_en(Id) \leftarrow menciona(Id, ortega)\}$$
$$\mathcal{P}^{+}_{5,3} = \{interesado\_en(Id) \leftarrow (menciona(Id, clásico), menciona(Id, riquelme))\}$$
$$Q_5 = busca(noticia(Id, fútbol, Equipo, Autor, periódico2))$$

El *SRN* $\langle \mathcal{I}_{lp}, \{\oplus, \ominus\}, \mathcal{P}_{sports} \rangle$, al momento de responder estas consultas, primero computará el programa modificado, para luego calcular la respuesta. Por ejemplo, para la consulta $[\langle(\mathcal{P}^{+}_{5,2}, \oplus)\rangle, Q_5]$, el *SRN* agrega al programa almacenado el programa $\mathcal{P}^{+}_{5,2}$ y sobre este nuevo programa ejecuta el intérprete para computar la respuesta para la consulta Q_5, obteniendo:

$$(SI, \{busca(noticia(7, fútbol, river, autor3, periódico2))\})$$

En el caso de la consulta $[\langle(\mathcal{P}^{+}_{5,1}, \oplus)\rangle, Q_5]$, el *SRN* agrega al programa almacenado el programa $\mathcal{P}^{+}_{5,1}$ y computa la respuesta sobre este nuevo programa utilizando el intérprete y logrando la siguiente respuesta:

$$\left(SI, \left\{ \begin{array}{l} busca(noticia(2, fútbol, boca, autor3, periódico2)), \\ busca(noticia(7, fútbol, river, autor3, periódico2)) \end{array} \right\}\right)$$

Finalmente, para la consulta $[\langle(\mathcal{P}^{+}_{5,3}, \oplus)\rangle, Q_5]$, el *SRN* agrega al programa almacenado el programa $\mathcal{P}^{+}_{5,3}$ y calcula la respuesta a la consulta Q_5 sobre este nuevo programa, obteniendo la siguiente respuesta:

$$(SI, \{busca(noticia(2, fútbol, boca, autor3, periódico2))\})$$

En este escenario se puede observar como afectan los contextos en la resolución de las consultas, ya que los tres clientes realizaron la misma consulta con distintos contextos y cada uno de ellos obtuvo una respuesta diferente. En la siguiente sección se mostrará otra versión de *SR*, en este caso, en el marco de representación de conocimiento DeLP, con sus operadores de programa y las consultas contextuales correspondientes.

2.3.2. Servicio de Razonamiento DeLP

En esta sección se introducirá un nuevo tipo de *SR*, denominado *servicio de razonamiento DeLP*. Este tipo de servicio de razonamiento, así como también las consultas contextuales que serán utilizadas para obtener respuestas, están basadas en la programación lógica rebatible. La programación lógica rebatible combina la programación lógica con la argumentación rebatible, resultando en una herramienta atractiva para representar conocimiento y definir el razonamiento de agentes deliberativos. Además, este lenguaje de programación, al igual que la programación lógica, permite especificar un nuevo tipo de servicio de razonamiento, como caso particular del definido al inicio de este capítulo.

En DeLP, el conocimiento es representado utilizando hechos, reglas estrictas y reglas rebatibles. Los *hechos* son literales fijos representando información básica o la negación de cierta información básica utilizando la negación fuerte (representada en esta tesis con el símbolo '$\sim$'). Las *reglas estrictas* representan información segura y libre de excepciones, mientras que las *reglas rebatibles* representan información tentativa. En los ejemplos introducidos en esta tesis se utiliza una versión restringida de DeLP, que incluye únicamente *hechos* y *reglas rebatibles*, éstas últimas, denotadas de la siguiente manera:

$$L_0 \prec L_1, \ldots, L_n.$$

Siendo la *cabeza* L_0 un literal y el *cuerpo* $L_1, \ldots, L_n$ un conjunto de literales.

Las reglas rebatibles permiten el uso de la negación fuerte en la cabeza de dichas reglas, y por lo tanto, la representación de información contradictoria. Luego, de un programa DeLP restringido se pueden derivar literales contradictorios, pese a que el conjunto de hechos debe mantener la coherencia interna (no puede contener literales contradictorios entre sí). Dado un literal L, el complemento con respecto a la negación fuerte será denotado $\sim L$. Agregar hechos a un programa DeLP puede producir un conjunto de hechos contradictorio, y por lo tanto, un programa DeLP inválido. En cambio, las reglas rebatibles pueden ser agregadas sin ningún tipo de restricción.

A lo largo de esta sección se utilizará el programa DeLP $\mathcal{P}^{Ej}_{delp}$ para ejemplificar los conceptos que serán introducidos. Este programa contiene información sobre el mercado de valores y se muestra a continuación:

$$\mathcal{P}^{Ej}_{delp} = \left\{ \begin{array}{ll} comprar\_acciones(X) \prec buen\_precio(X). & \\ \sim comprar\_acciones(X) \prec buen\_precio(X), empresa\_riesgosa(X). & \\ empresa\_riesgosa(X) \prec en\_fusión(X). & \\ empresa\_riesgosa(X) \prec tiene\_deuda(X). & \\ & \\ buen\_precio(a). & en\_fusión(a). \\ buen\_precio(b). & tiene\_deuda(b). \\ buen\_precio(c). & tiene\_deuda(c). \end{array} \right\}$$

Al momento de razonar con información contradictoria y dinámica, DeLP construye *argumentos* utilizando partes de esta información y los compara para decidir cuál prevalece. En el programa DeLP $\mathcal{P}^{Ej}_{delp}$ se puede construir el argumento $\mathcal{A}_1$ en favor de comprar acciones de la empresa a:

$$\mathcal{A}_1 = \left\{ \begin{array}{l} comprar\_acciones(a) \prec buen\_precio(a). \\ buen\_precio(a). \end{array} \right\}$$

Aunque también se puede construir el argumento $\mathcal{A}_2$ en favor de no comprar acciones de la empresa a:

$$\mathcal{A}_2 = \left\{ \begin{array}{ll} \sim comprar\_acciones(a) \prec buen\_precio(a), empresa\_riesgosa(a). & \\ empresa\_riesgosa(a) \prec en\_fusión(a). & \\ buen\_precio(a). & en\_fusión(a). \end{array} \right\}$$

El argumento que logra prevalecer se constituye en una *garantía* para la información que soporta. En DeLP, una consulta Q estará *garantizada* a partir de un determinado programa DeLP, si existe un argumento que soporta a Q y no está derrotado. Para determinar si un argumento está o no derrotado, deben considerarse los posibles *derrotadores* para dicho argumento, es decir, contra-argumentos que por algún criterio son preferidos con respecto al argumento original. Es claro que el argumento $\mathcal{A}_2$ es un posible derrotador del argumento Arg_1 y viceversa.

Es importante aclarar que en DeLP, el criterio de comparación es modular, y por lo tanto, permite el empleo del criterio que se considere más apropiado para el dominio de aplicación en particular. A lo largo de esta tesis se utilizará el criterio de *especificidad generalizada*, el cuál favorece dos aspectos de un argumento: (1) un argumento *más preciso* (conteniendo más información) o (2) un argumento más conciso (utilizando menos reglas).

Utilizando este criterio de comparación, el argumento $\mathcal{A}_1$ es derrotado por el argumento $\mathcal{A}_2$, siendo este último más preciso.

Un derrotador para un argumento puede ser *propio* (preferido por el criterio de comparación) o *de bloqueo* (ninguno de los dos es preferido por el criterio de comparación). A su vez, un derrotador puede atacar la conclusión de otro argumento o a un punto interno del mismo. Considerando que los derrotadores son argumentos, pueden existir nuevos derrotadores para ellos, incluso derrotadores de estos derrotadores, y así siguiendo, surgiendo de esta manera las *líneas de argumentación*, las cuales corresponden a una secuencia argumentos y derrotadores.

Claramente, pueden existir más de un derrotador para un argumento en particular, dando lugar a múltiples líneas de argumentación para un mismo argumento. Estas múltiples líneas de argumentación generan una estructura de árbol, denominado, *árbol de dialéctica*. En un árbol de dialéctica, cada nodo (excepto la raíz) es un derrotador para su padre y los nodos hojas corresponden a argumentos no derrotados. Un árbol de dialéctica permite considerar todas las posibles líneas de argumentación que pueden generarse, posibilitando, además, marcar a cada nodo como derrotado o no derrotado, dependiendo de la estructura del árbol. Este procedimiento de marcado permitirá determinar el estado de garantía para un determinado argumento.

En el Capítulo 5 se describirá más en detalle la construcción de estos árboles de dialéctica. En lo que respecta a una caracterización más formal de DeLP, en el Apéndice B se incluyen las principales definiciones de este formalismo, incluyendo los conceptos de programa, consulta y respuesta. El marco de representación de conocimiento de la programación lógica rebatible está representado por la terna $(\mathcal{D}_{delp}, \mathcal{D}_{delq}, \mathcal{D}_{dela})$, donde el domino de programas $\mathcal{D}_{delp}$ corresponde al conjunto de todos los posibles programas DeLP (Definición B.4). El dominio de consultas $\mathcal{D}_{delq}$ es el conjunto de todas las posibles consultas que se le pueden realizar a un programa DeLP (Definición B.5), mientras que el dominio de respuestas $\mathcal{D}_{dela}$ está representado por el conjunto de todos los pares $(\mathcal{A}ns, Just)$, donde $\mathcal{A}ns$ puede ser una de las siguientes respuestas {SI, NO, INDECISO, DESCONOCIDO} y $Just$ es un conjunto de literales fijos, representando las posibles respuestas que puede entregar un programa DeLP a una consulta (Definición B.15).

A continuación se introducirá el concepto de intérprete DeLP, definido como una función que dados un programa DeLP y una consulta DeLP, devuelve una respuesta DeLP con las siguientes características:

Definición 2.9 (Intérprete DeLP)**.** *Un* intérprete DeLP *para el marco de representación de conocimiento DeLP es una función* $\mathcal{I}_{delp} : \mathcal{D}_{delp} \times \mathcal{D}_{delq} \rightarrow \mathcal{D}_{dela}$ *que dada un programa DeLP* $\mathcal{P}_{delp}$ *y una consulta DeLP* Q *devuelve el par:*

- (SI, $Just$)*, si existe un* q *tal que sea una instancia fija de* Q *y esté garantizado a partir de* $\mathcal{P}_{delp}$ *siendo* $Just$ *es el conjunto de todas las instancias fijas de* Q *garantizadas en* $\mathcal{P}_{delp}$*;*
- (INDECISO, $Just$)*, si no existe una instancia fija de* Q *que esté garantizada a partir de* $\mathcal{P}_{delp}$*, pero existe al menos un* q *tal que sea instancia fija de* Q *y ni* q *ni su complemento estén garantizados a partir de* $\mathcal{P}_{delp}$*. El conjunto* $Just$ *estará compuesto por todas las instancias fijas de* Q *cuya respuesta es* INDECISO *en* $\mathcal{P}_{delp}$*;*
- (NO, $\{\}$)*, si para todo* q *tal que* q *es una instancia fija de* Q*, el complemento de* q *está garantizado a partir de* $\mathcal{P}_{delp}$ *o*
- (DESCONOCIDO, $\{\}$)*, si* Q *no pertenece a la signatura del programa* $\mathcal{P}_{delp}$*.*

Los programas DeLP presentan nuevas oportunidades a la hora de modificar su conocimiento o información. En particular, el hecho de agregar información nueva (en especial, agregar reglas estrictas o hechos) puede tornar a los programas modificados en inconsistentes. Teniendo en cuenta que se asumirá, como fue dicho previamente, que en un programa DeLP, el conjunto de hechos y reglas estrictas es no contradictorio, éstas modificaciones podrían generar que los programas DeLP dejen de ser válidos. Es por esto que los operadores de programas DeLP a definir deben tener en cuenta esta posibilidad de generar programas inválidos.

Como se mencionó previamente, en esta tesis se utiliza un tipo especial de programas DeLP, denominados programas DeLP restringidos, los cuales asumen que el conjunto de reglas estrictas de los programas DeLP es vacío. En estos programas DeLP restringidos, el conjunto de hechos mantiene la restricción de que no puede tener literales contradictorios. Por lo tanto, un operador de programa DeLP no puede retornar un programa que contenga el mismo literal, tanto positiva como negativamente en el conjunto de hechos. Para lograr esto existen en un principio tres posibilidades al momento de agregar información: asumir que la unión de los conjuntos de hechos de los programas de los operandos no es inconsistente o, en caso contrario, otorgar prioridad al primer operando o al segundo.

En el caso de la primera alternativa, la responsabilidad de evitar la generación de programas inconsistentes descansa en el cliente. La ventaja de esta posibilidad es que minimiza la necesidad de control por parte del servicio de razonamiento al momento de actualizar el programa. Por otro lado, la principal desventaja de este posible operador es que, en general, el cliente desconocerá el programa almacenado en el servicio de razonamiento y, por lo tanto, puede no contar con las herramientas necesarias para lograr evitar la creación de un programa inconsistente.

Las restantes alternativas resuelven los posibles problemas de consistencia prefiriendo los literales de alguno de los dos operandos. Esto es posible debido a que los operandos individualmente son consistentes y, en el caso de los programas DeLP restringidos, solo puede surgir inconsistencia al unir los conjuntos de hechos de los programas DeLP. Por lo tanto, al priorizar uno de los operandos, se tomará como base de la unión al programa DeLP correspondiente al operando priorizado y se agregarán, del otro operando, los hechos que no contradigan a los existentes y todas las reglas rebatibles, ya que estas últimas no pueden generar programas inválidos.

Cuando se considera la posibilidad de definir operadores sobre programas DeLP completos, incluyendo las reglas estrictas, los posibles cursos de acción para evitar generar programas inválidos aumentan. Es decir, no alcanza con decidir si priorizar un operando u otro, es necesario tomar más decisiones respecto de que reglas estrictas y hechos se incluyen y cuales no, con el objetivo de mantener la consistencia del programa DeLP. Estas decisiones están íntimamente relacionadas con el área de investigación de revisión de creencias y su análisis escapa al alcance de esta tesis.

Por lo tanto, queda claro que es posible definir operadores que modifiquen programas DeLP completos. Por cuestiones de simplicidad, en esta tesis únicamente se definen los operadores sobre programas DeLP restringidos. En lo que resta de la tesis, a menos que se indique lo contrario, cuando se hable de programa DeLP, siempre se estará asumiendo que es un programa DeLP restringido.

A continuación se introducirán tres operadores de programa DeLP, los cuales fueron publicados en [GRTS07]. El primero, denominado *Priorizado*, le otorga prioridad a la información enviada como parte del contexto. Mientras que en el caso del segundo, denominado *No Priorizado*, la prioridad la tiene la información almacenada en el servicio de razonamiento.

El *operador priorizado* "+" es un ejemplo de operador para el dominio de programas DeLP $\mathcal{D}_{delp}$. Es decir, dados (Π_1, Δ_1) y (Π_2, Δ_2) dos programas DeLP y siendo $C(\Pi) = \{\overline{L}$ if $L \in \Pi\}$, luego $(\Pi_1, \Delta_1)+(\Pi_2, \Delta_2) = ((\Pi_1 \setminus C(\Pi_2)) \cup \Pi_2, \Delta_1 \cup \Delta_2)$.

Otro ejemplo de operador para el dominio de programas DeLP $\mathcal{D}_{delp}$corresponde al *operador no priorizado* "$*$". En este caso, dados (Π_1, Δ_1) y (Π_2, Δ_2) dos programas DeLP, luego $(\Pi_1, \Delta_1)*(\Pi_2, \Delta_2) = (\Pi_1 \cup (\Pi_2 \setminus C(\Pi_1)), \Delta_1 \cup \Delta_2)$.

Ejemplo 2.6. *A continuación se introducen diversos programas DeLP representando cierta información que posee un cliente C1 y que será utilizada para modificar el programa* $\mathcal{P}^{Ej}_{delp}$. *El programa* $\mathcal{P}1$ *incluye tres hechos con información respecto de las empresas* b *y* c*:*

$$\mathcal{P}1 = \{\sim tiene\_deuda(b), en\_fusión(c), internacional(c)\}$$

Por otro lado, el programa $\mathcal{P}2$ *contiene un hecho con información de la empresa* a*, y una regla rebatible indicando que una empresa no será considerada riesgosa si pese a estar en fusión, es una empresa internacional:*

$$\mathcal{P}2 = \{\sim en\_fusión(a), (\sim empresa\_riesgosa(X) \prec en\_fusión(X), internacional(X))\}$$

Estos programas podrían ser utilizados por el cliente C1 para modificar el programa $\mathcal{P}^{Ej}_{delp}$ *utilizando tanto el operador priorizado o el no priorizado. Por ejemplo, una alternativa sería modificar a* $\mathcal{P}^{Ej}_{delp}$ *con* $\mathcal{P}1$ *utilizando el operador priorizado y al resultado obtenido modificarlo con* $\mathcal{P}2$ *utilizando el operador no priorizado. Es decir, ejecutar* $(\mathcal{P}^{Ej}_{delp}+\mathcal{P}1)*\mathcal{P}2$*, obteniendo como resultado el siguiente programa DeLP:*

$$(\mathcal{P}^{Ej}_{delp}+\mathcal{P}1)*\mathcal{P}2 = \left\{ \begin{array}{ll} comprar\_acciones(X) \prec buen\_precio(X). & \\ \sim comprar\_acciones(X) \prec buen\_precio(X), empresa\_riesgosa(X). & \\ empresa\_riesgosa(X) \prec en\_fusión(X). & \\ empresa\_riesgosa(X) \prec tiene\_deuda(X). & \\ \sim empresa\_riesgosa(X) \prec en\_fusión(X), internacional(X). & \\ & \\ buen\_precio(a). & en\_fusión(a). \\ buen\_precio(b). & tiene\_deuda(b). \\ buen\_precio(c). & tiene\_deuda(c). \\ en\_fusión(c). & internacional(c). \end{array} \right\}$$

Como puede observarse, todos los hechos de $\mathcal{P}1$ *están contenidos en el programa DeLP obtenido como resultado de la ejecución de las operaciones. Esto se debe a que* $\mathcal{P}1$ *fue agregado utilizando el operador priorizado, el cual le otorga prioridad a la información nueva sobre la existente. En cambio, el hecho* $\sim en\_fusión(a)$ *incluido en* $\mathcal{P}2$ *no es parte del programa resultante. Esto se debe a que en este caso, para agregar* $\mathcal{P}2$*, se utilizó el operador no priorizado, con lo cual, la prioridad la obtuvo el programa* $\mathcal{P}^{Ej}_{delp}$*.* □

Por otro lado, puede ocurrir que el cliente desee quitar información o conocimiento del programa en el servicio de razonamiento, en cuyo caso, no existe la posibilidad de generar un programa DeLP inconsistente. Por lo tanto, un operador que *restringa* la información del servicio de razonamiento resulta muy similar al introducido para programas lógicos. Luego, el *operador restrictivo* "$-$" corresponde a otro ejemplo más de operador para el dominio de programas DeLP $\mathcal{D}_{delp}$. Dados (Π_1, Δ_1) y (Π_2, Δ_2) dos programas DeLP, $(\Pi_1, \Delta_1) - (\Pi_2, \Delta_2) = (\Pi_1 \setminus \Pi_2, \Delta_1 \setminus \Delta_2)$.

Ejemplo 2.7. *Para este ejemplo se utilizará el programa* $\mathcal{P}3$*, incluyendo un hecho con información de la empresa* c *y una regla rebatible indicando que una empresa es considerada riesgosa si tiene deuda. En este caso se define a este programa de la siguiente forma:*

$$\mathcal{P}3 = \{tiene\_deuda(c), empresa\_riesgosa(X) \prec tiene\_deuda(X)\}.$$

Una posible operación es quitarle a $\mathcal{P}^{Ej}_{delp}$ *el programa* $\mathcal{P}3$*. Es decir, ejecutar* $\mathcal{P}^{Ej}_{delp} - \mathcal{P}3$*, obteniendo el siguiente resultado:*

$$\mathcal{P}^{Ej}_{delp} - \mathcal{P}3 = \left\{ \begin{array}{ll} comprar\_acciones(X) \prec buen\_precio(X). & \\ \sim comprar\_acciones(X) \prec buen\_precio(X), empresa\_riesgosa(X). & \\ empresa\_riesgosa(X) \prec en\_fusión(X). & \\ & \\ buen\_precio(a). & en\_fusión(a). \\ buen\_precio(b). & tiene\_deuda(b). \\ buen\_precio(c). & \end{array} \right\}$$

□

Al inicio de esta sección se introdujo el concepto de intérprete DeLP. Utilizando este intérprete y los operadores mencionados previamente es posible definir el concepto de

servicio de razonamiento DeLP (*SR DeLP*). Un servicio de razonamiento DeLP para un dominio de programas $\mathcal{D}_{delp}$ es una terna $\langle \mathcal{I}_{delp}, O, \mathcal{P}_{delp} \rangle$ donde $\mathcal{I}_{delp}$ es un intérprete DeLP, O es el conjunto de operadores $\{+, *, -\}$ y $\mathcal{P}_{delp}$ es un programa en el dominio $\mathcal{D}_{delp}$.

Ejemplo 2.8. *Es posible definir un servicio de razonamiento DeLP conteniendo el programa $\mathcal{P}^{Ej}_{delp}$. Este servicio de razonamiento se define de la siguiente forma: $\langle \mathcal{I}_{delp}, \{+, *, -\}, \mathcal{P}^{Ej}_{delp} \rangle$, y sugiere a los clientes que lo consulten acciones de qué empresa comprar de acuerdo a la información del estado de las mismas.* □

Un *SR DeLP* utilizará *contextos DeLP* para modificar su programa almacenado, con el objetivo de resolver consultas de clientes utilizando información o conocimiento provistos por ellos mismos. Un contexto DeLP es una secuencia de pares $(\mathcal{P}_{delp}, O)$ donde cada $\mathcal{P}_{delp}$ es un programa DeLP y cada O está incluido en el conjunto de operadores presentados previamente ($\{+, *, -\}$).

Ejemplo 2.9. *Es posible definir diversos contextos DeLP utilizando los programas definidos en ejemplos previos. Por ejemplo, a continuación se define un contexto lógico para el cliente C1 utilizando los programas $\mathcal{P}1$ y $\mathcal{P}2$ definidos en el Ejemplo 2.3.2 y el programa $\mathcal{P}3$ definido en el Ejemplo 2.3.2. En este escenario, el cliente C1 podría querer agregar el programa $\mathcal{P}1$ utilizando el operador priorizado, al resultado obtenido agregarle $\mathcal{P}2$, pero esta vez empleando el operador no priorizado, y finalmente quitarle la información incluida en el programa $\mathcal{P}3$. El contexto DeLP que logra este comportamiento se define a continuación: $\langle (\mathcal{P}1, +), (\mathcal{P}2, *), (\mathcal{P}3, -) \rangle$.* □

Los contextos DeLP permiten a los clientes modificar el programa en el *SR DeLP*, ya sea otorgándole prioridad a la información enviada por los propios clientes, como también prefiriendo el conocimiento almacenado en el servidor. Además, también permite ignorar cierta información que consideren no relevante. Luego, una *consulta contextual DeLP* estará compuesta por un contexto DeLP y su correspondiente consulta en el dominio de consulta $\mathcal{D}_{delq}$.

Al recibir una *CC DeLP*, un *SR DeLP* primero generará un nuevo programa a partir del contexto de la consulta, a través de ejecutar las operaciones sucesivas (correspondientes a los operadores del contexto), para luego calcular la respuesta utilizando el intérprete DeLP.

Ejemplo 2.10. *El cliente C1 está interesado en obtener una sugerencia acerca de cuáles empresas conviene comprar acciones. De acuerdo al conocimiento que posee dicho cliente, introducido en los ejemplos anteriores, éste podría consultar al servicio de razonamiento* $\langle \mathcal{I}_{delp}, \{+, *, -\}, \mathcal{P}^{Ej}_{delp} \rangle$ *por* $comprar\_acciones(E)$. *Esta consulta estaría conformada de la siguiente manera:* $(\langle (\mathcal{P}1, +), (\mathcal{P}2, *), (\mathcal{P}3, -) \rangle, comprar\_acciones(E))$. *La respuesta que brinda el servicio de razonamiento* $\langle \mathcal{I}_{delp}, \{+, *, -\}, \mathcal{P}^{Ej}_{delp} \rangle$ *a dicha consulta contextual lógica rebatible es* $(SI, \{comprar\_acciones(b), comprar\_acciones(c)\})$, *indicando que, de acuerdo al análisis realizado por el servicio de razonamiento, es recomendable comprar acciones tanto de la empresa* b *como de la* c. □

En esta sección se introdujo la versión del *SR* en el marco de representación de conocimiento DeLP, junto con las *CCs* correspondientes. Para mostrar su uso se utilizó un ejemplo relacionado con el mercado de valores. En la siguiente subsección se mostrará un ejemplo de aplicación en el cuál un *SR DeLP* recomienda departamentos de acuerdo a las características y preferencias de los clientes.

2.3.3. Servicio DeLP de Recomendación de Inmuebles

Como aplicación particular de un *SR DeLP*, se muestra un *Servicio de Recomendación de Inmuebles* (*SRI*). En este caso, el *SRI* contiene información acerca de inmuebles y, a su vez, razones a favor y en contra de los distintos inmuebles con el objetivo de estar capacitado para recomendar a los clientes, los inmuebles que considere más cercanos a sus preferencias. El *SRI* utilizará el marco de representación de conocimiento $(\mathcal{D}_{delp}, \mathcal{D}_{delq}, \mathcal{D}_{dela})$, el intérprete DeLP y el conjunto de operadores $\{+, *, -\}$.

El *SRI* mantiene un conjunto de inmuebles disponibles para alquiler y/o venta y los clientes le realizan consultas de acuerdo a sus intereses y preferencias. Al igual que en el *SRN*, el *SRI* debe obtener información específica de los clientes. Es por esto que las *CCs* resultan útiles, ya que les permiten a los clientes proveer al servicio de información personalizada.

A continuación se introduce un *SR DeLP* específico, siendo su objetivo principal el de representar un *SRI* que recomiende departamentos en alquiler. Para el desarrollo del programa almacenado en el *SRI* se utilizará el dominio de programas $\mathcal{D}_{delp}$ mencionado en la sección anterior. Además, el *SRI* incluirá el operador priorizado, el no-priorizado y el

restrictivo, y el intérprete DeLP. Todos estos conceptos fueron introducidos en la sección anterior.

Una versión reducida del programa DeLP que será utilizado en lo que resta de la sección para representar el conocimiento del *SRI* es introducido a continuación. La información respecto de los departamentos en alquiler será almacenada utilizando hechos, incluyendo la disponibilidad, la ubicación, así como también sus características:

$$\mathcal{P}_{inf} = \left\{ \begin{array}{lll} disponible(depto1) & disponible(depto2) & disponible(depto3) \\ afueras(depto1) & céntrico(depto2) & afueras(depto3) \\ dos\_dormitorios(depto1) & dos\_dormitorios(depto2) & living\_grande(depto3) \\ subte(depto1) & \sim subte(depto2) & \sim subte(depto3) \\ económico(depto1) & & económico(depto3) \end{array} \right\}$$

Además, el programa estará formado por reglas rebatibles representando el conocimiento para construir argumentos a favor y en contra de cada departamento, de acuerdo a las características de los clientes.

$$\mathcal{P}_{con} = \left\{ \begin{array}{l} prefiere\_afueras(X) \prec auto. \\ prefiere\_afueras(X) \prec \sim auto, subte(X). \\ \sim prefiere\_afueras(\_) \prec \sim auto. \\ quiere\_living\_grande \prec pareja\_joven. \\ \sim quiere\_living\_grande \prec quiere\_dos\_dormitorios. \\ quiere\_dos\_dormitorios \prec pareja\_joven, hijos. \\ quiere\_dos\_dormitorios \prec pareja\_joven, profesionales. \end{array} \right\}$$

Por ejemplo, la regla rebatible $living\_grande \prec pareja\_joven$ representa que: usualmente las parejas jóvenes prefieren departamentos con un estar amplio. Esta regla podría ser usada para construir argumentos para recomendar a una pareja joven un departamento en particular (*e.g.*, $depto3$). En cambio, si la recomendación es para una pareja joven con hijos, luego, la regla $quiere\_dos\_habitaciones \prec pareja\_joven, hijos$ puede ser usada para construir un argumento que recomiende un departamento con dos habitaciones (*e.g.*, $depto1$) y como normalmente los departamentos de dos habitaciones no tienen un estar grande ($\sim living\_grande \prec dos\_habitaciones$), se puede construir un argumento en contra de recomendar "$depto3$".

Por último, el programa incluye reglas que serán las encargadas de determinar cuando un departamento resulta de interés para un cliente. A partir de estas reglas, el intérprete

de programa lógico rebatible intentará construir razones a favor y en contra en base a los departamentos disponibles en el programa, al conocimiento del *SRI* y a las preferencias del cliente.

$$\mathcal{P} = \left\{ \begin{array}{l} recomendar(X) \prec disponible(X), concuerdan\_caract(X), concuerda\_ubicacion(X). \\ concuerdan\_caract(X) \prec quiere\_living\_grande, living\_grande(X). \\ concuerdan\_caract(X) \prec quiere\_dos\_dormitorios, dos\_dormitorios(X). \\ concuerdan\_ubicación(X) \prec \sim prefiere\_afueras(X), céntrico(X). \\ concuerdan\_ubicación(X) \prec prefiere\_afueras(X), afueras(X). \end{array} \right\}$$

Por lo tanto, es posible definir el *SRI* en base al programa introducido previamente. Dicho *SRI* será identificado con la terna $\langle \mathcal{I}_{delp}, \{+, *, -\}, \mathcal{P}_{deptos} \rangle$, incluyendo como programa almacenado al programa lógico rebatible $\mathcal{P}_{deptos}$, con la información de los departamentos y las reglas rebatibles necesarias para construir argumentos a favor y en contra de los distintos departamentos. Además, esta definición incluye los operadores provistos por el servicio de razonamiento para modificar el programa almacenado y el intérprete DeLP.

Los clientes interesados en obtener recomendaciones sobre departamentos consultarán al *SRI* y este utilizará el contexto para modificar el conocimiento público que tiene almacenado para que contenga información específica de las preferencias del cliente. En particular, el *SRI* asume ciertas características de los clientes, como por ejemplo, el hecho de que tengan *auto*. Por lo tanto, al momento de modificar el programa en el *SRI*, la información enviada por el cliente respecto de sus características (*e.g.*, $\sim auto, pareja\_joven, profesionales$) tendrá prioridad por sobre la asumida por el *SRI*. Además, puede darse la situación en la cual el cliente prefiera que la información asumida por el *SRI* no sea tenida en cuenta en la recomendación (por ejemplo, un cliente puede desear que las razones relativas a la posesión de un automóvil no sean tenidas en cuenta). En este caso, el operador restrictivo de programa lógico rebatible permite a los clientes quitar información del *SRI*. Otra alternativa es que el cliente envíe información extra respecto de algún departamento incluido en la consulta, como por ejemplo, que cree que *depto1* está ubicado cerca de una estación de subte ($subte(depto1)$). En este caso, la información almacenada en el *SRI* tendrá prioridad, debido a que generalmente resulta ser más confiable.

A continuación se muestra como sería el contexto lógico rebatible para un agente A1 representando una pareja joven de profesionales que quiere conocer la opinión del

SRI respecto de un departamento en particular. Teniendo en cuenta que dicha pareja prefiere que el departamento se encuentre alejado del centro solo en el caso que sea económico y que creen que está cerca de una estación de subte. Además, prefieren que la información relativa al automóvil no sea considerada debido a que están evaluando la posibilidad de comprar uno y, por lo tanto, no desean que dicha información sea considerada en la decisión del departamento a alquilar. En este caso en particular, el contexto lógico rebatible correspondiente sería: $\langle((\Pi_1, \Delta_1), +), ((\Pi_2, \Delta_2), *), ((\Pi_3, \Delta_3), -)\rangle$, siendo $\Pi_1 = \{pareja\_joven, profesionales\}, \Delta_1 = \{alejado(D) \prec económico(D)\}$, $\Pi_2 = \{subte(depto1)\}, \Delta_2 = \{\}$, $\Pi_3 = \{auto, \sim auto\}$ y $\Delta_3 = \{\}$.

Teniendo en cuenta estas preferencias y el hecho que la pareja de jóvenes profesionales está interesada en saber si el *SRI* recomienda el departamento 1, el agente A1 ejecutará la siguiente consulta contextual lógica rebatible, incluyendo el contexto lógico rebatible introducido previamente y la consulta correspondiente al departamento ($recomienda(depto1)$):

$$[\langle\ ((\Pi_1, \Delta_1), +), ((\Pi_2, \Delta_2), *), ((\Pi_3, \Delta_3), -)\ \rangle, recomienda(depto1)].$$

Al momento de responder esta *CC*, el *SRI* $\langle \mathcal{I}_{delp}, \{+, *, -\}, \mathcal{P}_{deptos}\rangle$ primero, actualizará el programa $\mathcal{P}_{deptos}$, agregando el programa (Π_1, Δ_1) otorgándole prioridad a la información enviada en el contexto, agregando el programa (Π_2, Δ_2), pero esta vez priorizando la información almacenada en el servicio de razonamiento, y finalmente quitando del programa resultante aquello que coincida con el programa (Π_3, Δ_3). Luego, calcula la respuesta a la *CC* enviada utilizando el intérprete de programación en lógica rebatible con el programa modificado. La respuesta obtenida es: $(SI, \{recomienda(depto1)\})$.

Por otro lado, un agente A2 representando a otra pareja de jóvenes, en este caso, con hijos y sin auto, podría realizar una nueva consulta al *SRI*. A diferencia del caso anterior, donde la consulta era por un departamento en particular, esta pareja de jóvenes puede estar interesada en saber si el *SRI* le recomienda alguno de los departamento que posee en alquiler. En este escenario, el agente A2 puede realizar la siguiente consulta: $[\langle\ ((\Pi_4, \Delta_4), +)\rangle, recomienda(Depto)]$, siendo $\Pi_4 = \{\sim auto, pareja\_joven, hijos\}$ y $\Delta_4 = \{\}$. En este caso, el *SRI*, luego de procesar la consulta, entregará la siguiente respuesta: $(SI, \{recomienda(depto1), recomienda(depto2)\})$.

En el ejemplo introducido en esta sección se mostraron distintos agentes consultando un *SR DeLP*. En particular, se presentaron dos agentes y las distintas *CCs* que estos

agentes podrían enviar a un *SRI* buscando una recomendación relativa a la información que posee dicho *SRI* sobre departamentos.

2.4. Conclusiones

En este capítulo se han introducido los conceptos de servicio de razonamiento y consulta contextual. Dichos conceptos representan uno de los principales aportes de esta tesis y tienen como objetivo permitir el desarrollo de servicios que representen conocimiento y respondan, de manera personalizada, consultas de diversos clientes.

Los servicios de razonamiento tienen la capacidad de representar conocimiento y permiten realizar modificaciones temporales a dicho conocimiento para responder consultas de clientes. Para permitir modificaciones sobre su conocimiento, los servicios de razonamiento proveen operadores específicos, mientras que para responder las consultas poseen un intérprete que computa las respuestas.

Los clientes que deseen consultar este tipo de servicios utilizarán las consultas contextuales. Una consulta contextual posee un contexto que será usado al momento de computar la respuesta para modificar el conocimiento en el servicio de razonamiento. Dicho contexto incluye conocimiento propio de los clientes a ser utilizado en el cálculo de la respuesta por parte del servicio de razonamiento.

Como se mencionó previamente, un *SR* puede recibir consultas de varios clientes simultáneamente. La forma en la cual un *SR* procesa una CC, modificando temporalmente el conocimiento público a partir del contexto de la consulta, posibilita que un *SR* pueda resolver *CCs* distintas simultáneamente sin que se vean afectadas las respuestas obtenidas por los clientes.

En esta tesis, las consultas contextuales realizarán modificaciones *temporales* al conocimiento público del *SR*. Esta decisión, se basa, fundamentalmente en que corresponde a la semántica esperada de acuerdo a su denominación, es decir, el término *consulta* no connota modificación en el conocimiento o la información de quien es consultado.

En cambio, una semántica diferente para las consultas contextuales, en la cual las modificaciones realizadas por el contexto de las mismas fuera *permanente*, generarían ciertos inconvenientes dificultando el procesamiento concurrente de las mismas. Es decir, consultas contextuales cuyas modificaciones en el conocimiento público del *SR* no

sean temporales sino permanentes, implicaría en un principio, la necesidad de un control de concurrencia, donde claramente, el orden en que se ejecutan las consultas importa. Además, dichas consultas contextuales especiales podrían estar asociadas a privilegios específicos de ciertos clientes con las capacidades de ejecutarlas.

Como ejemplos de aplicación, se introdujeron dos tipos específicos de servicios de razonamiento, utilizando programación lógica y la programación lógica rebatible respectivamente, junto con ejemplos concretos de posibles dominios de aplicación. Estos servicios de razonamiento específicos representan alternativas concretas de servicios de razonamiento y serán utilizados a lo largo de esta tesis para ejemplificar los conceptos introducidos. De todas formas, es posible definir más servicios de razonamiento utilizando otros formalismos que permitan la representación de conocimiento y respondan consultas.

Capítulo 3

Consultas Contextuales Extendidas: Eficiencia y Distribución de carga

Las Consultas Contextuales (*CCs*) definidas en el capítulo anterior permiten a un cliente realizar distintas consultas a uno o más Servicios de Razonamiento (*SRs*) específicos. En un escenario donde existan gran cantidad de clientes y *SRs*, y el nivel de interacción entre ellos sea alto, se requerirá del intercambio de grandes cantidades de información entre clientes y *SRs*. En este capítulo se definirán nuevos tipos de consultas contextuales que ofrecen distintas alternativas respecto de la forma en la cuál un cliente puede consultar a un *SR*. El objetivo de definir estas nuevas alternativas es lograr eficiencia en el cálculo de las respuestas buscando mejorar los siguientes aspectos:

- la sobrecarga en el intercambio de mensajes,
- las actualizaciones del conocimiento en el *SR* y
- el procesamiento secuencial.

De esta manera, las *CCs* que se formalizarán y analizarán a lo largo del capítulo, buscan mejorar la eficiencia del sistema, sin alterar la semántica de las *CCs*. Dicha mejora en la eficiencia se obtiene a partir de la *disminución* de la cantidad de mensajes intercambiados entre clientes y *SRs* y el volumen de modificaciones necesarias por parte del *SR* para responder las consultas Estas alternativas permiten, además, facilitar la *distribución del cómputo* de las respuestas en distintos *SRs*, representando una cualidad que será explorada en detalle en el próximo capítulo.

En primer lugar, se definirá una *CC múltiple* (*CCM*) donde un mismo cliente podrá enviar en un solo mensaje varias consultas individuales y recibir las respuestas a cada una de ellas en un único mensaje. Como se verá, esto permite mejorar la sobrecarga sin afectar la semántica de las *CCs* y ofrecerá una oportunidad de paralelismo.

En segundo lugar, se definirán las *CCs múltiples factorizadas* (*CCMF*), las cuales corresponden a un caso particular de las consultas contextuales múltiples donde se repite el contexto de las consultas. Por lo tanto, este nuevo tipo de consulta permite minimizar el tamaño de la consulta enviada y, a su vez, las modificaciones necesarias en los servicios de razonamiento para resolver las consultas.

Por último, se definirán las *interrogaciones contextuales* (*IC*), las cuales buscan obtener las ventajas de las *CCMs* y las *CCMFs* de una manera más general, de forma tal de permitir su aprovechamiento en cualquier situación. De esta manera, el cliente podrá optar por *CCs* simples, por *CCMs*, por *CCMFs* o por *ICs*. Luego, en la sección 3.5 se mostrará un análisis comparativo y se indicarán las propiedades que relacionan a estos tipos de consultas.

3.1. Configuraciones con Múltiples Clientes y *SRs*

Las *CCs* definidas en el capítulo anterior brindan un mecanismo de interacción entre un cliente y un *SR*. La Figura 3.1 muestra las posibles configuraciones que surgen ante la existencia de múltiples clientes y múltiples *SRs*. Luego, en el capítulo 4 se analizarán otras configuraciones más complejas y se definirá el concepto de grupo.

La configuración más simple (ver Figura 3.1 (a)), está asociada a un único cliente y un único *SR*. La segunda configuración posible, mostrada en la Figura 3.1 (b), corresponde a la existencia de un único cliente y múltiples *SRs*. En cualquiera de estas dos configuraciones, el cliente puede necesitar realizar múltiples *CCs*, ya sea a más de un *SR* simultáneamente o, incluso, a un mismo *SR*, dando lugar a nuevos desafíos que serán analizados en este capítulo.

Por otro lado, un mismo *SR* puede recibir *CCs* de varios clientes simultáneamente, dando lugar a la tercer configuración posible, mostrada en la Figura 3.1 (c) y finalmente, la última configuración posible corresponde a una combinación de las dos configuraciones mencionadas previamente, es decir, la existencia de múltiples clientes ejecutando consultas

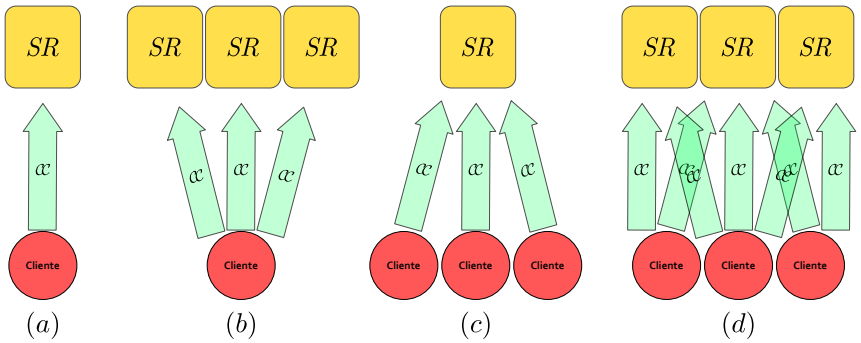

Figura 3.1: Distintas Configuraciones de *SRs* y Clientes

en diversos *SRs*. En el capítulo anterior se explicaron las razones por las cuales un mismo *SR* puede procesar simultáneamente múltiples *CCs*.

Utilizando las *CCs*, un cliente puede ejecutar de a una consulta por contexto. Por lo tanto, si el cliente quisiera ejecutar más de una consulta para un mismo contexto o múltiples consultas con distintos contextos, la única alternativa, hasta el momento, corresponde a enviar múltiples *CCs* de a una por vez. Sin embargo, este esquema de interacción entre el cliente y el *SR* tiene ciertas desventajas comparado con la posibilidad de enviar una única consulta que incluya todas las *CCs* requeridas.

Por un lado, el costo en el uso de la infraestructura de mensajes es menor si se envía una única consulta, en lugar de realizar numerosos intercambios de mensajes más pequeños. Esto está asociado directamente con el tiempo requerido desde que un mensaje es enviado por el cliente hasta que llega al *SR*, independientemente de la velocidad de transferencia con la que pueda lograrse dicho envío.

Por otro lado, desde el punto de vista del *SR* que responde las *CCs* del cliente, la posibilidad de recibir todas las *CCs* agrupadas en una única consulta le permite un procesamiento más eficiente de las mismas. Es decir, al recibir todas las consultas juntas, el *SR* tiene la posibilidad de minimizar las modificaciones en el conocimiento público para responder las *CCs*. Además, permite al *SR* una implementación paralela o incluso la distribución del cómputo de las respuestas entre distintos *SRs*.

A continuación, se introducirán nuevos tipos de consultas que permitirán al cliente agrupar sus consultas en una única interacción con el objetivo de optimizar el uso de

la infraestructura de mensajes y mejorar la eficiencia de los *SRs* en el cómputo de las respuestas. Luego, en el Capítulo 4 se definirá el concepto de grupo de razonamiento, que permite paralelizar el cómputo de las respuestas.

3.2. Consulta Contextual Múltiple

En esta sección se introduce un nuevo tipo de consulta que agrupa una secuencia de *CCs*. Si bien la semántica de esta nueva consulta será igual a la ejecución de las distintas consultas individualmente, la ventaja de este nuevo tipo de consulta es que permite mayor declaratividad y una implementación más eficiente, en cuanto al uso de la infraestructura de mensajes y al aprovechamiento de los procesadores existentes.

La *Consulta Contextual Múltiple* (CCM) será definida como una versión extendida de la CC, mientras que la respuesta para este tipo de consultas será una secuencia de respuestas, cada una de ellas correspondiendo a una CC de la CCM. En la Figura 3.2 se muestra en forma esquemática los elementos involucrados en el cálculo de la respuesta para este nuevo tipo de consulta. En esta figura en particular, se muestra como es el cálculo de la segunda CC incluida en la CCM. Las flechas muestran como son utilizados los datos por el *SR*. Por ejemplo, el contexto $\mathcal{C}o_2$ es utilizado para modificar el programa $\mathcal{P}$, utilizando los operadores del *SR*, generando un nuevo programa temporal $\mathcal{P} \diamond \mathcal{C}o_2$. Luego, a partir de este programa y la consulta Q_2, el intérprete $\mathcal{I}$ obtiene la respuesta $\mathcal{A}ns_2$. Los nuevos conceptos son definidos a continuación:

Definición 3.1 (Consulta Contextual Múltiple (CCM))**.** *Una* consulta contextual múltiple *en un marco de representación de conocimiento $\mathcal{M}$ es una secuencia de pares $\langle[\mathcal{C}o_1, Q_1], [\mathcal{C}o_2, Q_2], \ldots, [\mathcal{C}o_n, Q_n]\rangle$ donde cada $[\mathcal{C}o_i, Q_i]$ es una consulta contextual en $\mathcal{M}$. La consulta contextual múltiple será* adecuada *para un servicio de razonamiento $\mathcal{SR}$ en el marco $\mathcal{M}$, si cada consulta contextual $[\mathcal{C}o_i, Q_i]$ $(1 \leq i \leq n)$ es adecuada para $\mathcal{SR}$.*

Definición 3.2 (Respuesta para una CCM)**.** *Sea $\mathcal{CCM} = \langle[\mathcal{C}o_1, Q_1], [\mathcal{C}o_2, Q_2], \ldots, [\mathcal{C}o_n, Q_n]\rangle$ una consulta contextual múltiple para el servicio de razonamiento $\mathcal{SR}$ en el marco de representación de conocimiento $\mathcal{M}$. La respuesta para $\mathcal{CCM}$ en $\mathcal{SR}$ corresponde a la secuencia $\langle\mathcal{A}ns_1, \mathcal{A}ns_2, \ldots, \mathcal{A}ns_n\rangle$, donde cada $\mathcal{A}ns_i$ $(1 \leq i \leq n)$ es la respuesta para la correspondiente consulta contextual $[\mathcal{C}o_i, Q_i]$ en $\mathcal{SR}$.*

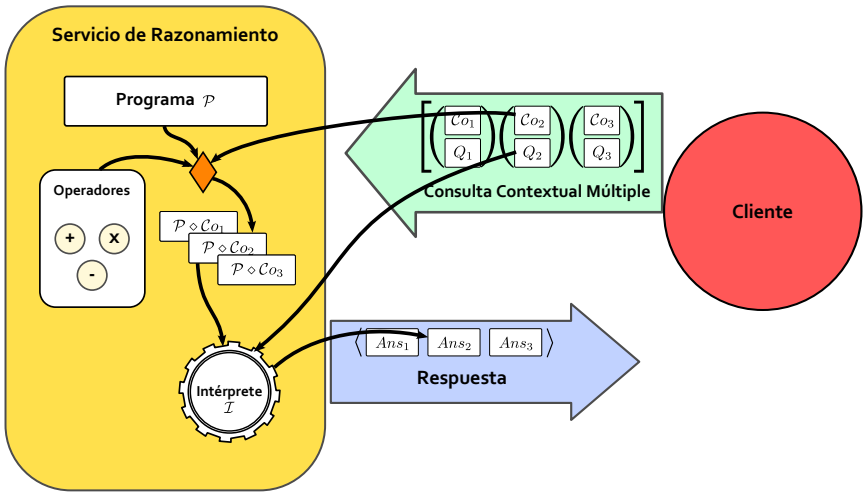

Figura 3.2: Respuesta para una Consulta Contextual Múltiple

Este nuevo tipo de consulta permite agrupar en un único mensaje múltiples *CCs*, minimizando, de esta forma, el uso de la infraestructura de mensajes y permitiendo, a la vez, un mayor control por parte del *SR* a la hora de procesar dicha consulta. Es importante mencionar que el orden en el cual se resuelven las consultas en el *SR* no afecta la respuesta obtenida. En la Figura 3.3 se muestra una *CCM* incluyendo 3 *CCs*. La flecha indica cuál es el alcance de las modificaciones del contexto en el conocimiento público para las consultas. Es decir, el contexto $\mathcal{C}o_1$ afecta únicamente a la consulta Q_1, el contexto $\mathcal{C}o_2$ afecta solamente a la consulta Q_2 y finalmente, el contexto $\mathcal{C}o_3$ afecta a la consulta Q_3. Por lo tanto, las consultas son independientes entre sí, debido a que, como se muestra en la Figura 3.3, el alcance de las modificaciones realizadas en el programa almacenado en el *SR* para considerar cada uno de los contextos sólo afecta a la consulta correspondiente. Por lo tanto, como se mostrará en el capítulo 4, cada *CC* individual incluida en una *CCM* podría ser resuelta en paralelo.

Ejemplo 3.1. *Considere nuevamente el Servicio de Razonamiento* $\mathcal{SR}_{tr} = \langle \mathcal{I}_{lp}, \{\oplus\}, \mathcal{P}_{tr} \rangle$, *introducido previamente en el Ejemplo 2.1, cuyo principal objetivo es sugerir tratamientos para enfermedades de acuerdo a los síntomas de los pacientes. El conocimiento público de* $\mathcal{SR}_{tr}$ *está representado por el programa* $\mathcal{P}_{tr}$ *incluido a continuación.*

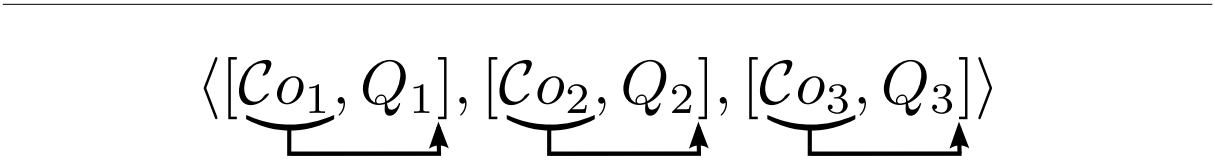

Figura 3.3: Alcance de los Contextos en una Consulta Contextual Múltiple

Considere la existencia de tres pacientes que poseen distintos síntomas. Uno de los pacientes posee los síntomas s1 *y* s2*, otro los síntomas* s1 *y* s4*, mientras que el tercero, posee los síntomas* s2 *y* s4*. Un cliente desea consultar al* $\mathcal{SR}_{tr}$ *para conocer a qué enfermedades están asociados los síntomas de cada paciente.*

$$\mathcal{P}_{tr} = \left\{ \begin{array}{l} \textit{enfermedad(e1) :- síntoma(s1), síntoma(s2), síntoma(s3).} \\ \textit{enfermedad(e2) :- síntoma(s1), síntoma(s4).} \\ \textit{enfermedad(e3) :- síntoma(s2), síntoma(s4).} \\ \textit{tratamiento(t1) :- enfermedad(e1).} \\ \textit{tratamiento(t2) :- enfermedad(e2), not(alérgico(t2)).} \\ \textit{tratamiento(t3) :- enfermedad(e2), not(enfermedad(e3)).} \\ \textit{tratamiento(t4) :- enfermedad(e2), enfermedad(e3).} \\ \textit{tratamiento(t5) :- enfermedad(e3), not(enfermedad(e2)).} \end{array} \right\}$$

Por lo tanto, se podría enviar al servicio de razonamiento $\mathcal{SR}_{tr}$ *la CCM* $\langle[\mathcal{C}o_{lp1}, Q_{lp1}], [\mathcal{C}o_{lp2}, Q_{lp2}], [\mathcal{C}o_{lp3}, Q_{lp3}]\rangle$*, siendo:*

$$\begin{array}{ll} \mathcal{C}o_{lp1} = \langle(\{sintoma(s1), sintoma(s2)\}, \oplus)\rangle & Q_{lp1} = enfermedad(E) \\ \mathcal{C}o_{lp2} = \langle(\{sintoma(s1), sintoma(s4)\}, \oplus)\rangle & Q_{lp2} = enfermedad(E) \\ \mathcal{C}o_{lp3} = \langle(\{sintoma(s2), sintoma(s4)\}, \oplus)\rangle & Q_{lp3} = enfermedad(E) \end{array}$$

La respuesta otorgada por el SR correspondiente es: $\langle(NO, \{\}),$ $(SI, \{enfermedad(e2)\}),$ $(SI, \{enfermedad(e3)\})\rangle$.
Esta respuesta indica que los síntomas s1 *y* s2 *no están asociados por sí solos a ninguna enfermedad, mientras que los síntomas* s1 *y* s4 *están asociados a la enfermedad* e2 *y los síntomas* s2 *y* s4 *están asociados a la enfermedad* e3.

□

Al final de este capítulo se presentarán más ejemplos de este nuevo tipo de consulta, ampliando los dominios de aplicación introducidos en el capítulo anterior relacionados principalmente con el Servicio de Recomendación de Noticias y el Servicio de Recomendación de Inmuebles.

3.3. Consulta Contextual Múltiple Factorizada

En el caso particular de un cliente que necesita ejecutar varias *CCs* con el mismo contexto, éste puede simplemente ejecutar varias *CCs*, o bien, ejecutar una *CCM* repitiendo el contexto. Cabe aclarar que ambas alternativas requieren de las mismas modificaciones sobre el programa almacenado en el *SR*, repetidas veces. Por lo tanto, se definirá un nuevo tipo de consulta, denominada *Consulta Contextual Múltiple Factorizada* (*CCMF*), como un caso particular de la *CCM*, consistiendo de una secuencia de consultas con un único contexto:

Definición 3.3 (Consulta Contextual Múltiple Factorizada (*CCMF*))**.** *Sea* $\mathcal{D}c$ *un dominio de consultas en el marco de representación de conocimiento* $\mathcal{M}$*. Una* consulta contextual múltiple factorizada *en* $\mathcal{M}$ *es un par* $[\mathcal{C}o, Q_s]$ *donde* $\mathcal{C}o$ *es un contexto en* $\mathcal{M}$ *y* Q_s *es una secuencia* $\langle Q_1, Q_2, \ldots, Q_n \rangle$ *siendo cada* Q_i $(1 \leq i \leq n)$ *una consulta perteneciente al dominio* $\mathcal{D}c$*. La consulta contextual múltiple factorizada* $[\mathcal{C}o, Q_s]$ *será* adecuada *para un servicio de razonamiento* $\mathcal{SR}$ *en el marco* $\mathcal{M}$*, si el contexto* $\mathcal{C}o$ *es adecuado para* $\mathcal{SR}$*.*

En este nuevo tipo de consulta, las modificaciones necesarias para considerar el contexto podrían realizarse una única vez en el programa almacenado en el *SR*, para luego resolver todas las consultas de la secuencia. Por lo tanto, como se muestra en la Figura 3.4, el alcance de las modificaciones realizadas por el contexto incluye a todas las consultas dentro de la secuencia. Esta nueva definición permite una implementación más eficiente de un *SR* respondiendo consultas con igual contexto y evita la sobrecarga innecesaria de la infraestructura de intercambio de mensajes, así como también una modificación reiterativa del *SR*.

La respuesta para una *CCMF* incluirá una secuencia de respuestas, cada una de ellas correspondiendo a una consulta incluida en la *CCMF*. Como se muestra de manera esquemática en la Figura 3.5, el programa almacenado es modificado por el contexto de la consulta utilizando los operadores correspondientes y a partir del nuevo programa, se

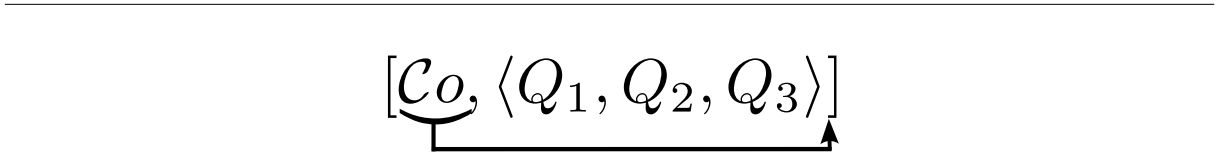

Figura 3.4: Alcance de los Contextos en una Consulta Contextual Múltiple Factorizada

computan las respuestas para las distintas consultas. Por ejemplo, las flechas de la figura indican que a partir del programa modificado y la consulta Q_2, el intérprete computa la respuesta $\mathcal{A}ns_2$. A continuación de introduce la definición correspondiente:

Definición 3.4 (Respuesta para una *CCMF*)**.** *Sea* $\mathcal{CCMF} = [\mathcal{C}o, \langle Q_1, Q_2, \ldots, Q_n \rangle]$ *una consulta contextual múltiple factorizada para el servicio de razonamiento* $\mathcal{SR}$ *en el marco de representación de conocimiento* $\mathcal{M}$*. La respuesta para* $\mathcal{CCMF}$ *en* $\mathcal{SR}$ *es la secuencia* $\langle \mathcal{A}ns_1, \mathcal{A}ns_2, \ldots, \mathcal{A}ns_n \rangle$*, donde cada* $\mathcal{A}ns_i$ $(1 \leq i \leq n)$ *es la respuesta para la consulta contextual* $[\mathcal{C}o, Q_i]$ *en* $\mathcal{SR}$*.*

Ejemplo 3.2. *Un paciente con los síntomas* s1*,* s2 *y* s4 *y alérgico al tratamiento* t1 *podría desear conocer cuál o cuáles son las posible enfermedades que posee así como también los tratamientos disponibles que le puede sugerir el SR. En este caso, dicho paciente podría enviar la CCMF* $[\mathcal{C}o_{lp1}, \langle enfermedad(E), tratamiento(T) \rangle]$*, siendo:*

$$\mathcal{C}o_{lp1} = \{sintoma(s1), sintoma(s2), sintoma(s4), alérgico(t1)\}.$$

La respuesta que brinda el SR es la siguiente:

$$\langle (SI, \{enfermedad(e2), enfermedad(e3)\}), (SI, \{tratamiento(t4)\}) \rangle.$$

□

En las secciones 3.6 y 3.7 se incluyen más ejemplos que muestran el uso de este nuevo tipo de consulta en los dominios de aplicación introducidos previamente en esta tesis. Dependiendo de la implementación, la secuencia de respuestas generada por el *SR* puede ser almacenado temporalmente en el propio *SR* y luego enviado al cliente en un único mensaje, o bien, enviar la respuesta a cada consulta ni bien es obtenida, requiriendo el

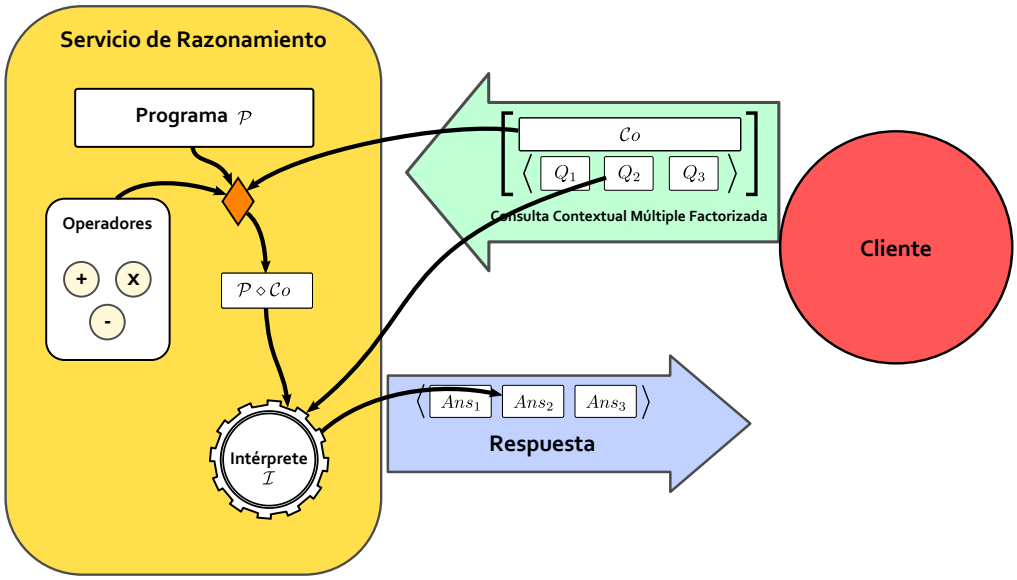

Figura 3.5: Respuesta para una Consulta Contextual Múltiple Factorizada

intercambio de un mensaje por consulta. Por lo tanto, existe una relación de compromiso entre el número de mensajes intercambiados y el tiempo de respuesta.

Los nuevos tipos de consultas introducidos buscan lograr eficiencia en el cálculo de las respuestas. En particular, las *CCMs* permiten minimizar la cantidad de mensajes intercambiados, mientras que las *CCMFs* permiten, además, disminuir las modificaciones requeridas por parte del *SR*. Sin embargo, existen situaciones que no están contempladas por estos tipos de consultas y donde es posible obtener un mejor desempeño.

En la siguiente sección se introducirán nuevos tipos de consultas contextuales con el objetivo de mejorar la eficiencia en el cálculo de las respuestas en dichas situaciones. Luego, en la sección 3.5 se muestra cómo se relacionan los nuevos tipos de consultas definidas tanto en esta sección como en la que viene y las *CCs*, cuya definición fue introducida en el capítulo anterior.

3.4. Interrogación Contextual

Un cliente puede necesitar ejecutar diversas consultas con diferentes contextos. Por ejemplo, es posible que un cliente desee realizar las siguientes *CCs*: $[\langle(\mathcal{P}_1, O_1)(\mathcal{P}_2, O_2)(\mathcal{P}_3, O_3)\rangle, Q_a]$ y $[\langle(\mathcal{P}_1, O_1)(\mathcal{P}_2, O_2)(\mathcal{P}_3, O_3)(\mathcal{P}_4, O_4)\rangle, Q_b]$. Con el objetivo de reducir el intercambio de mensajes, el cliente puede utilizar una *CCM* $\langle[\mathcal{C}o_a, Q_a], [\mathcal{C}o_b, Q_b]\rangle$, siendo $\mathcal{C}o_a$ y $\mathcal{C}o_b$ los contextos anteriores. Sin embargo, como los contextos de las consultas se superponen casi enteramente, esta alternativa repite gran parte de los programas y las modificaciones necesarias en el *SR*. La opción de utilizar una *CCMF* no es viable, debido a que no son exactamente los mismos contextos.

Se introducirá entonces un nuevo tipo de consulta, denominada *Interrogación Contextual* (*IC*), similar a la *CCM* en el hecho de que estará formada por una secuencia de pares $(contexto, programa)$, los contextos modificarán al *SR* para luego responder las consultas, obtendrá una secuencia de respuestas y no realizará modificaciones permanentes al programa almacenado. La principal diferencia corresponde al alcance de las modificaciones realizadas por los contextos. Estas modificaciones, como se muestra en la Figura 3.6 (c), afectarán no sólo la consulta actual sino que también a las posteriores consultas de la misma *IC*.

A continuación se introduce la definición del concepto de *Interrogación Contextual*, donde, como se mencionó previamente, las modificaciones realizadas por el contexto de las consultas en el *SR* serán mantenidas para la resolución de consultas subsiguientes de la misma *IC*. Utilizando este nuevo tipo de consulta, en el ejemplo previo, los programas y operadores del contexto $\mathcal{C}o_a$ podrán ser enviados una sola vez junto con la consulta Q_a, mientras que el programa $\mathcal{P}_4$ y su operador correspondiente podrá ser enviado junto con la consulta Q_b sin la necesidad de repetir el contexto anterior, ni sus modificaciones en el *SR*, permitiendo, de esta forma, un uso eficiente tanto de la infraestructura de mensajes así como también del tiempo de cómputo en el *SR*. La *IC* es definida como una secuencia de consultas contextuales especiales.

Definición 3.5 (Interrogación Contextual (*IC*))**.** *Sea $\mathcal{D}c$ un dominio de consultas en el marco de representación de conocimiento $\mathcal{M}$. Una* interrogación contextual *en $\mathcal{M}$ es una secuencia $[\mathcal{C}c_1, \mathcal{C}c_2, \ldots, \mathcal{C}c_n]$ donde cada $\mathcal{C}c_i$ $(1 \leq i \leq n)$ es un par $(\mathcal{C}o_i, Q_i)$ representando lo que será denominado Consulta Contextual Continuada en $\mathcal{M}$, siendo $\mathcal{C}o_i$ un contexto en $\mathcal{M}$ y Q_i una consulta del dominio $\mathcal{D}c$. La interrogación contextual será* adecuada *para*

un servicio de razonamiento $\mathcal{SR}$ en $\mathcal{M}$, si cada contexto $\mathcal{C}o_i$ de $(\mathcal{C}o_i, Q_i)$ es adecuado para $\mathcal{SR}$.

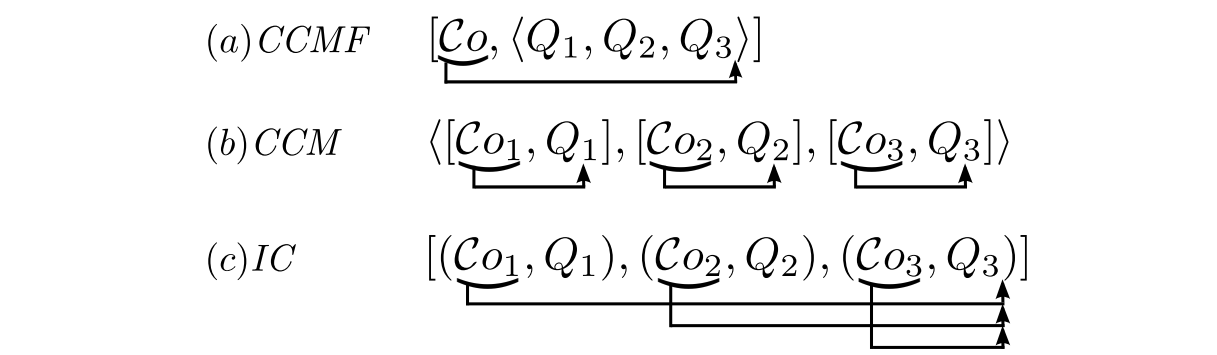

Figura 3.6: Alcance de los Contextos en una *CCMF*, una *CCM* y una *IC*

Observación 3.1. *Con el objetivo de distinguir sintácticamente una Consulta Contextual Continuada de una CC, se utilizan paréntesis ('(' y ')') en vez de corchetes ('[' y ']')*

Es importante notar que el par $(\mathcal{C}o_i, Q_i)$ incluido en una *IC*, introducido en la definición anterior, posee algunas diferencias con respecto al par $[\mathcal{C}o_i, Q_i]$ incluido en una *CCM*. En la Figura 3.6 se puede observar el alcance de los contextos de las *CCMFs* (a), de las *CCMs* (b) y de las *ICs* (c). En una *IC*, las modificaciones realizadas por un contexto de la *IC* en el programa almacenado en el *SR* no sólo afectan a la resolución de la consulta asociada, sino que además, perduran influyendo en el cálculo de las respuestas de las próximas **consultas de la misma *IC***.

La *IC* puede ser considerada como una *CC* extendida donde está permitido intercalar consultas entre las modificaciones llevadas a cabo por el contexto. Es decir, no sólo es posible modificar, a través del contexto, el programa almacenado en el *SR* y ejecutar una consulta, sino que además es posible seguir modificando el nuevo programa generado y consultándolo reiteradamente. Es por esta razón que fue denominada *Interrogación* Contextual.

De acuerdo a la descripción de la *IC*, una respuesta para una consulta de este tipo, como se puede ver en la Figura 3.7, será una secuencia de respuestas, una para cada

consulta, calculada utilizando tanto el contexto dado con la consulta, como el contexto de las consultas previas de la misma IC. A continuación se introducirá la noción de concatenación de contextos con el objetivo de facilitar la posterior definición de la respuesta para la IC. La noción de concatenación de contextos, como indica su nombre, concatenará dos contextos, es decir, dos secuencias de pares de programa y operador.

Definición 3.6 (Concatenación de Contextos)**.** *Sean* $\mathcal{C}o_A = \langle(\mathcal{P}_{A1}, O_{A1}),\ (\mathcal{P}_{A2}, O_{A2}),\ \ldots(\mathcal{P}_{An}, O_{An})\rangle$ *y* $\mathcal{C}o_B = \langle(\mathcal{P}_{B1}, O_{B1}), (\mathcal{P}_{B2}, O_{B2}), \ldots, (\mathcal{P}_{Bn}, O_{Bn})\rangle$ *dos contextos. La concatenación de dos contextos* $\mathcal{C}o_A$ *y* $\mathcal{C}o_B$*, denotado* $\mathcal{C}o_A \square \mathcal{C}o_B$*, será el resultado de concatenar ambas secuencias de pares (programa, operador). Es decir,* $\mathcal{C}o_A \square \mathcal{C}o_B = \langle(\mathcal{P}_{A1}, O_{A1}), (\mathcal{P}_{A2}, O_{A2}), \ldots, (\mathcal{P}_{An}, O_{An}),\ (\mathcal{P}_{B1}, O_{B1}),\ (\mathcal{P}_{B2}, O_{B2}), \ldots, (\mathcal{P}_{Bn}, O_{Bn})\rangle$

Definición 3.7 (Respuesta para una IC)**.** *Sea* $\mathcal{IC} = [(\mathcal{C}o_1, Q_1), (\mathcal{C}o_2, Q_2), \ldots,\ (\mathcal{C}o_n, Q_n)]$ *una interrogación contextual para el servicio de razonamiento* $\mathcal{SR}$ *en el marco de representación de conocimiento* $\mathcal{M}$*. La respuesta para* $\mathcal{IC}$ *en* $\mathcal{SR}$ *corresponde a la secuencia de respuestas* $\langle \mathcal{A}ns_1,\ \mathcal{A}ns_2, \ldots, \mathcal{A}ns_n\rangle$*, donde cada* $\mathcal{A}ns_i$ $(1 \leq i \leq n)$ *es la respuesta a la consulta contextual* $[\mathcal{C}o_1 \square \mathcal{C}o_2 \square \ldots \square \mathcal{C}o_i, Q_i]$ *en* $\mathcal{SR}$*.*

Esta definición formal de la respuesta para una IC traduce cada una de las Consultas Contextuales Continuadas incluidas en la IC, a *CCs* utilizando la concatenación de contextos. Esto, claramente, corresponde a una definición declarativa de la respuesta, sin que esto implique que las modificaciones en el conocimiento público del *SR* realizadas para la resolución de una Consulta Contextual Continuada no puedan aprovecharse en la resolución de las restantes consultas de la misma IC. En la implementación más natural de este tipo de consulta, mostrada en la Figura 3.7, las modificaciones en el conocimiento público del *SR* para resolver la primer Consulta Contextual Continuada son aprovechadas al momento de resolver la segunda consulta de este mismo tipo.

Ejemplo 3.3. *Continuando con el caso del paciente introducido en el Ejemplo 3.3, dicho paciente, con los síntomas* s1*,* s2 *y* s4 *y alérgico al tratamiento* t1*, podría estar interesado en conocer qué tratamientos existen para su condición. En particular, puede estar interesado en conocer cuáles son los tratamientos si se consideran sólo los síntomas* s1 *y* s4 *por un lado, los síntomas* s2 *y* s4 *por otro y, finalmente, todos los síntomas juntos.*

En este escenario, el paciente puede ejecutar la IC $[(\mathcal{C}o_{lp1}, Q_{lp1}),\ (\mathcal{C}o_{lp2}, Q_{lp2}),\ (\mathcal{C}o_{lp3}, Q_{lp3})]$*, siendo:*

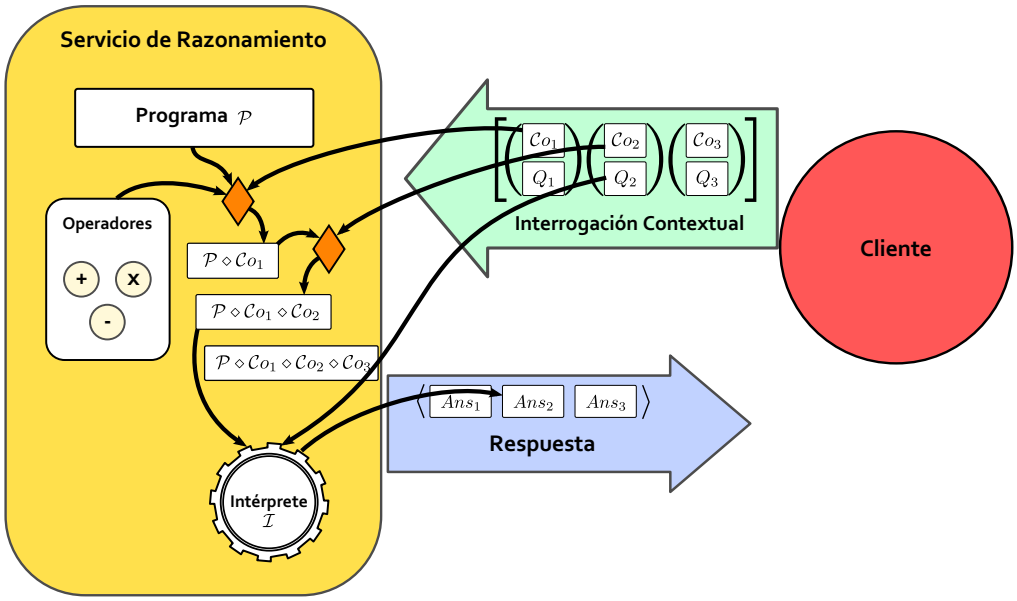

Figura 3.7: Respuesta para una Interrogación Contextual

$$Q_{lp1} = tratamiento(Tr) \quad \mathcal{C}o_{lp1} = \langle(\{sintoma(s1), sintoma(s4), alergico(t1)\}, \oplus)\rangle$$
$$Q_{lp2} = tratamiento(Tr) \quad \mathcal{C}o_{lp2} = \langle(\{sintoma(s2)\}, \oplus), (\{sintoma(s1)\}, \ominus)\rangle$$
$$Q_{lp3} = tratamiento(Tr) \quad \mathcal{C}o_{lp3} = \langle(\{sintoma(s1)\}, \oplus)\rangle$$

En este caso, para responder esta IC se utilizará una versión extendida del SR introducido originalmente en el Ejemplo 2.1. Esta versión extendida incluye el operador $\ominus$ *para poder quitar, del conocimiento temporal utilizado por el SR para responder consultas, síntomas introducidos en la misma IC y se define como* $\langle \mathcal{I}_{lp}, \{\oplus, \ominus\}, \mathcal{P}_{tr}\rangle$*. La respuesta otorgada por este SR extendido es:*

$$\langle(SI, \{tratamiento(t3)\}), (SI, \{tratamiento(t5)\}), (SI, \{tratamiento(t4)\})\rangle.$$

Esta respuesta indica que para los síntomas s1 *y* s4*, el SR extendido recomienda el tratamiento* t3*, mientras que para los síntomas* s2 *y* s4 *recomienda* t5 *y para todo los síntomas recomienda* t4*.*

En general, una IC en el marco de la programación lógica estará formada por una secuencia $[\mathcal{C}c_{lp1}, \mathcal{C}c_{lp2}, \ldots, \mathcal{C}c_{lpn}]$ *donde cada* $\mathcal{C}c_{lpi}$ $(1 \leq i \leq n)$ *es un par* $(\mathcal{C}o_{lpi}, Q_{lpi})$ *siendo*

$\mathcal{C}o_{lpi}$ un contexto lógico y Q_{lpi} una consulta del dominio de consulta $\mathcal{D}_{lq}$. La respuesta para una consulta de este tipo será calculada de la misma forma que para una IC, salvo por el intérprete utilizado, el cuál, en este caso, corresponderá al intérprete de programación lógica. □

Las *ICs*, como se demostrará en la sección 3.5, son un caso particular de las *CCMs* y, a su vez, las *CCMFs* son un caso particular de las *ICs*. Esto se debe a que las *ICs* permiten no sólo modificar, a través del contexto, el programa almacenado en el *SR* y ejecutar una consulta, sino que además, posibilitan seguir modificando el nuevo programa generado y consultándolo reiteradamente.

Una *IC* puede ser extendida para permitir intercalar consultas contextuales que toman como base el programa modificado por las consultas anteriores y consultas que utilizarán el programa almacenado originalmente en el *SR* para ser resueltas. Este nuevo tipo de consulta será denominada *Interrogación Contextual Múltiple*.

Por lo tanto, la *Interrogación Contextual Múltiple* (*ICM*) será definida como una versión extendida de la *IC*, formada por una secuencia de *ICs*. Cada *IC* utilizará para el cálculo de las respuestas el programa del *SR* de manera independiente. Es decir, las modificaciones realizadas por las consultas de una *IC* no afectará a las consultas incluidas en otras *ICs* de la *ICM*.

A continuación se introduce la definición de este nuevo tipo de consulta, así como también la definición de la respuesta, la cuál será una secuencia de respuestas, cada una de ellas correspondiendo a una *IC* de la *ICM*:

Definición 3.8 (Interrogación Contextual Múltiple (*ICM*))**.** *Una* interrogación contextual múltiple *en un marco de representación de conocimiento $\mathcal{M}$ es una secuencia $\langle \mathcal{IC}_1, \mathcal{IC}_2, \ldots, \mathcal{IC}_n \rangle$ donde cada $\mathcal{IC}_i$ es una interrogación contextual en $\mathcal{M}$. La interrogación contextual múltiple será* adecuada *para un servicio de razonamiento $\mathcal{SR}$ en el marco $\mathcal{M}$, si cada interrogación contextual $\mathcal{IC}_i$ es adecuada para $\mathcal{SR}$.*

Definición 3.9 (Respuesta para una *ICM*)**.** *Sea $\mathcal{ICM} = \langle \mathcal{IC}_1, \mathcal{IC}_2, \ldots, \mathcal{IC}_n \rangle$ una interrogación contextual múltiple para el servicio de razonamiento $\mathcal{SR}$ en el marco de representación de conocimiento $\mathcal{M}$. La respuesta para $\mathcal{ICM}$ en $\mathcal{SR}$ corresponde a la secuencia $\langle \mathcal{A}ns_1, \mathcal{A}ns_2, \ldots, \mathcal{A}ns_n \rangle$, donde cada $\mathcal{A}ns_i, 1 \leq i \leq n$, es la respuesta para la correspondiente interrogación contextual $\mathcal{IC}_i$ de la secuencia en $\mathcal{SR}$.*

3.5. Análisis y Propiedades

El Cuadro 3.1 resume las principales ventajas y características de las distintas consultas contextuales definidas en esta tesis. Para cada tipo de consulta contextual se indica si minimiza la interacción entre el cliente y el *SR*, si minimiza las modificaciones necesarias en el *SR*, si facilita la distribución de la carga por parte del *SR*, si permite la utilización de contextos diferentes y, finalmente, si posibilita intercalar consultas con modificaciones contextuales incrementales.

Consulta	Minimiza Interacc.	Minimiza Modif.	Facilita Dist. de Carga	Distintos Contextos	Intercala Cons / Cont.
Varias *CCs*	No	No	No	Si	No
CCM	Si	No	Si	Si	No
CCMF	Si	Si	Si	No	No
IC	Si	Si	Si	Si	Si
ICM	Si	Si	Si	Si	Si

Cuadro 3.1: Características de los distintos tipos de consultas contextuales

Como se mostrará a continuación, la *ICM* corresponde a la consulta contextual extendida más general de las definidas en esta tesis. Se introducirán una serie de propiedades que muestran las relaciones existentes entre los distintos tipos de consultas, resaltando sus similitudes y diferencias.

Existen diversos tipos de relaciones entre las consultas contextuales extendidas definidas en este capítulo. Estas relaciones están asociadas a qué consultas incluyen o son más generales que otras. La Figura 3.8 muestra las relaciones existentes entre los distintos tipos de consultas definidos, de acuerdo a las propiedades que serán introducidas en esta sección. En la figura, una relación $C_1 \xrightarrow{P} C_2$ indica que el tipo de consulta C_1 *es un caso particular de* C_2 y que la Propiedad P establece y demuestra dicha relación. En la figura se puede apreciar cómo, a partir de las propiedades 3.1, 3.4 y 3.5 y el hecho de que la *IC* es, por definición, una caso particular de la *ICM*, se puede concluir que la *ICM* incluye a todos los tipos de consultas contextuales definidos previamente.

En el caso específico de las *CCs*, es de esperar que sean un caso particular de cualquier consulta contextual extendida, como se mostrará en la siguiente propiedad y su correspondiente demostración.

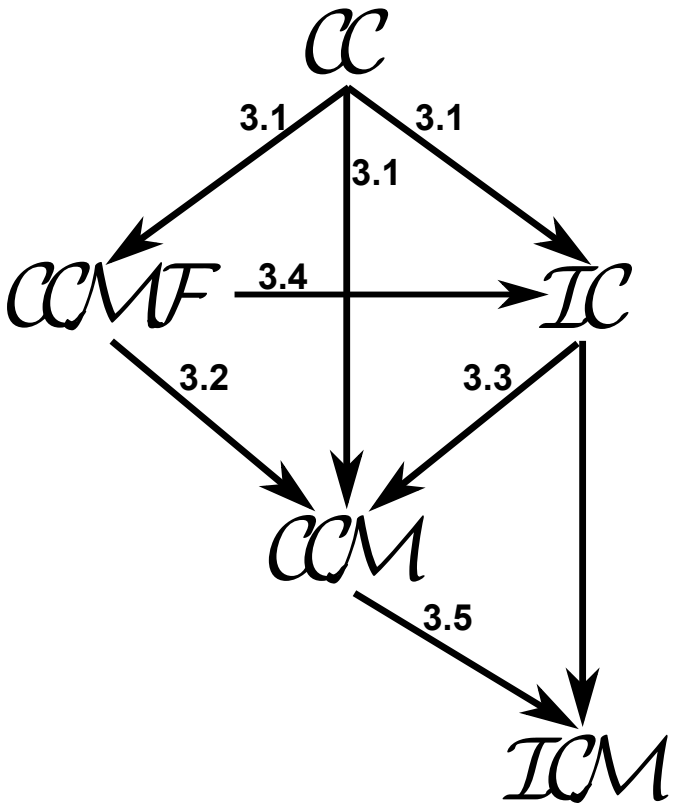

Figura 3.8: Relaciones entre los distintos tipos de Consultas

Propiedad 3.1 (de las Consultas Contextuales)**.** *Las CCs son un caso particular de las CCMs, de las CCMFs y de las ICs.*

Demostración. Una CC $[\mathcal{C}o, Q]$ puede ser reescrita como la CCM $\langle[\mathcal{C}o, Q]\rangle$, como la $CCMF$ $[\mathcal{C}o, \langle Q\rangle]$ y como la IC $[(\mathcal{C}o, Q)]$. La respuesta para $[\mathcal{C}o, Q]$ es la misma respuesta que para las consultas $\langle[\mathcal{C}o, Q]\rangle$, $[\mathcal{C}o, \langle Q\rangle]$ y $[(\mathcal{C}o, Q)]$. □

En el caso de las *CCMFs*, a continuación se introduce una propiedad con su correspondiente demostración donde se prueba que este tipo de consultas son un caso particular de las *CCMs*.

Propiedad 3.2 (de las Consultas Contextuales Múltiples Factorizadas)**.** *Las CCMFs son un caso particular de las CCMs.*

Demostración. Una *CCMF* de la forma $[\mathcal{C}o, \langle Q_1, Q_2, \ldots, Q_n\rangle]$ puede ser reescrita como la *CCM* $\langle[\mathcal{C}o, Q_1], [\mathcal{C}o, Q_2], \ldots [\mathcal{C}o, Q_n]\rangle$. La respuesta para $[\mathcal{C}o, \langle Q_1, Q_2, \ldots, Q_n\rangle]$ es la misma respuesta que para $\langle[\mathcal{C}o, Q_1], [\mathcal{C}o, Q_2], \ldots [\mathcal{C}o, Q_n]\rangle$ de acuerdo a las Definiciones 3.4 y 3.2 respectivamente. □

Una propiedad interesante de las *ICs* corresponde al hecho de que son un caso particular de las *CCMs* es decir, cualquier *IC* puede ser traducida a una *CCM* equivalente. A continuación se introduce dicha propiedad con la demostración correspondiente.

Propiedad 3.3 (de las Interrogaciones Contextuales)**.** *Las ICs son un caso particular de las CCMs.*

Demostración. Una *IC* $[(\mathcal{C}o_1, Q_1), (\mathcal{C}o_2, Q_2), \ldots, (\mathcal{C}o_n, Q_n)]$ puede ser reescrita como la *CCM* $\langle[\mathcal{C}o_1, Q_1], [\mathcal{C}o_1\Box\mathcal{C}o_2, Q_2], \ldots [\mathcal{C}o_1\Box\mathcal{C}o_2\Box\ldots\ \Box\mathcal{C}o_n, Q_n]\rangle$. La respuesta para $[(\mathcal{C}o_1, Q_1), (\mathcal{C}o_2, Q_2), \ldots, (\mathcal{C}o_n, Q_n)]$ es la misma respuesta que para $\langle[\mathcal{C}o_1, Q_1], [\mathcal{C}o_1\Box\mathcal{C}o_2, Q_2], \ldots [\mathcal{C}o_1\Box\mathcal{C}o_2\Box\ldots\ \Box\mathcal{C}o_n, Q_n]\rangle$ de acuerdo a las Definiciones 3.7 y 3.2 respectivamente. □

En la demostración de esta propiedad se puede observar la ventaja de las *ICs* con respecto a las *CCMs*. La *CCM* que obtiene la misma respuesta que la *IC* incluye contextos repetidos y, por lo tanto, generará modificaciones repetidas por parte del *SR* encargado de responderla.

En lo que respecta a las *CCMFs*, éstas pueden verse como un caso particular de las *ICs*. La siguiente propiedad establece dicha relación y su demostración corrobora la ventaja de la definición de las *CCMFs*.

Propiedad 3.4 (de las Consultas Contextuales Múltiples Factorizadas)**.** *Las CCMFs son un caso particular de las ICs.*

Demostración. Una *CCMF* de la forma $[\mathcal{C}o, \langle Q_1, Q_2, \ldots, Q_n\rangle]$ puede ser reescrita como la *IC* $[(\mathcal{C}o, Q_1), (\mathcal{C}o_\emptyset, Q_2), \ldots, (\mathcal{C}o_\emptyset, Q_n)]$, siendo $\mathcal{C}o_\emptyset$ el contexto vacío. La respuesta para $[\mathcal{C}o, \langle Q_1, Q_2, \ldots, Q_n\rangle]$ es la misma respuesta que para $[(\mathcal{C}o, Q_1), (\mathcal{C}o_\emptyset, Q_2), \ldots, (\mathcal{C}o_\emptyset, Q_n)]$ de acuerdo a las Definiciones 3.4 y 3.7 respectivamente. □

Considerando que una *CC* es un caso particular de una *IC* (Propiedad 3.1) y que una *CCM* es una secuencia de *CCs* de la misma forma que una *ICM* es una secuencia de *ICs*. Es posible establecer la siguiente propiedad:

Propiedad 3.5 (de las Consultas Contextuales Múltiples)**.** *Las CCMs son un caso particular de las ICMs.*

Demostración. Una *CCM* $\langle[\mathcal{C}o_1, Q_1], [\mathcal{C}o_2, Q_2], \ldots, [\mathcal{C}o_n, Q_n]\rangle$ es equivalente a la *ICM* $\langle[(\mathcal{C}o_1, Q_1)], [(\mathcal{C}o_2, Q_2)], \ldots, [(\mathcal{C}o_n, Q_n)]\rangle$. □

En la siguiente sección se continuará con el ejemplo del Servicio de Recomendación de Noticias (*SRN*), introducido en el capítulo anterior, utilizándolo para mostrar cómo se pueden aprovechar los distintos tipos de consultas introducidos en este capítulo para consultar al *SRN* en busca de recomendaciones de acuerdo a las preferencias de los clientes. En particular, se mostrarán distintas situaciones donde las consultas contextuales extendidas resultan útiles para los clientes del *SRN*.

3.6. Ejemplo de Aplicación de Consultas Extendidas: Servicio de Recomendación de Noticias

En el capítulo anterior se utilizó un Servicio de Recomendación de Noticias (*SRN*) para ejemplificar el uso de las *CCs* en el marco de la programación lógica. En esta sección se utilizará dicho *SRN* para ejemplificar el uso de las consultas contextuales extendidas definidas previamente. En particular, se mostrará a través de ejemplos concretos como reacciona el *SR* específico ante los distintos tipos de consultas.

Un cliente podría estar interesado en noticias relacionadas con el *fútbol* o el *básquet* del *Club Atlético River Plate*, deseando evitar las noticias del periódico *periódico1*, y también estar interesado en noticias de *tenis*. En especial, en aquellas que tengan a *David Nalbandian* como protagonista, donde se mencione la *Copa Davis* y que no hayan sido publicadas por *periódico2*. En este caso, el cliente podría ejecutar la *CCM* $\langle[\langle(\mathcal{P}_1^+, \oplus), (\mathcal{P}_1^-, \ominus)\rangle, Q_1], [\langle(\mathcal{P}_2^+, \oplus), (\mathcal{P}_2^-, \ominus)\rangle, Q_2]\rangle$ siendo:

$$\mathcal{P}_1^+ = \left\{ \begin{array}{l} interesado\_en(Id) \leftarrow deporte(Id, fútbol), \\ interesado\_en(Id) \leftarrow deporte(Id, básquet) \end{array} \right\}$$
$$\mathcal{P}_1^- = \{considera(periódico, periódico1)\}$$
$$Q_1 = busca(noticia(Id, Deporte, river, Autor, Periódico))$$
$$\mathcal{P}_2^+ = \{interesado\_en(Id) \leftarrow menciona(Id, copa\_davis)\}$$
$$\mathcal{P}_2^- = \{considera(periódico, periódico2)\}$$
$$Q_2 = busca(noticia(Id, tenis, david\_nalbandian, Autor, Periódico))$$

En este caso en particular, el *SRN* $\langle \mathcal{I}_{lp}, \{\oplus, \ominus\}, \mathcal{P}_{sports} \rangle$, al momento de responder la *CCM*, tomará de a una todas las consultas de la secuencia, en cualquier orden, y las irá respondiendo como si fuera una *CC* común. Suponiendo que primero computa la respuesta para la primer *CC*, el *SRN* modificará el programa $\mathcal{P}_{sports}$, agregando el programa lógico $\mathcal{P}_1^+$ y quitando $\mathcal{P}_1^-$, para luego utilizar el interprete lógico para obtener la respuesta a Q_1. Al momento de calcular la respuesta para la segunda *CC*, todas las modificaciones en el *SRN* para responder la consulta anterior no serán tenidas en cuenta y se iniciará el cálculo de la nueva respuesta utilizando el programa $\mathcal{P}_{sports}$. En este caso, se agregará a dicho programa almacenado el programa $\mathcal{P}_2^+$ y luego se quitará el programa $\mathcal{P}_2^-$, y sobre este nuevo programa se ejecutará el intérprete para computar la respuesta a Q_2. Finalmente, el *SRN* devolverá la siguiente respuesta al cliente:

$$\left\langle \begin{array}{l} \left(SI, \left\{ \begin{array}{l} busca(noticia(5, fútbol, river, autor1, periódico3)), \\ busca(noticia(7, fútbol, river, autor3, periódico2)) \end{array} \right\} \right), \\ \left(SI, \left\{ busca(noticia(4, tenis, david\_nalbandian, autor2, periódico1)) \right\} \right) \end{array} \right\rangle$$

Otra posibilidad es que el cliente esté interesado en obtener noticias de fútbol de los Clubes *River Plate* y *Boca Juniors*, en los cuales *Diego Maradona* sea mencionado. Dicho cliente puede no estar interesado en noticias escritas por *autor2* ni en aquellas publicadas por *periódico1*. En este caso, este cliente puede ejecutar la siguiente *CCMF* $[\langle(\mathcal{P}_3^+, \oplus), (\mathcal{P}_3^-, \ominus)\rangle, \langle Q_1, Q_3\rangle]$ siendo:

$$\mathcal{P}_3^+ = \{interesado\_en(Id) \leftarrow menciona(Id, diego\_maradona)\}$$
$$\mathcal{P}_3^- = \{considera(periódico, periódico1), considera(autor, autor2)\}$$
$$Q_3 = buscar(noticia(Id, Deporte, boca, Autor, Periódico))$$

En el caso de las *CCMFs*, los *SRs* tienen la posibilidad de modificar el programa almacenado una única vez para responder todas las consultas de la secuencia, evitando

modificaciones repetitivas e innecesarias. En el caso particular de la *CCMF* introducida anteriormente, el *SRN* $\langle \mathcal{I}_{lp}, \{\oplus, \ominus\}, \mathcal{P}_{sports} \rangle$ procederá a modificar el programa $\mathcal{P}_{sports}$ agregando el programa $\mathcal{P}_3^+$ y quitando $\mathcal{P}_3^-$. Una vez que se obtiene el nuevo programa, el ejecutará el intérprete de programación lógico para obtener las respuestas a las consultas Q_1 y Q_3, en cualquier orden. Luego, la respuesta a la *CCMF* es la secuencia:

$$\left\langle \begin{array}{l} (SI, \{busca(noticia(5, fútbol, river, autor1, periódico3))\}), \\ (SI, \{busca(noticia(2, fútbol, boca, autor3, periódico2))\}) \end{array} \right\rangle$$

Un cliente podría estar interesado en obtener una recomendación de una noticia del *Club Atlético River Plate*, en la cual se mencione a *Diego Maradona* y que no haya sido publicada por *periódico1* ni escrita por *autor2*. A su vez, también podría desear obtener otra noticia donde se mencione nuevamente al Club River Plate, pero en este caso, que esta noticia esté relacionada con Boca Juniors. Dicho cliente está interesado en mantener la restricción de que la noticia no haya sido publicada por *periódico1* ni escrita por *autor2*. En este caso, este cliente podría ejecutar la *IC* $[(\langle(\mathcal{P}_3^+, \oplus), (\mathcal{P}_3^-, \ominus)\rangle, Q_1), (\langle(\mathcal{P}_3^+, \ominus), (\mathcal{P}_4^+, \oplus)\rangle, Q_1)]$ siendo:

$$\mathcal{P}_4^+ = \left\{ \begin{array}{ll} interesado\_en(Id) \leftarrow & relacionado\_con(Id, OtroId), \\ & equipo(OtroId, boca) \end{array} \right\}$$

Al momento de resolver este tipo de consulta, el *SRN* $\langle \mathcal{I}_{lp}, \{\oplus, \ominus\}, \mathcal{P}_{sports} \rangle$ debe considerar el orden en el cual aparece cada consulta en la secuencia para computar la correspondiente respuesta. En la *IC* introducida previamente, el *SRN* lleva a cabo algunos cambios temporales asociados a la resolución de la Consulta Contextual Continuada inicial. Estos cambios corresponden a agregar $\mathcal{P}_3^+$ y quitar $\mathcal{P}_3^-$, para luego resolver la consulta Q_1 utilizando el intérprete de programación lógica. En este tipo de consulta, los cambios realizados sobre el programa almacenado se mantienen para futuras consultas de la misma *IC*. Luego, al momento de resolver la segunda consulta del *IC*, el *SRN* le quita al programa modificado el programa $\mathcal{P}_3^+$ para luego agregar el programa $\mathcal{P}_4^+$. Una vez obtenido el nuevo programa, el *SRN* procederá a calcular la respuesta a la consulta Q_1 utilizando el interprete de programación lógica. Finalmente, el *SRN* le envía la siguiente respuesta al cliente:

$$\left\langle \begin{array}{l} (SI, \{busca(noticia(5, fútbol, river, autor1, periódico3))\}), \\ (SI, \{busca(noticia(7, fútbol, river, autor3, periódico2))\}) \end{array} \right\rangle$$

En esta sección se mostró a través de ejemplos concretos una posible aplicación de las consultas contextuales extendidas para situaciones específicas donde un cliente necesitaba obtener información de un *SR*. En la siguiente sección se introducirán nuevas versiones de estas consultas, en este caso, en el marco de representación de conocimiento DeLP, junto con la continuación del ejemplo de aplicación introducido en el capítulo anterior, involucrando la recomendación de departamentos por parte de un *SR*.

3.7. Consultas Contextuales Extendidas en DeLP

En esta sección se introducirán las consultas contextuales extendidas en el marco de representación de conocimiento y razonamiento DeLP. La especificación de este tipo especial de consultas se llevará a cabo de manera similar a lo realizado en las primeras dos secciones de este capítulo para el marco de la programación lógica. Además, se mostrará su uso a través de la continuación del ejemplo de aplicación introducido en el capítulo anterior correspondiente al Servicio de Recomendación de Inmuebles (*SRI*).

La primer consulta contextual extendida que se introducirá en el marco de DeLP corresponde a la *CCM*, la cuál estará formada por una secuencia de consultas contextuales DeLP. De esta forma, un cliente puede enviar múltiples *CCs* en DeLP en una única consulta. La respuesta para este tipo de consultas corresponde a una secuencia de respuestas, una por cada consulta incluida en la *CCM* en DeLP .

Ejemplo 3.4 (Consulta Contextual Múltiple en DeLP). *Un agente puede estar interesado en obtener una recomendación sobre un departamento para alquilar, considerando que actúa en representación de una pareja de jóvenes profesionales y que dichos jóvenes no poseen un auto que les permita movilizarse con comodidad. Teniendo en cuenta que la pareja está planeando comprarse un auto, ellos también estarían interesados en una recomendación por parte del SRI considerando la posibilidad de que sean propietarios de una automóvil. En este caso, la pareja de jóvenes preferiría un departamento alejado del centro en el caso de que sea más económico. Luego, el agente puede realizar la Consulta Contextual Múltiple en DeLP* $\mathcal{CCM}_{delp} = \langle[\langle((\Pi_1, \Delta_1), +)\rangle, Q_1], [\langle((\Pi_2, \Delta_2), +)\rangle, Q_2]\rangle$ *al SRI, siendo:*

$$\Pi_1 = \{pareja\_joven, profesionales, \sim auto\},$$
$$\Delta_1 = \{\},$$
$$Q_1 = recomienda(Departamento)$$
$$\Pi_2 = \{pareja\_joven, profesionales, auto\},$$
$$\Delta_2 = \{alejado(D) \prec económico(D)\},$$
$$Q_2 = recomienda(Departamento).$$

La respuesta para la consulta contextual múltiple $\mathcal{CCM}_{delp}$ en el servicio de recomendación de inmuebles es:

$$\langle (SI, \{recomienda(ap1)\}), (SI, \{recomienda(ap3)\})\rangle$$

□

En el caso que el cliente necesite realizar diversas consultas con un mismo contexto, éste puede utilizar la $CCMF$. En el marco de representación de conocimiento DeLP $\mathcal{M}_{delp}$, este tipo especial de consultas estará compuesta por un contexto $\mathcal{C}o$ en el marco $\mathcal{M}_{delp}$ y por una secuencia de consultas DeLP $\langle Q_1, Q_2, \ldots, Q_n\rangle$. La respuesta correspondiente también será una secuencia, donde cada elemento de la secuencia corresponde al resultado de la CC en DeLP formada por el contexto $\mathcal{C}o$ y la consulta DeLP Q_i $(1 \leq i \leq n)$.

Ejemplo 3.5 (Consulta Contextual Múltiple Factorizada en DeLP). *La pareja de jóvenes puede estar interesada en conocer no sólo qué departamentos recomienda el SRI, sino además que departamentos cumplen al menos con las características buscadas por ellos y por otro lado, cuáles se encuentra ubicados en zonas de su preferencia. Luego, la pareja de jóvenes profesionales, que en este caso prefieren un departamento en los suburbios si es económico y no desean utilizar la información relacionada al auto en la recomendación, pueden realizar la siguiente Consulta Contextual Múltiple Factorizada en DeLP $\mathcal{CCMF}_{delp} = [\langle((\Pi_1, \Delta_1), +), ((\Pi_2, \Delta_2), -)\rangle, \langle Q_1, Q_2, Q_3\rangle]$:*

$$\Pi_1 = \{pareja\_joven, profesionales\},$$
$$\Delta_1 = \{alejado(D) \prec económico(D)\},$$
$$\Pi_2 = \{auto, \sim auto\},$$
$$\Delta_2 = \{\},$$
$$Q_1 = concuerdan\_caracteristicas(Departamento)$$
$$Q_2 = concuerda\_ubicación(Departamento)$$
$$Q_3 = recomienda(Departamento).$$

La respuesta para $\mathcal{CCMF}_{delp}$ en el SRI es:

$$\left\langle \begin{array}{c} (SI, \{recomienda(ap1)\}), \\ (SI, \{recomienda(ap1), recomienda(ap3)\}), \\ (SI, \{recomienda(ap1)\}) \end{array} \right\rangle$$

□

Por último, la IC puede utilizarse para agrupar varias consultas y sus contextos, minimizando las modificaciones repetitivas en el SR. La versión de este tipo de consulta en el marco $\mathcal{M}_{delp}$ estará compuesta por una secuencia de consultas contextuales continuadas $(\mathcal{C}o, Q)$, donde $\mathcal{C}o$ es un contexto DeLP y Q es una consulta DeLP. Cabe destacar que este tipo de consultas *continuadas* poseen una semántica especial asociada al contexto. Es decir, el contexto que afectará a un consulta continuada corresponde no sólo al contexto propio sino también al contexto de las consultas continuadas anteriores dentro de la misma IC. Luego, la respuesta para esta consulta especial será una secuencia formada por las respuestas a las consultas en la IC utilizando el programa almacenado con todas las modificaciones contextuales de las consultas anteriores en la misma IC.

Ejemplo 3.6 (Interrogación Contextual DeLP). *Una posibilidad es que la pareja joven de profesionales desee saber que departamentos le recomienda el servicio de razonamiento, primero asumiendo que tienen auto, luego que no tienen y finalmente sin utilizar dicha información. Luego, el agente correspondiente podría ejecutar la Interrogación Contextual en DeLP $\mathcal{IC}_{delp} = [(\langle((\Pi_1, \Delta_1), +)\rangle, Q_1), (\langle((\Pi_2, \Delta_2), +)\rangle, Q_2), (\langle((\Pi_3, \Delta_3), -)\rangle, Q_3)]$, siendo:*

$$\begin{array}{ll} \Pi_1 = \{pareja\_joven, profesionales, auto\}, & \Delta_1 = \{\}, \\ Q_1 = recomienda(Departamento), & \\ \Pi_2 = \{\sim auto\}, & \Delta_2 = \{\}, \\ Q_2 = recomienda(Departamento), & \\ \Pi_3 = \{\sim auto\}, & \Delta_3 = \{\}, \\ Q_3 = recomienda(Departamento). & \end{array}$$

En este caso, la primer consulta de la secuencia agrega el programa DeLP (Π_1, Δ_1) al programa almacenado en el SRI. Es importante notar que, como se utiliza el operador priorizado, en el programa modificado resultante estará el hecho auto*. Al momento de*

resolver la segunda consulta de la secuencia, se deben considerar las modificaciones realizadas por la consulta anterior, con lo cual, la modificación correspondiente a agregar el hecho ∼auto al programa actual generará la eliminación del hecho auto *para mantener la consistencia del mismo. En el caso de la tercer consulta contextual, se deben considerar las modificaciones realizadas tanto por la primer consulta como por la segunda. Luego, el programa actual tiene el hecho ∼auto, el cuál será removido por la operación de sustracción y, por lo tanto, al momento de computar la respuesta a la consulta no se utilizará información relativa al automóvil.*

La respuesta para $\mathcal{IC}_{delp}$ *en el SRI es:*

$$\left\langle \begin{array}{c} (SI, \{recomienda(ap1)\}), \\ (SI, \{recomienda(ap1), recomienda(ap3)\}), \\ (SI, \{recomienda(ap1)\}) \end{array} \right\rangle$$

□

En esta sección se mostraron las variantes de las consultas contextuales extendidas en el marco de representación de conocimiento DeLP y se ejemplificó su uso continuando el ejemplo de aplicación introducido en el capítulo anterior, relativa a la recomendación de departamentos. Como se mostró a lo largo del presente capítulo, estos nuevos tipos de consultas permiten un cómputo eficiente de las respuestas a las consultas de los clientes, conforme a lo que motivó su creación.

3.8. Conclusiones

En este capítulo se introdujeron cuatro tipos de consultas que extienden a las consultas contextuales definidas en el capítulo anterior. El principal objetivo de este nuevo tipo de consultas es lograr eficiencia en el cálculo de las respuestas, explorando todos los posibles escenarios que se le pueden presentar a los agentes que deseen interactuar con un servicio de razonamiento. Estas consultas permiten mayor declaratividad en los agentes que las utilizan, evitando la sobrecarga en el intercambio de mensajes y las incorporaciones de los contextos, así como también facilitando la distribución del cómputo de las respuestas.

Las consultas contextuales múltiples minimizan el costo de la infraestructura de mensajes y comunicación. Las consultas contextuales múltiples factorizadas, además de minimizar dicho costo, en especial, en el tamaño requerido para representar la consulta, evitan la realización de cambios repetitivos en el servicio de razonamiento.

En el caso de la interrogación contextual, éstas proveen de una gran flexibilidad y se adaptan a las necesidades de los agentes, permitiendo modificaciones incrementales al programa en el servicio de razonamiento intercalando consultas. Completando el espectro, las interrogaciones contextuales múltiples proveen la alternativa más general a la hora de consultar servicios de razonamiento.

Además, en el presente capítulo se incluyeron las distintas consultas múltiples para los formalismos previamente abordados en esta tesis, es decir, para la programación lógica y DeLP. Se continuó también con los ejemplos de aplicación introducidos en el capítulo anterior, mostrando la utilidad de estos tipos de consultas en los dominios de aplicación elegidos.

Capítulo 4

Grupos de Razonamiento

En el modelo desarrollado en los capítulos 2 y 3, un cliente interesado en realizar una *CC* debe saber a qué *SR* enviar la misma. Es decir, el cliente es el responsable de decidir a cuál o cuales *SRs* enviar su consulta. Para poder tomar la decisión correcta, un cliente debe poseer información actualizada de los distintos *SRs* disponibles. En este capítulo se introducirá el concepto de *Grupos de Razonamiento*, siendo uno de sus principales objetivos el de evitar al cliente el proceso de decisión para determinar cuál de los *SRs* disponibles debe resolver su consulta.

Las consultas introducidas en los capítulos previos corresponden a esquemas de interacción entre clientes y *SRs*, representando distintas formas en las cuales un cliente puede consultar un *SR* utilizando un contexto para su consulta. En este capítulo se analizarán las alternativas que poseen los clientes ante la existencia de múltiples *SRs*. El concepto de *Grupo de Razonamiento* posee como uno de sus objetivos el de permitir distribuir el cómputo de la respuesta. Esta distribución brindará, por un lado, la posibilidad de seleccionar el *SR* más indicado para la resolución de la consulta, y por otro lado, también permitirá calcular la respuesta en paralelo en distintos *SRs*, cuando esto sea posible.

El comportamiento de un Grupo de Razonamiento será definido tanto para la recepción de *CCs* como para la recepción de consultas contextuales extendidas. La definición de este concepto tiene como principales objetivos abstraer al cliente de la necesidad de decidir a que *SR* enviar la consulta, como se verá en la sección 4.1, y permitir mejorar la eficiencia en el cálculo de la respuesta a consultas contextuales extendidas de manera automática y transparente, como se mostrará en la sección 4.2.

El procesamiento de consultas contextuales extendidas por parte de Grupos de Razonamiento también presenta nuevas oportunidades al momento de determinar como se resuelven dichas consultas:

- Considerar a la consulta contextual extendida como una consulta normal y seleccionar el *SR* más indicado para resolverla,
- Considerar a la consulta contextual extendida como múltiples *CCs* (equivalentes) y seleccionar para cada una de estas consultas el *SR* indicado, o
- Transformar la consulta contextual original en nuevas consultas contextuales extendidas equivalentes, permitiendo una distribución eficiente entre los distintos *SRs*.

Desde el punto de vista de la interacción, el cliente podrá consultar indistintamente a un *SR* o a un Grupo de Razonamiento. La diferencia residirá en la forma en la cuál se resuelven las consultas.

El capítulo está organizado de la siguiente manera. En primer lugar, en la sección 4.1, se definirá el concepto de *Grupo de Razonamiento* (*GR*) y se distinguirán *GR homogéneos* y *GR heterogéneos*. En esa sección se mostrará además, cómo estos conceptos permiten abstraer al cliente de la decisión de a que *SR* enviar su *CC*, y el cliente podrá ver al grupo como si fuera un único *SR*. La tarea de decisión estará encapsulada en una *función de selección* de *SR*.

Luego, en la sección 4.2, se presentan tres alternativas que poseen los *GRs* para procesar una consulta contextual extendida, buscando maximizar la eficiencia en el cálculo de la respuesta. Se analizarán dichas posibilidades, comparándolas e introduciendo nuevas definiciones de respuesta. En la sección 4.3 se mencionan diferentes formas de implementar las alternativas antes mencionadas, incluyendo tanto la descomposición de las consultas como la distribución de las mismas. Finalmente, se considera la extensión del concepto de *GR* a Grupos de *GR*.

4.1. Grupos de *SR* y Función de Selección

Existen escenarios en los cuales el cliente puede no estar interesado en decidir a qué *SR* enviar la consulta, es decir, le resulta indiferente quién resuelve su consulta. Otra alternativa es que el cliente se encuentre incapacitado de determinar cuál de los *SRs* debería

resolver su consulta, posiblemente por falta de información. Para esto, se propone considerar a los múltiples *SRs* como un grupo, incluyendo un mecanismo que permita determinar ante la llegada de una consulta al grupo qué *SR* debe resolverla.

Cuando se dispone de un grupo de *SRs*, es necesario poder diferenciar entre las capacidades de los distintos *SRs* con respecto a las posibles *CCs* que pueden recibir. Es decir, resulta necesario poder determinar qué *SR* dentro de un grupo resulta más indicado para responder las *CCs*. En el modelo que se desarrollará en este capítulo, esta tarea estará determinada por una función de selección que dado un conjunto de *SRs* y una *CC* devolverá un *SR* para resolver la consulta. De esta manera, un grupo de razonamiento estará formado por un conjunto de SRs y una función de selección σ (ver Figura 4.1).

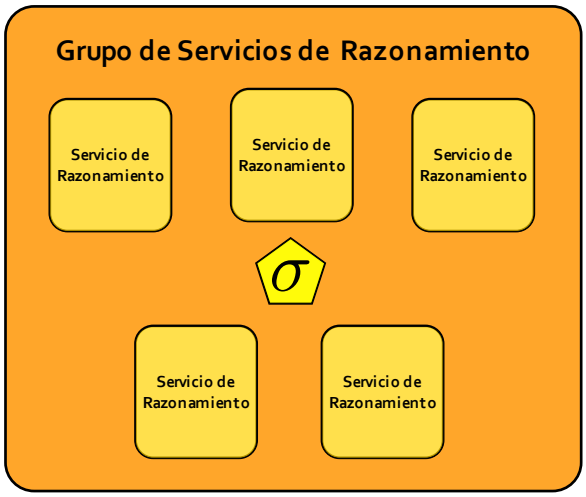

Figura 4.1: Grupo de Razonamiento

La noción de función de selección permite encapsular en un *GR* el mecanismo encargado de seleccionar que *SR* deberá resolver una *CC* determinada. De esta manera, la función σ es modular y puede ser reemplazada dependiendo de las necesidades del dominio de aplicación.

En la literatura existen diversos enfoques para llevar a cabo esta tarea [HS99], algunos de estos enfoques serán mencionados más adelante en la sección 4.3. Una función de selección podría implementarse de diferentes maneras desde elegir un *SR* al azar, hasta utilizar información respecto de los *SRs* existentes, así como también la *CC* recibida, para decidir cuál de ellos es el más *indicado*.

La definición de esta función es fundamental y determina el comportamiento de todas las *CCs* enviadas a un grupo. En particular, una función de selección podría llevar un registro de la cantidad de *CCs* respondidas por cada *SR* junto con el tiempo promedio de respuesta. De esta forma, ante la llegada de una *CC*, la función seleccionará al *SR* que haya respondido la menor cantidad consultas y tenga el menor tiempo promedio de respuesta.

Definición 4.1 (Función de Selección)**.** *Sea* $\mathcal{M}$ *un marco de representación de conocimiento,* $\mathbb{SR}_{\mathcal{M}}$ *el conjunto de todos los SRs para* $\mathcal{M}$ *y* $\mathbb{CC}_{\mathcal{M}}$ *el conjunto de todas las posibles CCs en* $\mathcal{M}$*. Una* función de selección *en* $\mathcal{M}$ *es una función* $\sigma : \mathcal{P}(\mathbb{SR}_{\mathcal{M}}) \times \mathbb{CC}_{\mathcal{M}} \Rightarrow \mathbb{SR}_{\mathcal{M}}$.

Por otro lado, también es posible utilizar una función de selección que considere la información incluida tanto en la *CC* como en los distintos *SRs*. En este sentido, las *CCs* proveen información que puede ser aprovechada a la hora de decidir la aptitud de un determinado *SR* para resolver la consulta. Por ejemplo, una función de selección puede utilizar el contexto de la *CC* recibida y compararlo con los distintos programas de los *SRs*, obteniendo un valor que indique el grado de pertinencia de dicha consulta al *SR*. Dicho valor se puede calcular, sintácticamente, contando la cantidad de cláusulas o términos incluidos en los programas del contexto que están a su vez incluidos en el programa del *SR*. De esta forma, la función podría seleccionar al *SR* que mayor valor de pertinencia obtenga.

A partir de la función de selección es posible definir cómo estará compuesto un *Grupo de Razonamiento* (*GR*), mostrado en la Figura 4.1. A continuación se introduce la definición correspondiente:

Definición 4.2 (Grupo de Razonamiento (*GR*))**.** *Sea* $\mathbb{SR}_{\mathcal{M}}$ *el conjunto de todos los SRs en el marco de representación de conocimiento* $\mathcal{M}$*. Un* grupo de razonamiento *para* $\mathcal{M}$ *es un par* $\langle \mathbf{SR}, \sigma \rangle$*, donde* $\mathbf{SR}$ *es un subconjunto de SRs de* $\mathbb{SR}_{\mathcal{M}}$ *y* σ *es una función de selección en* $\mathcal{M}$*.*

La definición de estos *GRs* tiene como objetivo formalizar su conformación y, a su vez, permitir a los clientes consultar de manera transparente a un *GR* o a un *SR*. Es decir, los clientes deberían poder enviar la misma *CC* tanto a un *SR* como a un *GR*.

En lo que resta de esta sección se formalizarán las distintas características de los *GRs* relacionadas al mecanismo de decisión y la llegada de *CCs* así como también se definirán

las respuestas para dichas consultas por parte de estos *GRs*. Cabe aclarar que un análisis más detallado de las alternativas a la hora de desarrollar un mecanismo de selección para dominios de aplicación específicos está fuera del alcance de esta tesis.

Considerando que en el Capítulo 2 se definió la respuesta para una *CC* por parte de un *SR*, a continuación se procederá a definir la respuesta para una *CC* por parte de un grupo.

Definición 4.3 (Respuesta para una *CC* por un *GR*)**.** *Sea* $\mathcal{CC}$ *una consulta contextual para el grupo de razonamiento* $\langle \mathbf{SR}, \sigma \rangle$ *en el marco de representación de conocimiento* $\mathcal{M}$*. La* respuesta *para* $\mathcal{CC}$ *en* $\langle \mathbf{SR}, \sigma \rangle$ *es* $\mathcal{A}ns(\mathcal{CC}, \sigma(\mathbf{SR}, \mathcal{CC}))$*.*

Cabe destacar que esta definición muestra como resuelve un *GR* la llegada de una *CC*. Es decir, como se puede ver en la Figura 4.2, el *GR* utilizará su función de selección para determinar cuál de los *SRs* que dispone es el más indicado para resolver la consulta y, luego, le enviará la consulta correspondiente. Por lo tanto, la respuesta otorgada por el *GR* será la respuesta que brinde el *SR* seleccionado.

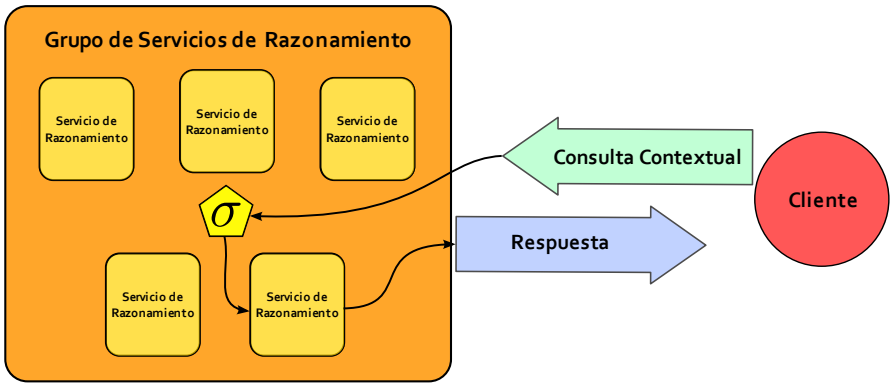

Figura 4.2: Respuesta para una *CC* por un Grupo

Los *SRs* que conforman un *GR* podrían ser todos idénticos, es decir, *SRs* que ante una misma consulta generarán una misma respuesta. Por lo tanto, un Grupo de Razonamiento Homogéneo es aquel en el cuál todos los *SRs* que forman el *GR* son idénticos. A continuación se introducen las definiciones correspondientes:

Definición 4.4 (Servicios de Razonamiento Idénticos)**.** *Sean* $\mathcal{SR}_1$ *y* $\mathcal{SR}_2$ *dos servicios de razonamiento para un marco de representación* $\mathcal{M}$*. Dichos servicios* $\mathcal{SR}_1$ *y* $\mathcal{SR}_2$

serán considerados idénticos *si para toda consulta contextual* $\mathcal{CC}$ *en* $\mathcal{M}$, $\mathcal{Ans}(\mathcal{CC}, \mathcal{SR}_1) = \mathcal{Ans}(\mathcal{CC}, \mathcal{SR}_2)$.

Definición 4.5 (Grupo de Razonamiento Homogéneo)**.** *Un grupo de razonamiento* $\langle \mathbf{SR}, \sigma \rangle$ *es* homogéneo *si para todo* $\mathcal{SR}_1, \mathcal{SR}_2 \in \mathbf{SR}$, $\mathcal{SR}_1$ *y* $\mathcal{SR}_2$ *son idénticos.*

En este caso, la decisión de a qué *SR* enviarle una consulta dentro de un mismo *GR* tiene como principal objetivo obtener la respuesta en el menor tiempo posible. Es decir, el problema está asociado a la eficiencia, en particular a la búsqueda del *SR* que pueda resolver la consulta más rápidamente.

Otra alternativa es que el *GR* esté conformado por *SRs* que no sean idénticos, en cuyo caso, existe la posibilidad de que las respuestas obtenidas desde estos *SRs* ante la misma consulta sean diferentes. En este escenario, la búsqueda puede estar orientada a obtener la respuesta utilizando el *SR* más *indicado* de los que forman el *GR*.

Definición 4.6 (Grupo de Razonamiento Heterogéneo)**.** *Un grupo de razonamiento* $\langle \mathbf{SR}, \sigma \rangle$ *es* heterogéneo *si existen al menos dos servicios de razonamiento* $\mathcal{SR}_1, \mathcal{SR}_2 \in \mathbf{SR}$, *tal que* $\mathcal{SR}_1$ *y* $\mathcal{SR}_2$ *no sean idénticos.*

En la literatura, uno de los enfoques más utilizados a la hora de determinar cuando un servicio es *indicado* para resolver una consulta de un cliente se basa en la especificación de las características de cada servicio [HZB+06, LAWG05]. Se han desarrollado lenguajes, como por ejemplo OWL-S [MPM+04], cuyo principal objetivo es permitir una descripción semántica de los servicios para facilitar una selección automática y uso de los mismos por parte de los clientes.

Al momento de determinar qué *SR* resolverá una *CC*, es importante destacar que el mecanismo responsable de tomar esa decisión (implementando el concepto de función de selección) puede aprovechar la información incluida en la *CC*, es decir, el contexto. Este contexto puede resultar muy valioso a la hora de inclinarse por un *SR* u otro, dependiendo del dominio de aplicación de las *CCs*.

Por ejemplo, una posible alternativa es seleccionar al *SR* que posea más conocimiento relacionado con el contexto y la consulta en sí. Otra posibilidad es que se utilice una combinación de conocimiento relacionado con la consulta y prestigio del *SR* en el área o tema. En general, la decisión dependerá de los *SRs* disponibles, el cliente y, principalmente, con la *CC* a resolver.

La noción de *Grupo de Razonamiento* permite abstraer al cliente de la decisión de a que *SR* enviar la *CC*. En este escenario, esta tarea recae en el *GR*, más específicamente en la función de selección, la cuál cuenta con la *CC* a resolver y la posibilidad de obtener información actualizada de los *SRs* disponibles. Por lo tanto, dicha función de selección posee toda la información necesaria para tomar la decisión correcta.

Por ejemplo, un *GR* podría estar formado por *SRs* que representen distintas agencias de turismo. Cada *SR*, representando una agencias de turismo, responde consultas sobre los destinos turísticos que ofrecen a sus clientes. Supongamos la existencia de tres *SRs*, cada uno de ellos representando a una agencia de turismo distinta. El primer *SR* representa a una agencia de turismo que ofrece viajes dentro de Argentina. El segundo *SR* responde consultas de una agencia de turismo que comercializa viajes a distintas partes del mundo, incluyendo preferentemente destinos con mucha historia y lugares por conocer. En el caso del tercer *SR*, éste representa a una agencia de turismo que ofrece, principalmente, viajes a playas paradisíacas del Caribe y el norte de Brasil.

La función de selección será la encargada de seleccionar el *SR* que considere más indicado de acuerdo a las preferencias de los clientes. Un cliente podría estar interesado en realizar un viaje a lugar con playa de aguas transparentes y clima cálido. La función de selección, utilizando el contexto de la consulta, enviará la misma al tercer *SR* para ser resuelta. En cambio, la consulta de un cliente interesado en conocer las principales ciudades y museos de Europa será asignada, por la función de selección, al segundo *SR*.

En el caso de recibir una consulta contextual extendida, un *GR* posee nuevas posibilidades. Es decir, no sólo puede seleccionar el *SR* disponible más indicado, sino que también puede distribuirla para calcular la respuesta en paralelo. En la siguiente sección se analizarán las distintas alternativas que poseen los *GRs* ante la llegada de consultas contextuales extendidas.

4.2. Grupos y Consultas Contextuales Extendidas: Oportunidad de Paralelismo

La noción de *Grupo de Razonamiento*, además de abstraer al cliente de la necesidad de seleccionar un *SR* para resolver las *CCs*, permitirá mejorar la eficiencia en el cálculo de la respuesta a consultas contextuales extendidas. A lo largo de esta sección se describirán las

alternativas que poseen los *GRs* al momento de determinar cómo se resuelven las consultas contextuales extendidas.

4.2.1. Selección Automática de *SR*

La primera alternativa que poseen los *GRs* al momento de recibir una consulta contextual extendida es seleccionar el *SR* que crea más conveniente para resolver dicha consulta, de la misma forma que lo hacen con las *CCs*. Esta alternativa requiere que la función de selección tenga la capacidad de seleccionar el *SR* conveniente a partir de cualquier tipo de consulta, incluyendo las consultas contextuales extendidas. Es decir, debe poder diferenciar entre los cinco tipos de consultas posibles y seleccionar al *SR* más indicado. Por lo tanto, en esta alternativa sólo resulta necesario que el *GR* incluya una función de selección con la capacidad de interpretar todos los tipos de consultas. Las restantes definiciones resultan equivalentes.

4.2.2. Descomposición y Selección Automática

Los distintos tipos de consultas múltiples tienen la particularidad de incluir varias consultas contextuales individuales. Es por eso que, como se mostró en el capítulo anterior, cualquier tipo de consulta contextual extendida se puede descomponer en diversas consultas contextuales individuales.

Por lo tanto, la *segunda alternativa* consiste en descomponer la consulta recibida en consultas contextuales individuales para procesar cada una de ellas utilizando la Definición 4.3. De esta manera, el *GR* podría distribuir el cálculo de la respuesta a la consulta contextual extendida en los distintos servicios disponibles de acuerdo a la función de selección.

En el caso particular de una *CCM*, ésta puede separarse en múltiples *CCs* individuales sin mayores inconvenientes. La idea es considerar, desde el *GR*, la llegada de una *CCM* como la llegada de las distintas *CCs* individuales que surgen de la consulta múltiple. Por lo tanto, como puede verse en la Figura 4.3, el *GR* decidirá para cada *CC* individual cuál es el *SR* más indicado para resolver dicha consulta y luego se encargará de armar la respuesta a la *CCM* agrupando las respuestas a las *CCs* individuales.

A continuación se introduce la definición de respuesta para una *CCM* por un *GR* que describe el comportamiento antes mencionado, basándonos en la respuesta para una *CC* individual por un *GR*.

Definición 4.7 (Respuesta de un *GR* para una *CCM*)**.** *Sea* $\mathcal{M}$ *un marco de representación de conocimiento,* $\mathcal{CCM} = \langle [\mathcal{C}o_1, Q_1], [\mathcal{C}o_2, Q_2], \ldots, [\mathcal{C}o_n, Q_n] \rangle$ *una consulta contextual múltiple en* $\mathcal{M}$ *y* $\langle \mathbf{SR}, \sigma \rangle$ *un GR para* $\mathcal{M}$*. La respuesta para* $\mathcal{CCM}$ *en* $\langle \mathbf{SR}, \sigma \rangle$ *es la secuencia* $\langle \mathcal{A}ns_1, \mathcal{A}ns_2, \ldots, \mathcal{A}ns_n \rangle$*, donde cada* $\mathcal{A}ns_i, 1 \leq i \leq n$*, es la respuesta para la correspondiente consulta contextual* $[\mathcal{C}o_i, Q_i]$ *por* $\langle \mathbf{SR}, \sigma \rangle$*.*

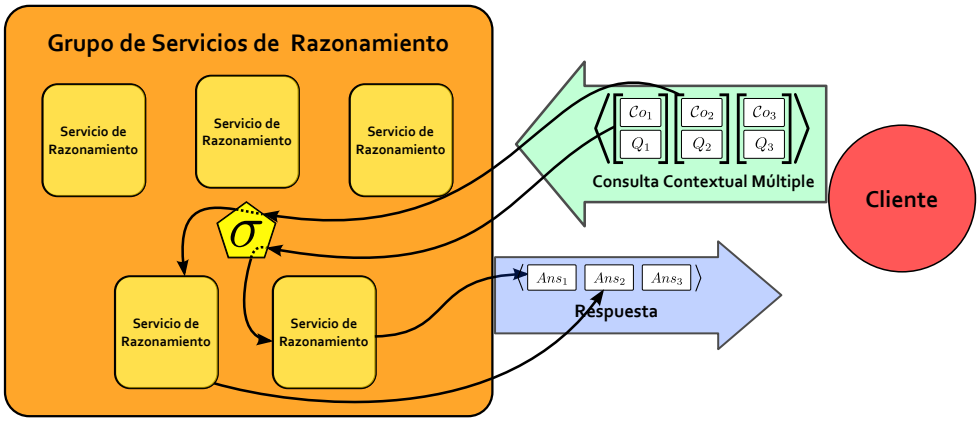

Figura 4.3: Respuesta para una *CCM* por un *GR*

En esta definición se puede notar la similitud con la Definición 3.2, correspondiente a la respuesta para una *CCM* en un *SR*. Las definiciones para las consultas contextuales extendidas restantes resultan análogas y es por eso que no se incluyen en esta tesis.

En esta alternativa, el mecanismo de decisión necesita estar capacitado para separar las consultas contextuales extendidas en *CCs* individuales para luego utilizar la función de selección y distribuir las *CCs* individuales. Finalmente, una vez obtenidas las distintas respuestas por parte de los *SRs* debe construir la respuesta correspondiente a la consulta contextual extendida. En este caso, el *GR* sólo necesita poder determinar para cada *CC* individual, cuál es el *SR* más indicado. Por lo tanto, la función de selección no sufre ningún tipo de modificación.

Comparando las alternativas presentadas hasta el momento, en la primera opción, el *GR* estaría cumpliendo con su objetivo principal de abstraer al cliente de la decisión de a

qué *SR* enviar su consulta. En cambio, la segunda posibilidad lograría no sólo cumplir ese objetivo, sino que además permitiría obtener más eficiencia en el cálculo de las respuestas a las *CCs* individuales incluidas en la consulta contextual extendida, a partir de la distribución del cálculo en los *SRs* más indicados.

Sin embargo, las consultas contextuales extendidas fueron creadas para evitar la sobrecarga en el intercambio de mensajes y las incorporaciones de los contextos, a través de agrupar *CCs* en distintos tipos de consultas contextuales extendidas. Por lo tanto, es de esperar que la separación de las consultas contextuales extendidas en *CCs* individuales aumente la cantidad de mensajes necesarios, el tamaño de los mismos y las modificaciones en los *SRs* de acuerdo a los contextos.

4.2.3. Reagrupación y Selección Automática

La *tercera posibilidad* busca obtener los beneficios de las dos alternativas antes mencionadas. Esta nueva alternativa intentará mantener las ventajas de las consultas contextuales extendidas asociadas a sus principales motivaciones, permitiendo, a su vez, una distribución eficiente del cálculo de las respuestas para obtener una mejor performance al momento de resolver las consultas.

La característica distintiva de esta nueva alternativa corresponde a que no sólo distribuirá el cálculo de la respuesta, como lo hace la segunda alternativa, sino que además, agrupará las *CCs* individuales generadas a partir de la consulta contextual extendida con el objetivo de enviar una única subconsulta contextual extendida a cada *SR*. Es decir, el *GR* separará las consulta contextual extendida en subconsultas contextuales extendidas asignadas a distintos *SRs* de acuerdo a la función de selección.

Para lograr esto, se utilizará una función de agrupamiento que, en el caso particular de las *CCMs*, dada una consulta de este tipo y un *GR*, nos devolverá un conjunto de pares de *CCMs* y *SRs*. Es decir, a partir de una *CCM*, generará nuevas subconsultas contextuales múltiples a partir de la consulta original y le asociará a cada una de ellas un *SR*. A continuación se introduce la definición correspondiente.

Definición 4.8 (Función de Agrupamiento)**.** *Sea $\mathcal{M}$ un marco de representación de conocimiento, $\mathbb{CCM}$ el conjunto de todas las posibles CCMs en $\mathcal{M}$, $\mathbb{GR}$ el conjunto de todos los posibles GRs para $\mathcal{M}$, $\mathbb{SR}$ el conjunto de todos los SRs para $\mathcal{M}$ y $\mathbb{MP}$ el conjunto*

de todos los posibles pares $(\mathcal{CCM}, \mathcal{SR}) \in \mathbb{CCM} \times \mathbb{SR}$. *Una función de agrupamiento en* $\mathcal{M}$ *es una función* $\alpha : \mathbb{CCM} \times \mathbb{GR} \mapsto \mathcal{P}(\mathbb{MP})$.

Ejemplo 4.1. *Sea* $SR_{4,1} = \{SR_1, SR_2, SR_3, SR_4\}$ *un conjunto de SRs,* $\sigma_{4,1}$ *una función de selección y* $CCM_{4,1} = \langle CC_1, CC_2, CC_3\rangle$ *una CCM. Dado el* $GR_{4,1} = \langle SR_{4,1}, \sigma_{4,1}\rangle$ *y los siguientes resultados de la función de selección:*

- $\sigma_{4,1}(SR_{4,1}, CC_1) = SR_2$
- $\sigma_{4,1}(SR_{4,1}, CC_2) = SR_4$
- $\sigma_{4,1}(SR_{4,1}, CC_3) = SR_2$

En este escenario, pueden existir distintas funciones de agrupamiento, entre ellas:

1. $\alpha_1(GR_{4,1}, CCM_{4,1}) = \{(\langle CC_1\rangle, SR_2), (\langle CC_2\rangle, SR_4), (\langle CC_3\rangle, SR_2)\}$
2. $\alpha_2(GR_{4,1}, CCM_{4,1}) = \{(\langle CC_1\rangle, SR_3), (\langle CC_2\rangle, SR_4), (\langle CC_3\rangle, SR_2)\}$
3. $\alpha_3(GR_{4,1}, CCM_{4,1}) = \{(\langle CC_1, CC_3\rangle, SR_2), (\langle CC_2\rangle, SR_4), (\langle CC_4\rangle, SR_1)\}$
4. $\alpha_4(GR_{4,1}, CCM_{4,1}) = \{(\langle CC_1, CC_3\rangle, SR_2), (\langle CC_2\rangle, SR_4)\}$
5. $\alpha_5(GR_{4,1}, CCM_{4,1}) = \{(\langle CC_1\rangle, SR_2), (\langle CC_2\rangle, SR_4)\}$

□

Una función de agrupamiento deberá cumplir con ciertas restricciones y condiciones para lograr un correcto desempeño. Una restricción que debe cumplir una función de agrupamiento es que cada *CC* incluida en una subconsulta contextual múltiple del conjunto de pares de respuesta debe pertenecer a la *CCM* original. Además, debe respetar la condición de que para cada *CC* individual incluida en una subconsulta contextual múltiple, el *SR* asociado a dicha subconsulta debe ser el mismo que devuelve la función de selección para la *CC* individual. Finalmente, el conjunto de respuesta no puede tener dos pares con el mismo *SR*. De acuerdo a estas restricciones y condiciones, a continuación se define cuando una función de agrupamiento es sensata.

Definición 4.9 (Función de Agrupamiento Sensata)**.** *Sea $\mathcal{M}$ un marco de representación de conocimiento, α una función de agrupamiento en $\mathcal{M}$, $\mathcal{CCM}$ una consulta contextual múltiple, $\mathcal{GR} = \langle \mathbf{SR}, \sigma \rangle$ un grupo de razonamiento para $\mathcal{M}$ y MP el resultado de la función $\alpha(\mathcal{CCM}, \mathcal{GR})$, la función de agrupamiento α será* sensata *si:*

- *para todo par $(subCCM, \mathcal{SR}) \in MP$, donde $subCCM = \langle \mathcal{CC}_1, \mathcal{CC}_2, \ldots, \mathcal{CC}_n \rangle$, cada consulta contextual $\mathcal{CC}_i$ pertenece a la consulta contextual múltiple $\mathcal{CCM}$ y $\sigma(\mathbf{SR}, \mathcal{CC}_i) = \mathcal{SR}$ $(1 \leq i \leq n)$.*
- *no existen dos pares $(subCCM_i, \mathcal{SR}_i)$ y $(subCCM_j, \mathcal{SR}_j)$ en MP con $i \neq j$ tal que $\mathcal{SR}_i = \mathcal{SR}_j$.*

En el Ejemplo 4.2.3, se pueden ver que la función de agrupamiento α_1 no es sensata, dado que no cumple con la segunda restricción de que no existan dos pares con igual *SR*. La función de agrupamiento α_2 no es sensata porque, además de no cumplir con la segunda restricción, tampoco cumple con la segunda parte de la primer restricción, ya que la agrupación obtenida no respeta la función de selección. En el caso de la función de agrupamiento α_3, ésta tampoco es sensata, debido a que no cumple con la primera parte de la primer restricción, es decir, existe una consulta contextual que no pertenece a la consulta contextual múltiple original. Finalmente, las funciones de agrupamiento α_4 y α_5 son sensatas, ya que cumplen con todas las restricciones mencionadas.

Una característica deseable para una función de agrupamiento corresponde a la posibilidad de asegurar que todas las *CCs* individuales que forman la consulta contextual extendida original estén incluidas en una de las subconsultas contextuales extendidas generadas. Una función de agrupamiento será completa si cumple con esta característica y se define a continuación:

Definición 4.10 (Función de Agrupamiento Completa)**.** *Sea $\mathcal{M}$ un marco de representación de conocimiento, α una función de agrupamiento en $\mathcal{M}$, $\mathcal{CCM}$ una consulta contextual múltiple, $\mathcal{GR} = \langle \mathbf{SR}, \sigma \rangle$ un grupo de razonamiento para $\mathcal{M}$ y MP el resultado de la función $\alpha(\mathcal{CCM}, \mathcal{GR})$, la función de agrupamiento α será* completa *si:*

- *para cada consulta contextual $\mathcal{CC}$ perteneciente a $\mathcal{CCM}$, existe un par $(subCCM, \mathcal{SR})$ en MP tal que $\mathcal{CC}$ pertenece a $subCCM$.*

En las funciones de agrupamiento mostradas en el Ejemplo 4.2.3, se puede observar que las funciones α_1, α_2, α_3 y α_4 son completas, ya que incluyen a todas las *CCs* incluidas en la *CCM*. En cambio, la función de agrupamiento α_5 no es completa debido a que no incluye a la consulta contextual CC_3, incluida en $CCM_{4,1}$. Por lo tanto, la función de agrupamiento α_4 es la única sensata y completa del Ejemplo 4.2.3.

A partir de la definición de la función de agrupamiento y las propiedades de completitud y sensatez de dicha función, se define el concepto de *Grupo de Razonamiento Extendido* (*GRE*), como se muestra en la Figura 4.4. Este *GRE* incluye, además de la función de selección y el conjunto de *SRs*, una función de agrupamiento sensata y completa. A continuación se introduce su definición:

Definición 4.11 (Grupo de Razonamiento Extendido (*GRE*))**.** *Sea* $\mathbb{SR}_{\mathcal{M}}$ *el conjunto de todos los SRs para el marco de representación de conocimiento* $\mathcal{M}$*. Un* grupo de razonamiento extendido *para* $\mathcal{M}$ *es una terna* $\langle \mathbf{SR}, \sigma, \alpha \rangle$*, donde* $\mathbf{SR}$ *es un subconjunto de SRs de* $\mathbb{SR}_{\mathcal{M}}$*,* σ *es una función de selección en* $\mathcal{M}$ *y* α *es una función de agrupamiento sensata y completa en* $\mathcal{M}$*.*

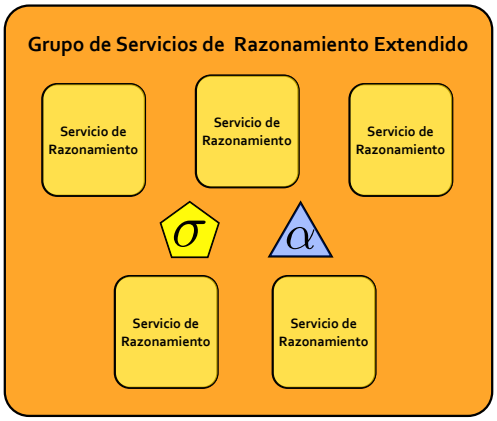

Figura 4.4: Grupo de Razonamiento Extendido

A continuación se define la respuesta para una *CCM* en un *GRE*, utilizando la tercera alternativa. En este caso, para calcular la respuesta a la *CCM* se utilizará, como se muestra en la Figura 4.5, la función de agrupamiento para obtener las subconsultas contextuales múltiples que serán enviadas a los distintos *SRs*. En este contexto, la respuesta a la *CCM*

corresponde a la secuencia de respuestas a las consultas individuales, cada una de ellas obtenidas en las subconsultas múltiples generadas y enviadas a los distintos *SRs*.

Definición 4.12 (Respuesta para una *CCM* por un *GRE*)**.** *Sea* $\mathcal{M}$ *un marco de representación de conocimiento,* $\mathcal{CCM} = \langle \mathcal{CC}_1, \mathcal{CC}_2, \ldots, \mathcal{CC}_n \rangle$ *una consulta contextual múltiple en* $\mathcal{M}$ *y* $\mathcal{GRE} = \langle \mathbf{SR}, \sigma, \alpha \rangle$ *un GRE para* $\mathcal{M}$*. La respuesta para* $\mathcal{CCM}$ *en* $\mathcal{GRE}$ *corresponde a la secuencia* $\langle \mathcal{A}ns_1, \mathcal{A}ns_2, \ldots, \mathcal{A}ns_n \rangle$*, donde cada* $\mathcal{A}ns_i$ $(1 \leq i \leq n)$ *se calcula de la siguiente forma. Sea* $MP = \{(sub\mathcal{CCM}_1, \mathcal{SR}_1), \ldots, (sub\mathcal{CCM}_m, \mathcal{SR}_m)\}$ *el resultado de la función de agrupamiento* $\alpha(\mathcal{CCM}, \langle \mathbf{SR}, \sigma \rangle)$*. Para cada* $\mathcal{CC}_i \in \mathcal{CCM}$ *existe un par* $(sub\mathcal{CCM}_j, \mathcal{SR}_j)$ *en MP, tal que* $\mathcal{CC}_i$ *pertenece a* $sub\mathcal{CCM}_j$ $(1 \leq i \leq n,\ 1 \leq j \leq m)$*. Luego,* $\mathcal{A}ns_i$ *es la respuesta para* $\mathcal{CC}_i$ *de la consulta contextual múltiple* $sub\mathcal{CCM}_j$ *en* $\mathcal{SR}_j$*.*

Análogamente a lo realizado para las *CCMs*, es posible introducir definiciones similares respecto de las *CCMFs*. En particular, se debería definir la función de agrupamiento y sus propiedades de sensatez y completitud. En todos los casos, al igual que en la definición de la respuesta, los cambios son simples renombres y modificaciones mínimas, con lo cual, la inclusión de dichas definiciones en esta tesis no lograría ningún aporte significativo.

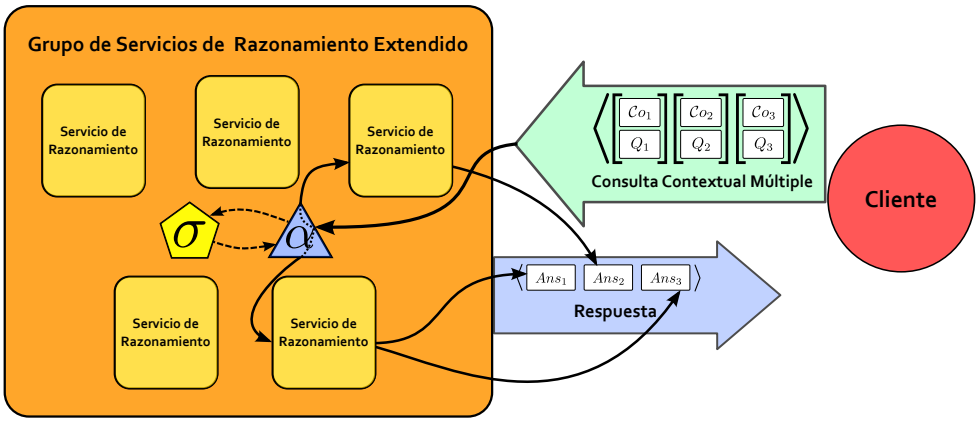

Figura 4.5: Respuesta para una *CCM* por un *GRE*

En el caso particular de las *ICs*, la función de agrupamiento de la interrogación, es decir, la idea de generar nuevas *ICs* a partir de una *IC* inicial, es básicamente la misma que para las *CCMs*. La principal diferencia es que en esta separación se debe tener especial cuidado de mantener la semántica de cada Consulta Contextual Continuada incluida en la *IC*.

Es por esta razón que se introducirá una nueva definición de función de agrupación sensata, en este caso en particular para las *ICs*. Por lo tanto, la función de agrupamiento, además de cumplir con las restricciones originales de función correspondiente para las *CCMs*, deberá respetar para cada consulta incluida en la *IC* original, el contexto sobre el cual se resuelve dicha consulta al momento de incluirla en una nueva *IC*.

Definición 4.13 (Función de Agrupamiento Sensata para *ICs*)**.** *Sea* $\mathcal{M}$ *un marco de representación de conocimiento,* α *una función de agrupamiento en* $\mathcal{M}$*,* $\mathcal{IC} = [(\mathcal{C}o_1, Q_1), \ldots, (\mathcal{C}o_n, Q_n)]$ *una interrogación contextual,* $\mathcal{GR} = \langle \mathbf{SR}, \sigma \rangle$ *un grupo de razonamiento para* $\mathcal{M}$ *y MP el resultado de la función* $\alpha(\mathcal{IC}, \mathcal{GR})$*, la función de agrupamiento* α *será* sensata *si:*

- *para todo par* $(sub\mathcal{IC}, \mathcal{SR}) \in MP$*,* $sub\mathcal{IC} = [(\mathcal{C}o_1^s, Q_1^s), \ldots, (\mathcal{C}o_m^s, Q_m^s)]$*, para cada* $(\mathcal{C}o_i^s, Q_i^s) \in sub\mathcal{IC}$*,* $\exists(\mathcal{C}o_j, Q_j) \in \mathcal{IC}$ *tal que* $\mathcal{C}o_1^s \square \mathcal{C}o_2^s \square \ldots \square \mathcal{C}o_i^s = \mathcal{C}o_1 \square \mathcal{C}o_2 \square \ldots \square \mathcal{C}o_j$ *y* $Q_i^s = Q_j$*,* $(1 \leq i \leq n,\ 1 \leq j \leq m)$*.*
- *no existen dos pares* $(sub\mathcal{IC}_i, \mathcal{SR}_i)$ *y* $(sub\mathcal{IC}_j, \mathcal{SR}_j)$ *en MP con* $i \neq j$ *tal que* $\mathcal{SR}_i = \mathcal{SR}_j$*.*

Las restantes definiciones para las *ICs* con respecto a esta alternativa, al igual que con las *CCMFs*, no presentan mayores diferencias y, por lo tanto, no se justifica su inclusión en esta tesis. Lo mismo ocurre para las *ICMs*, las cuales corresponden a una combinación entre las *ICs* y las *CCMs*.

Tanto la función de selección σ como la función de agrupamiento α corresponden a abstracciones que permiten definir el comportamiento de un *GR* a alto nivel. En la siguiente sección se presentarán distintas opciones a la hora de concretar los mecanismos de selección y agrupamiento para las consultas contextuales extendidas.

4.3. Opciones para Descomposición y Distribución

La concreción del concepto de *GR* requiere de un mecanismo que considere, ante la llegada de consultas contextuales, la posibilidad de descomponer la consulta y su posterior distribución entre los *SRs* que forman el *GR*, implementando de esta forma los conceptos definidos formalmente como función de selección y función de agrupamiento.

Este mecanismo puede desarrollarse de manera centralizada, ya sea como parte de un *SR* o bien como un ente separado cuya principal responsabilidad sea la de recibir las consultas contextuales dirigidas al *GR* y descomponerlas y distribuirlas entre los *SR* que forman el *GR*. Otra posibilidad corresponde a desarrollar esta capacidad de manera distribuida, por ejemplo, siendo parte de cada *SR* incluido en el *GR*.

La alternativa de desarrollar un ente centralizado tiene como principal ventaja la simplicidad de implementación, desde el punto de vista que concentra toda la información, no sólo de las consultas contextuales recibidas sino también de los *SR* que conforman al *GR*. De esta forma, la decisión de descomponer una consulta contextual y la forma en que se distribuye se puede tomar utilizando toda la información relevante. Esta alternativa posee las desventajas de toda solución centralizada, asociadas al cuello de botella y único punto de falla que representa el ente centralizado.

Por otro lado, una implementación distribuida evitaría las desventajas antes mencionadas, permitiendo, además, a los clientes enviar sus consultas a un *GR* a través de múltiples puntos de acceso. Sin embargo, en este tipo de implementaciones, la información necesaria para decidir cómo descomponer y distribuir las consultas contextuales recibidas se encuentra distribuida entre los distintos puntos de acceso y los *SRs*.

En la literatura existen diversos enfoques para llevar a cabo la descomposición y distribución de tareas. Algunos de estos enfoques se basan en unidades de información [RTK07], otros en sistemas de ranking (en inglés, *ranking systems*) [AT07] o en métodos utilizando índices y tokens [Vou07], mientras que existen algunos trabajos que utilizan variantes de Sistemas MultiAgente [WD07, JFAS07, ZJ07].

En la siguiente subsección, se analizarán las alternativas existentes a la hora de descomponer tareas, siendo, en el caso particular de esta tesis, las consultas contextuales. Al momento de descomponer consultas contextuales (tarea), se debería considerar no sólo a la consulta propiamente dicha, sino también a los *SRs* (recursos) disponibles dentro del *GR*, debido a que éstos serán, posiblemente, los encargados de resolver las consultas.

4.3.1. Descomposición de Consultas Contextuales Extendidas

La descomposición de las tareas puede ser llevada a cabo por el diseñador del sistema, al momento de la programación del mismo. En el caso particular de las consultas contextuales extendidas, ésta descomposición podría ser inherente de acuerdo a la representación de

las mismas. Esta alternativa corresponde a separar una consulta contextual extendida en *CCs* equivalentes, de acuerdo a la segunda alternativa explicada en la sección anterior, y utilizando las propiedades introducidas en el Capítulo 3.

Otra posible alternativa a la hora de descomponer las consultas contextuales extendidas corresponde a una separación espacial, teniendo en cuenta las fuentes de información o puntos de decisión. En el caso particular de las *CCs*, las fuentes de información son los *SRs* incluidos en el *GR* encargado de responder la consulta. Por lo tanto, este tipo de descomposición debería considerar cómo separar una consulta contextual extendida para generar nuevas consultas de acuerdo los *SRs* que poseen más información para resolverlas.

Por ejemplo, un *GR* podría incluir *SRs* destinados a recomendar noticias. En particular, podría existir un *SR* que recomiende noticias deportivas, como el introducido en en Capítulo 2 en la Sección 2.3. Además, podrían existir otros *SRs*, recomendando noticias policiales, políticas, económicas, entre otras. Luego, una consulta contextual extendida podría incluir varias consultas y, teniendo en cuenta el contexto, ésta se podría dividir de acuerdo al tipo de noticia que se está buscando.

Una tercer alternativa corresponde a una descomposición funcional, es decir, dependiendo de las capacidades de los *SRs*. En esta alternativa, la decisión de cómo descomponer una consulta contextual extendida busca generar nuevas *CCs* que resulten indicadas para los *SRs* que serán los encargados de resolverlas. En esta alternativa, al igual que en la anterior, la función de selección es clave a la hora de determinar qué *SR* resulta más indicado o posee más información para las distintas *CCs*.

Un *GR* podría estar formado por *SRs* que sugieran tratamientos para enfermedades de acuerdo a los síntomas de los pacientes, como el mostrado en el Capítulo 2 en el Ejemplo 2.1. En este *GR*, cada *SR* tendrá asociada una especialidad, como por ejemplo, cardiología, psicología, neurología, entre otras. Luego, una consulta contextual extendida puede ser descompuesta de acuerdo al contexto y a las especialidades de los distintos *SRs* incluidos en el *GR*.

Las alternativas mencionadas previamente corresponden a algunas de las posibilidades a la hora de descomponer una consulta contextual extendida. En el Capítulo 6 se describirá una alternativa de descomposición de las consultas que permite ser adaptada para implementar las últimas dos posibilidades mencionadas en esta subsección. En la siguiente subsección se introducirán alternativas existentes en la literatura [Dur99] para distribuir las tareas (consultas contextuales) entre los *SRs* que forman un grupo.

4.3.2. Distribución de Consultas Contextuales

Una vez descompuesta una consulta contextual extendida en consultas más simples, sólo resta distribuir el cómputo de las mismas dentro de los *SRs* incluidos en el *GR*. Para llevar a cabo dicha distribución, existen diversos criterios [DLC87] que se pueden aplicar de acuerdo a los objetivos del *GR* que se está desarrollando. A continuación se describen algunos de estos criterios.

Un posible criterio corresponde a evitar recargar recursos (o *SRs*) críticos. Este criterio tiene como principal objetivo lograr un balance de la carga entre los distintos *SRs*. Su aplicación está asociada mayormente a *GRs* Homogéneos, tratando de alcanzar la mayor eficiencia posible. Resulta, también, aplicable a *GRs* Heterogéneos, probablemente combinado con el criterio introducido a continuación.

Otro criterio posible está asociado a asignar las consultas a los *SRs* de acuerdo a sus capacidades. Este criterio está directamente relacionado con la tercer alternativa de descomposición de las consultas mencionada en la subsección anterior. Su aplicación tiene sentido para *GRs* heterogéneos y puede ser combinada con el criterio mencionado previamente, con el objetivo de mejorar la eficiencia del sistema en general.

La posibilidad de asignar responsabilidades solapadas a los *SRs* corresponde a un criterio que busca lograr coherencia o redundancia. Este criterio puede utilizarse, también, con el objetivo de minimizar el tiempo de respuesta, esperando por la primer respuesta obtenida. Por otro lado, también es posible desarrollar un *SR* especial, con una visión abarcativa de los demás *SR* que forman el *GR*, de forma tal de que este *SR* sea el encargado de distribuir las tareas.

La distribución de las consultas contextuales se concreta utilizando algún mecanismo específico. Una alternativa corresponde a utilizar el protocolo Contract Net [Smi81], el cual será descripto en detalle en el Capítulo 6. Por otro lado, el mecanismo de mercado corresponde a otra alternativa en la cual las consultas contextuales son asignadas a los *SRs* en base a un acuerdo generalizado o una selección mutua. Otras alternativas corresponden a utilizar planificación, o incluso, una estructura fija de responsabilidades de los *SRs*.

4.4. Grupos de Grupos

De la misma forma que pueden existir múltiples *SRs*, también podrían existir múltiples *GRs* con distintos *SRs* cada uno. Consecuentemente, el problema de decisión que se le presenta a un cliente acerca de a qué *SR* enviar una consulta, resuelto por un *GR*, surge nuevamente ante la existencia de múltiples *GRs*. Por lo tanto, un cliente nuevamente deberá decidir, en este caso, a qué *GR* enviar su consulta. Considerando que la problemática para el cliente es la misma que antes, simplemente reemplazando a los *SRs* por *GRs*, es posible aplicar la misma solución propuesta previamente en este capítulo.

En este caso, la definición de un nuevo tipo de grupo incluye *GRs* en vez de *SRs*, constituyéndose ésta característica en la principal diferencia con el concepto de grupo definido previamente. Además, la función de selección deberá elegir entre *GRs* en vez de *SRs*. Claramente, el análisis realizado previamente para el concepto de *GR* es fácilmente adaptable a este nuevo concepto. Es por esto que no se incluyen las definiciones correspondientes, teniendo en cuenta además que las modificaciones requeridas sobres las definiciones anteriores no son significativas.

Un posible ejemplo de aplicación corresponde al desarrollo de un grupo de grupos con el objetivo de asistir a clientes que necesiten o deseen realizar un viaje al exterior. Al momento de organizar un viaje de estas características, se deben considerar diversos aspectos que van desde la compra de los pasajes y la reserva del hospedaje hasta trámites de documentación requerida e, incluso, una consulta al médico con el objetivo de prevenir posibles enfermedades. Este grupo podría estar compuesto por *GRs* de distintas índoles, estando cada uno de ellos encargados de algún aspecto específico del viaje, como por ejemplo un *GR* de agencias de turismo que se encarguen de ofrecer información relacionada con los pasajes y los hospedajes disponibles.

Luego, un cliente podría consultar a este grupo, siendo el contexto la información relativa al viaje, como por ejemplo, la fecha en la que planea viajar, el presupuesto del que dispone, el lugar al cuál desea viajar. Las consultas que podría realizar incluyen la búsqueda de disponibilidad y precios de vuelos para el destino elegido, trámites necesarios para poder ingresar al país correspondiente, y hasta vacunas que debería tener para evitar enfermarse. El grupo de grupos deberá distribuir las consultas a los *GRs* correspondientes, y a su vez, cada *GR* podrá descomponer y distribuir la consulta recibida para que sea resuelta por el *SR* apropiado.

4.5. Conclusiones

En este capítulo se introdujeron los Grupos de Razonamiento, cuyo principal objetivo corresponde a abstraer al cliente de la decisión acerca de a que *SR* enviar su consulta contextual. Un Grupo de Razonamiento está formado por un conjunto de *SRs* y una función de selección. La función de selección σ es la encargada de decidir, ante una consulta contextual, qué *SR* de los que forman el grupo resulta más indicado para la resolución de la consulta.

Los Grupos de Razonamiento permiten, además, mejorar la eficiencia en el cálculo de la respuesta a consultas múltiples. En este capítulo se presentaron y analizaron tres alternativas que tienen los grupos a la hora de procesar este tipo de consultas. Se introdujo el concepto de Grupo de Razonamiento Extendido, que incluye una función de agrupamiento, con el objetivo de maximizar la eficiencia en el cálculo en paralelo de consultas múltiples.

En cuanto a la concreción de la distribución de las consultas contextuales y, en particular, la concreción de las funciones antes mencionadas, se describieron algunos mecanismos existentes en la literatura. Los mecanismos introducidos son utilizados en la descomposición y distribución de tareas en entornos cooperativos, y por lo tanto, pueden ser adaptados para su empleo en los grupos de razonamiento.

Finalmente, se mencionó una posible extensión del concepto de grupos que consisten en definir grupos de grupos de *SR*. La principal motivación de esta extensión es la misma que motivó la creación del concepto de grupos en un primer lugar, es decir, el objetivo de abstraer al cliente de la decisión de a quién enviar su consulta contextual.

Capítulo 5

Una Aplicación para *GR*: Intérprete DeLP Paralelo

En este capítulo se mostrará cómo un Grupo de Servicios de Razonamiento puede utilizarse para realizar una aplicación que permita explotar la capacidad de los *GR* de procesamiento en paralelo. Para ello se desarrollará una aplicación que implementa un intérprete de DeLP paralelo. La aplicación desarrollada puede verse como una aplicación de propósito general que a su vez puede ser utilizada o extendida para resolver problemas de dominios específicos. El intérprete desarrollado permitirá resolver consultas a programas DeLP obteniendo paralelismo de manera transparente.

Como se verá a lo largo de este capítulo, los conceptos de *CC*, *SR* y *GR* introducidos previamente permitirán la construcción de un intérprete DeLP paralelo que utilizará a un *GR* formado por *SR* específicos en el marco de la programación lógica para calcular, en paralelo, el estado de garantía de una determinada consulta DeLP en un programa DeLP. Los *SRs* que conformarán dicho *GR* serán denominados *Servicios de Razonamiento Lógicos Argumentativos* e incluirán tanto un programa lógico como operadores de programas, todo esto especialmente definido para permitir la construcción de argumentos y derrotadores utilizando el intérprete lógico y el programa DeLP que será cargado como parte del contexto.

5.1. Motivación

La Programación Lógica Rebatible (DeLP) permite la representación de conocimiento que involucre información incompleta o potencialmente contradictoria. A continuación se incluyen en forma resumida los conceptos involucrados en la aplicación que se desarrollará, más detalles pueden encontrarse en el Apéndice B. El mecanismo de inferencia sobre el cual está basado DeLP permite decidir entre conclusiones contradictorias y adaptarse fácilmente a entornos cambiantes. Para decidir cuando una conclusión C puede considerarse garantizada a partir de un programa lógico rebatible, se realiza un análisis dialéctico, construyendo argumentos a favor y en contra de la conclusión C. Las implementaciones actuales de DeLP computan el estado de garantía de una conclusión C en forma secuencial. Sin embargo, como se verá a continuación, estas podrían computarse en forma paralela utilizando un *GR*.

Un argumento que sustenta una conclusión C podrá ser atacado por otros argumentos derrotadores que lo contradigan y que puedan construirse a partir del programa. Dichos derrotadores podrán a su vez ser atacados, y así sucesivamente, generando una secuencia de argumentos llamada línea de argumentación. Cada línea de argumentación deberá satisfacer ciertas propiedades, a fin de evitar que se generen argumentos falaces. El proceso completo considera para cada argumento todos sus posibles argumentos derrotadores, lo cual, en lugar de una única línea de argumentación, genera un conjunto de líneas representadas con un árbol de dialéctica.

En el caso de que exista más de un argumento sustentando la conclusión C, se generará un árbol de dialéctica por cada uno de estos argumentos. Este análisis dialéctico determinará si la conclusión en cuestión está garantizada o no, a partir del programa. A fin de ejemplificar, en la Figura 5.1 se muestran posibles árboles de dialéctica que se podrían generar a partir de un programa con el objetivo de determinar si una conclusión C está o no garantizada. En este ejemplo hay dos argumentos que sustentan a C, los cuales están etiquetados con los números 1 y 12. Para cada uno de estos argumento existen derrotadores, por ejemplo, el argumento 1 se encuentra derrotado por los argumentos etiquetados con los números 2, 6 y 8, mientras que el argumento 12 se encuentra derrotado por los argumentos 13, 15 y 19. A su vez, cada uno de estos argumentos posee otros derrotadores, salvo el argumento 19 que al no poseer derrotadores se encuentra no derrotado.

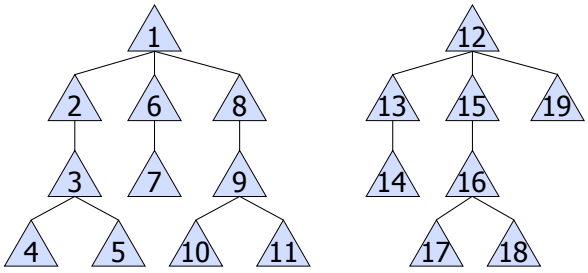

Figura 5.1: Posibles Árboles de Dialéctica

Las implementaciones existentes en la literatura de DeLP construyen los árboles de dialéctica de manera secuencial. Por ejemplo, en el orden en que están numerados los argumentos en la Figura 5.1. Sin embargo, al momento de construir estos árboles de dialéctica existe una oportunidad de paralelismo. En lo que resta de este capítulo se mostrará una alternativa a la hora de implementar un intérprete DeLP que intenta aprovechar esta oportunidad de paralelismo para disminuir el tiempo que se necesita para computar dichos árboles de dialéctica.

Por ejemplo, suponiendo que se dispone de 3 nodos, es posible distribuir el cómputo de los árboles en dichos nodos. Una posible distribución corresponde a la mostrada en la Figura 5.2, donde los argumentos 1 y 12 se calculan en el nodo 1, mientras que los argumentos 2, 6 y 8 se calculan en el nodo 2 y los argumentos 13, 15 y 19 en el nodo 3. El resto de los argumentos también se computan en estos nodos, logrando el cálculo en paralelo de los árboles de dialéctica.

En las siguientes secciones se mostrará como un *GR* permite implementar un intérprete de DeLP con procesamiento en paralelo. Para esto se utilizarán los conceptos de *SR*, *GR* y consultas contextuales extendidas. A continuación se introducirá un *SR* específico en el marco de la programación lógica que permitirá el cómputo de argumentos y contra-argumentos. Esta clase de *SR* será utilizada para formar *GRs*, los cuales estarán encargados de computar los argumentos que forman los árboles de dialéctica.

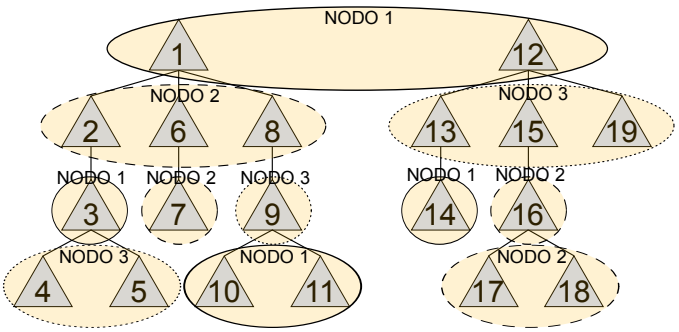

Figura 5.2: Árboles de Dialéctica calculados en paralelo

5.2. Servicio de Razonamiento Lógico Argumentativo

En esta sección introduciremos un *SR* específico en el marco de la programación lógica, denominado *Servicio de Razonamiento Lógico Argumentativo* (*SRLA*), que computará argumentos y contra-argumentos. Un *GR* formado por *SRLA*s (ver Figura 5.3) permitirá, como se verá en la siguiente sección, calcular en paralelo la respuesta a una consulta DeLP a partir de un programa DeLP, utilizando consultas contextuales.

Un *SRLA* (ver Figura 5.4) poseerá el intérprete de programa lógico ($\mathcal{I}_{lp}$), un programa lógico $\mathcal{P}_{Idelp}$ que permitirá calcular argumentos y derrotadores para estos argumentos y ciertos operadores específicos para programas DeLP. Estos operadores permitirán agregar programas DeLP, especificar el criterio de comparación entre argumentos e incluir las restricciones que deben cumplir las líneas de argumentación para evitar argumentos derrotadores falaces.

El programa lógico $\mathcal{P}_{Idelp}$ corresponde a una versión modificada del intérprete DeLP secuencial incluyendo dos predicados para ser utilizados por las consultas contextuales:

- $argumento(+Q, -\mathcal{A})$, y
- $derrotador(+\mathcal{A}, -\mathcal{D})$.

El primer predicado calculará un argumento $\mathcal{A}$ a partir de una consulta DeLP Q utilizando el programa DeLP agregado utilizando el operador correspondiente, que será explicado

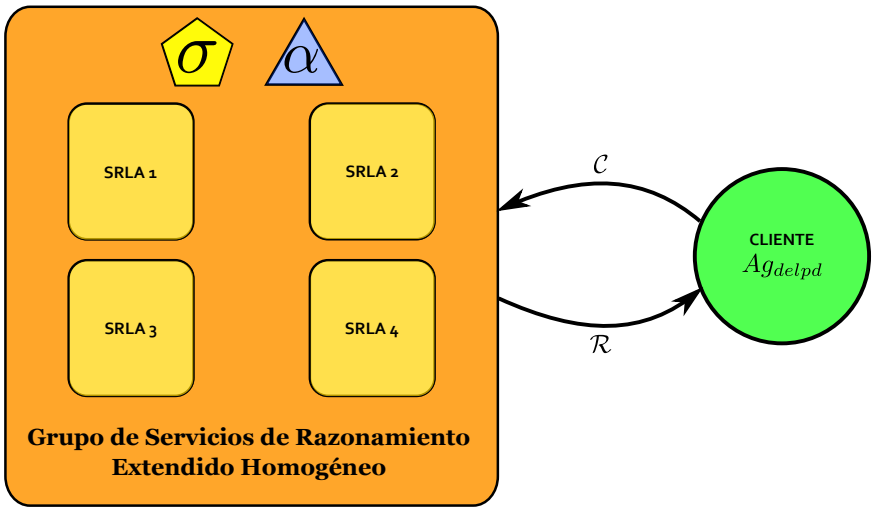

Figura 5.3: *GRE* formado por *SRLA*s

más adelante. El segundo predicado obtendrá un derrotador $\mathcal{D}$ para un argumento $\mathcal{A}$ dado utilizando, en este caso, no sólo el programa DeLP, sino también el criterio de comparación y las restricciones correspondientes para evitar derrotadores falaces, toda esta información será agregada utilizando los operadores correspondientes.

El *SRLA* proveerá tres operadores específicos:

- Operador de Programa DeLP $+_{delp}$,
- Operador de Actualización de Criterio de Comparación $*_{\succ}$, y
- Operador de Restricción de Derrotadores $-_{\mathcal{D}}$.

El primer operador permitirá agregar un programa DeLP al *SRLA*, el cuál será utilizado para calcular los argumentos y los derrotadores de argumentos. El segundo operador especificará el criterio de comparación a utilizar, reemplazando el existente, cuando corresponda. Finalmente, el último operador agregará al *SRLA* las restricciones correspondientes para evitar la construcción de derrotadores falaces.

Un cliente interesado en obtener respuestas a consultas DeLP en un programa DeLP puede consultar a el *SRLA*: $\mathcal{SR}_{la} = \langle \mathcal{I}_{lp}, \{+_{delp}, *_{\succ}, -_{\mathcal{D}}\}, \mathcal{P}_{Idelp} \rangle$. El cliente deberá enviar

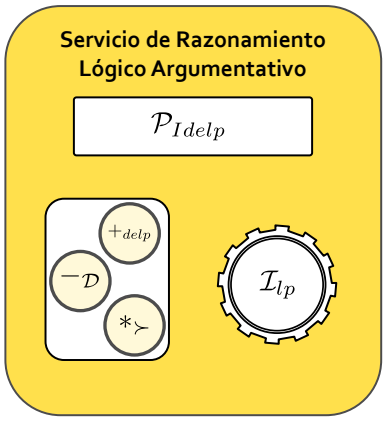

Figura 5.4: Servicio de Razonamiento Lógico Argumentativo

al *SRLA* como parte del contexto el programa DeLP para que el servicio pueda computar los argumentos en favor de su consulta DeLP.

Ejemplo 5.1. *Un cliente Ag_{delpd} puede estar interesado en determinar si una consulta DeLP está garantizada o no a partir del programa DeLP $\mathcal{P}_{arb}$, mostrado en la Figura 5.5. El cliente Ag_{delpd} puede desear utilizar el criterio de comparación de argumentos denominado* especificidad *para llevar a cabo la comparación entre argumentos. Dicho criterio de comparación de argumentos, introducido originalmente en [Gar01], será representado por el programa lógico $\mathcal{P}_{esp}$. Tanto una descripción de la semántica de este criterio de comparación como una explicación de la forma en la cuál se implementa el mismo están fuera del alcance de esta tesis. Lo mismo ocurre con la utilización de otros criterios de comparación. Cabe destacar que la implementación modular que utiliza esta aplicación de propósito general permite la utilización de distintos criterios de comparación sin mayores inconvenientes.*

$$\mathcal{P}_{arb} = \left\{ \begin{array}{lll} a \prec b, c. & c \prec h. & d. \quad h. \\ b \prec d. & \sim c \prec h, i. & e. \quad i. \\ \sim b \prec d, e. & c \prec j, k. & f. \quad j. \\ b \prec f, g. & & g. \quad k. \end{array} \right\}$$

Figura 5.5: Programa DeLP $\mathcal{P}_{arb}$

El cliente Ag_{delpd} puede consultar a SRLA en busca de argumentos que soporten la consulta DeLP a en el programa $\mathcal{P}_{arb}$. En ese caso, debería ejecutar la siguiente CC:

$$\mathcal{CC}_{5.2} = [\langle(\mathcal{P}_{arb}, +_{delp})\rangle, argumento(a, Arg)].$$

La respuesta para esta CC por parte del SRLA será:

$$\langle argumento(a, \mathcal{A}_1), argumento(a, \mathcal{A}_2), argumento(a, \mathcal{A}_3), argumento(a, \mathcal{A}_4)\rangle$$

donde cada argumento está formado de la siguiente manera:

$$\mathcal{A}_1 = \left\{ \begin{array}{l} a \prec b, c \\ b \prec d \\ c \prec h \end{array} \right\} \quad \mathcal{A}_2 = \left\{ \begin{array}{l} a \prec b, c \\ b \prec d \\ c \prec j, k \end{array} \right\} \quad \mathcal{A}_3 = \left\{ \begin{array}{l} a \prec b, c \\ b \prec f, g \\ c \prec h \end{array} \right\} \quad \mathcal{A}_4 = \left\{ \begin{array}{l} a \prec b, c \\ b \prec f, g \\ c \prec j, k \end{array} \right\}$$

□

Luego de obtener argumentos en favor de su consulta, el cliente enviará nuevas *CCs* al *SRLA* en busca de derrotadores para los argumentos obtenidos. En este caso, además de enviar el programa DeLP como parte del contexto, deberá enviar también el criterio de comparación entre argumentos así como también las restricciones relativas a la línea de argumentación, para que el *SRLA* pueda construir argumentos derrotadores que no sean falaces.

Ejemplo 5.2. *El cliente Ag_{delpd} obtuvo argumentos en favor de la consulta DeLP y deberá buscar derrotadores para estos argumentos. En el caso del argumento $\mathcal{A}_1$, el cliente puede consultar al SRLA en busca de derrotadores de la siguiente manera:*

$$\mathcal{CC}_{5.2} = [\langle(\mathcal{P}_{arb}, +_{delp}), (\mathcal{P}_{esp}, *_{\succ})\rangle, derrotador(\mathcal{A}_1, \mathcal{D})].$$

La CC incluye el criterio de comparación de argumentos especificidad. *En cuanto a las restricciones, hasta el momento no es necesario incluir ningún dato extra debido a que toda la información requerida se encuentra en la consulta misma.*

La respuesta para esta CC corresponde a la secuencia:

$$\langle derrotador(\mathcal{A}_1, \mathcal{A}_5), derrotador(\mathcal{A}_1, \mathcal{A}_6)\rangle$$
$$\mathcal{A}_5 = \{\sim c \prec h, i\} \qquad \mathcal{A}_6 = \{\sim b \prec d, e\}$$

Indicando que tanto el argumento $\mathcal{A}_5$ como el argumento $\mathcal{A}_6$ derrotan a $\mathcal{A}_1$. □

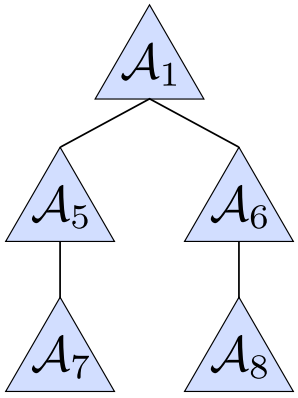

Figura 5.6: Árbol de Dialéctica para $\mathcal{A}_1$

De esta forma se obtienen nuevos argumentos que derrotan a los argumentos en favor de la consulta. A su vez, hay que continuar la búsqueda de nuevos derrotadores para estos argumentos hasta completar todas las líneas de argumentación, las cuales conformarán el árbol de dialéctica. En el ejemplo, para el argumento $\mathcal{A}_1$, se formará el árbol de dialéctica mostrado en la Figura 5.6.

Cabe aclarar que, en cada *CC* en busca de derrotadores se debe incluir la información respectiva a la línea de argumentación. En el caso particular del argumento que se está tratando de derrotar, si bien el argumento es parte de esta línea, dicho argumento se encuentra implícitamente incluido en la línea de argumentación y es omitido como restricción. En el ejemplo, para el caso del argumento $\mathcal{A}_5$ que derrota a $\mathcal{A}_1$, se debe incluir la línea de argumentación correspondiente, al momento de buscar nuevos derrotadores, la cuál corresponde en este caso al argumento $\mathcal{A}_1$.

Lo explicado anteriormente se debe repetir para todos los argumentos en favor de la consulta DeLP, generando de esta forma un árbol de dialéctica para cada argumento. A partir de este análisis dialéctico se podrá determinar si la consulta DeLP está garantizada o no en el programa DeLP correspondiente. En el ejemplo introducido, además del árbol de dialéctica ya mostrado, se generarán tres nuevos árboles de dialéctica, mostrados en la Figura 5.7, asociados a los argumentos restantes ($\mathcal{A}_2, \mathcal{A}_3$ y $\mathcal{A}_4$). Considerando que la construcción de estos árboles de dialéctica y la de las líneas de argumentación que forman los distintos árboles son procesos independientes entre sí, es posible realizar dichos cómputos en paralelo sin ningún tipo de requerimiento de sincronización entre ellos.

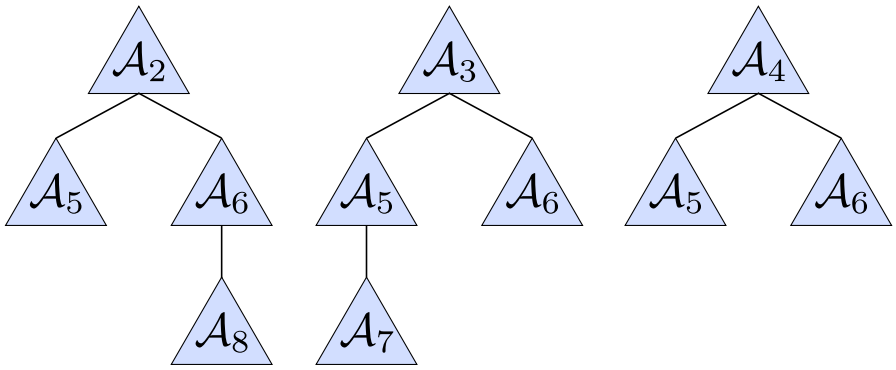

Figura 5.7: Árboles de Dialéctica para $\mathcal{A}_2$, $\mathcal{A}_3$ y $\mathcal{A}_4$

En la siguiente sección se mostrarán los tipos de consultas contextuales que son necesarias para, a partir de un *GR* y dichas consultas contextuales, obtener el estado de garantía para una consulta DeLP en un programa DeLP. Como se mostrará, este tipo de consultas contextuales permitirán paralelizar el cómputo de los argumentos en distintos *SRLA*s.

5.3. Consultas Contextuales para DeLP en paralelo

En esta sección se mostrarán cuáles de las distintas consultas contextuales definidas a lo largo de esta tesis serán utilizadas por Ag_{delpd} para obtener la construcción de los árboles de dialéctica en paralelo. Como se explicó previamente, un cliente con el objetivo de computar el estado de garantía de una consulta DeLP, debe ejecutar una serie de consultas contextuales para construir los árboles de dialéctica correspondientes. Continuando con el ejemplo introducido en la sección anterior, el cliente Ag_{delpd} puede desear computar el estado de garantía de a en el programa $\mathcal{P}_{arb}$ utilizando el *GRE* mostrado en la Figura 5.3.

Para iniciar el cómputo de la garantía de una consulta DeLP, un cliente deberá ejecutar una *CC* en busca de argumentos para su consulta. Una vez realizado esto, el cliente obtendrá todos los argumentos en favor de la consulta DeLP, los cuáles corresponden a las raíces de los árboles de dialéctica para su consulta. En el caso del ejemplo utilizado a lo largo de este capítulo, el cliente Ag_{delpd} debe ejecutar la consulta contextual $\mathcal{CC}_{5.2}$ introducida en la Figura 5.8. Como se puede ver en dicha figura, la respuesta $\mathcal{R}$ obteni-

$$\mathcal{CC} = [\langle(\mathcal{P}_{arb}, +_{delp})\rangle, argumento(a, Arg)].$$

$$\mathcal{R} = \langle argumento(a, \mathcal{A}_1), argumento(a, \mathcal{A}_2), argumento(a, \mathcal{A}_3), argumento(a, \mathcal{A}_4)\rangle$$

Figura 5.8: Consulta Contextual con su correspondiente Respuesta

da contiene cuatro argumentos, los cuáles generarán la misma cantidad de árboles de dialéctica.

$$\mathcal{C}o = \langle(\mathcal{P}_{arb}, +_{delp}), (\mathcal{P}_{esp}, *_{\succ})\rangle$$

$$\begin{aligned} \mathcal{C}_1 &= [\mathcal{C}o, derr(\mathcal{A}_1, \mathcal{D})] \\ \mathcal{C}_2 &= [\mathcal{C}o, derr(\mathcal{A}_2, \mathcal{D})] \\ \mathcal{C}_3 &= [\mathcal{C}o, derr(\mathcal{A}_3, \mathcal{D})] \\ \mathcal{C}_4 &= [\mathcal{C}o, derr(\mathcal{A}_4, \mathcal{D})] \end{aligned}$$

(a)

$$CMF_{14} = \left[\mathcal{C}o, \left\langle \begin{array}{l} derr(\mathcal{A}_1, \mathcal{D}), \\ derr(\mathcal{A}_2, \mathcal{D}), \\ derr(\mathcal{A}_3, \mathcal{D}), \\ derr(\mathcal{A}_4, \mathcal{D}) \end{array} \right\rangle\right]$$

(b)

Figura 5.9: Aplicación de *CCMFs*

Por cada argumento obtenido se debe iniciar el proceso de búsqueda de derrotadores, el cuál resultará en la generación de los distintos árboles de dialéctica. En el ejemplo utilizado en este capítulo, se deben buscar derrotadores para los cuatro argumentos en favor de la consulta "a" hasta llegar a formar los árboles de dialéctica ya mostrados en las Figuras 5.6 y 5.7. Para buscar derrotadores de un argumento, un cliente debe ejecutar las *CCs* que se muestran en la Figura 5.9 (a). Como se puede observar, cada uno de las consultas posee el mismo contexto, por lo tanto, es posible utilizar una única *CCMF* incluyendo todas estas consultas. En el caso del ejemplo empleado, en la Figura 5.9 (b) se muestra como esta compuesta la *CCMF* correspondiente.

Al momento de recibir esta *CCMF*, el *GRE* debe decidir como llevar a cabo el cómputo de la respuesta. Para esto, el *GRE* tiene la posibilidad de distribuir o no el cómputo de la respuesta de acuerdo a la cantidad de *SRLA*s incluidos en el *GRE* y a la información que posea con respecto a la disponibilidad de los mismos. Las posibilidades, como se muestra

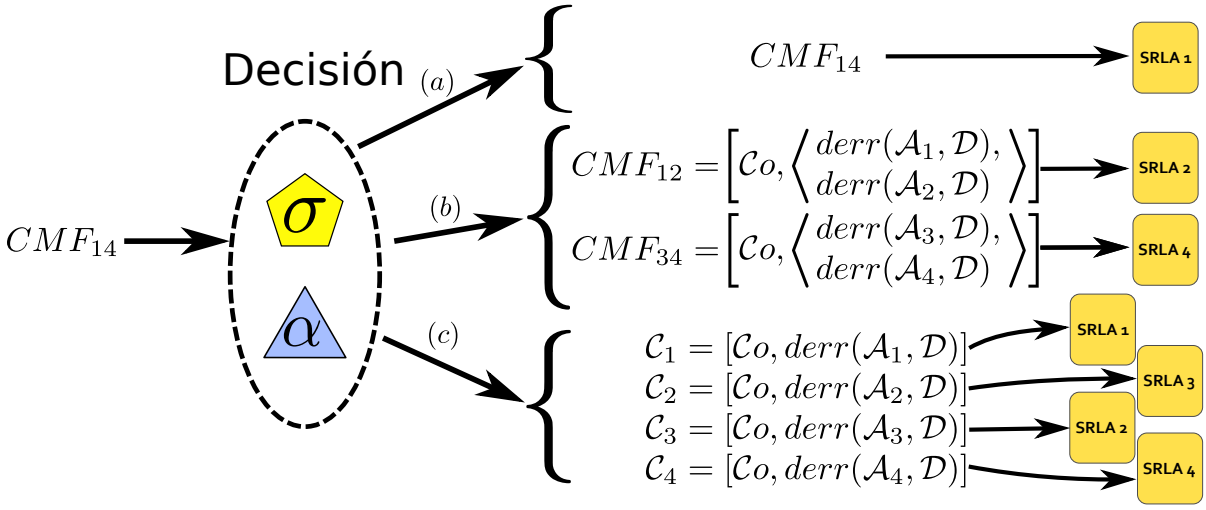

Figura 5.10: Alternativas del *GRE* ante una *CCMF*

en la Figura 5.10, van desde enviar la *CCMF* a un único *SRLA* (a), hasta la separación de la *CCMF* en tantas *CCs* como consultas incluya la *CCMF* (c), pasando por distintas posibles combinaciones intermedias (b).

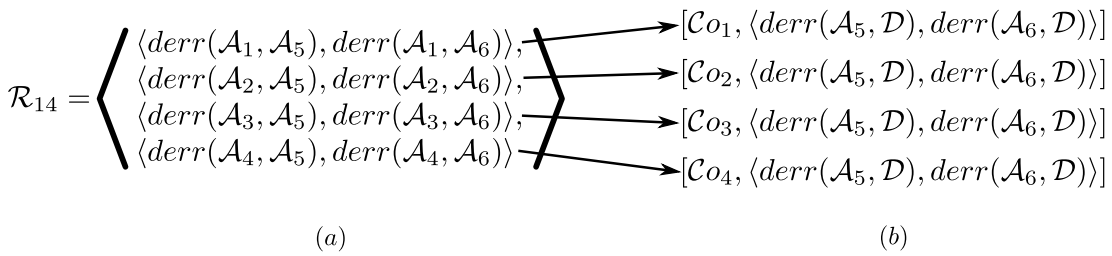

Figura 5.11: Generación de *CCMFs*

La respuesta a la *CCMF* incluirá una secuencia de derrotadores por cada consulta enviada. Como se muestra en la Figura 5.11 (a), el proceso continúa buscando nuevos derrotadores para cada uno de los derrotadores obtenidos, con la particularidad de que cada secuencia de derrotadores comparte la línea de argumentación y, por lo tanto, el contexto para la consulta que computará estos nuevos derrotadores. Por esta razón, el

cliente podría generar una nueva *CCMF* por cada secuencia de derrotadores obtenida (ver Figura 5.11 (*b*)).

$$[\mathcal{C}o_1, \langle derr(\mathcal{A}_5, \mathcal{D}), derr(\mathcal{A}_6, \mathcal{D})\rangle]$$
$$[\mathcal{C}o_2, \langle derr(\mathcal{A}_5, \mathcal{D}), derr(\mathcal{A}_6, \mathcal{D})\rangle]$$
$$[\mathcal{C}o_3, \langle derr(\mathcal{A}_5, \mathcal{D}), derr(\mathcal{A}_6, \mathcal{D})\rangle]$$
$$[\mathcal{C}o_4, \langle derr(\mathcal{A}_5, \mathcal{D}), derr(\mathcal{A}_6, \mathcal{D})\rangle]$$

(*a*)

$$ICM_{56} = \left\langle \begin{array}{l} [(\mathcal{C}o_1, derr(\mathcal{A}_5, \mathcal{D})), (\langle\rangle, derr(\mathcal{A}_6, \mathcal{D}))], \\ [(\mathcal{C}o_2, derr(\mathcal{A}_5, \mathcal{D})), (\langle\rangle, derr(\mathcal{A}_6, \mathcal{D}))], \\ [(\mathcal{C}o_3, derr(\mathcal{A}_5, \mathcal{D})), (\langle\rangle, derr(\mathcal{A}_6, \mathcal{D}))], \\ [(\mathcal{C}o_4, derr(\mathcal{A}_5, \mathcal{D})), (\langle\rangle, derr(\mathcal{A}_6, \mathcal{D}))] \end{array} \right\rangle$$

(*b*)

Figura 5.12: Aplicación de *ICMs*

Teniendo en cuenta la propiedad 3.4 que indica que para cada *CCMF* existe una *IC* equivalente, entonces, con todas las *ICs* obtenidas (ver Figura 5.12 (*a*)) es posible definir una *ICM* que incluya a todas las *CCMFs*(ver Figura 5.12 (*b*)), con el objetivo de construir una única consulta para enviarle al *GRE*. Esta alternativa genera que el *GRE* reciba todas estas consultas al mismo tiempo, permitiéndole tomar una mejor decisión de cómo distribuirlas en los *SRLA*s disponibles.

La *ICM* obtendrá como respuesta una secuencia de secuencias de respuestas (ver Figura 5.13 (*a*)), donde cada respuesta obtenida incluirá los derrotadores de un determinado argumento. En los casos donde la respuesta no incluye ningún derrotador, estamos en presencia de un nodo hoja y el argumento correspondiente se encuentra no derrotado (observar esto en las Figuras 5.6 y 5.7). Cuando se encuentra al menos un derrotador, se debe continuar con el procedimiento en busca de nuevos derrotadores y, por lo tanto, se debe generar una nueva *IC* (simulando una *CCMF*). De esta forma, se puede procesar cada respuesta obtenida e ir generando las nuevas *ICs* que formarán la *ICM* a enviar al *GRE*. Este proceso deberá continuar mientras sigan apareciendo derrotadores para los argumentos.

En la Figura 5.13 (*a*) se puede ver la respuesta obtenida a la consulta $\mathcal{ICM}_{56}$ mostrada en la Figura 5.12. En esta respuesta, podemos observar que el argumento $\mathcal{A}_5$ de la primer *IC* se encuentra derrotado por el argumento $\mathcal{A}_7$. En cambio, para el mismo argumento $\mathcal{A}_5$, en la segunda *IC*, no se encontraron derrotadores. A partir de estas respuestas, es posible generar la interrogación contextual múltiple $\mathcal{ICM}_{78}$ mostrada en la Figura 5.13 (*b*), en busca de derrotadores para los argumentos nuevos. Es claro que el contexto del

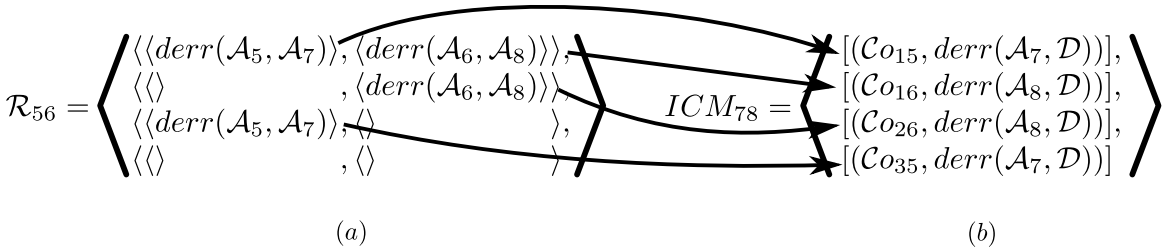

Figura 5.13: Respuesta para la Interrogación Contextual Múltiple y generación de la nueva

argumento $\mathcal{A}_7$ encontrado en la primer IC no es el mismo al del propio argumento $\mathcal{A}_7$ encontrado en la tercer IC.

Al momento de recibir una ICM, un *GRE* puede, al igual que con la $CCMF$, distribuir o no el cómputo de la respuesta a la misma. Las alternativas son básicamente las mismas que antes, con la diferencia que ahora existe un nuevo nivel donde es factible la decisión de distribuir o no. Es decir, no sólo es posible realizar la separación natural de la ICM en ICs, sino que además, las propias ICs pueden ser separadas en CC. En cualquiera de los casos, el *GRE* recibe todas las consultas juntas, lo cuál le permitirá tomar la mejor decisión, dependiendo de los *SRLA*s disponibles.

En la siguiente sección se presentará un esquema de algoritmo que, utilizando los distintos tipos de consultas contextuales mostradas en esta sección, permitirá generar los árboles de dialéctica en paralelo para una consulta DeLP en un programa DeLP a partir de un *GRE*, con el objetivo de obtener el estado de garantía de la consulta DeLP. Para lograr esto, el *GRE* distribuirá el cómputo de los argumentos (las respuestas a las consultas) utilizando un esquema similar al mencionado en la Figura 5.2.

5.4. Intérprete DeLP Paralelo

En el caso de contar con un *GRE* Homogéneo, formado por *SRLA*s, un cliente puede calcular en paralelo el estado de garantía de una consulta DeLP en un determinado programa DeLP. En esta sección se introduce un esquema de algoritmo que permite al cliente ejecutar de manera secuencial las correspondientes CCM. Es importante destacar que de

esta manera se está delegando en el *GRE* la responsabilidad de distribuir de la mejor manera posible el cálculo de las respuesta en los distintos servicios de razonamiento.

Con el objetivo de computar el estado de garantía de una consulta DeLP Q a partir de un programa DeLP $\mathcal{P}_{delp}$, se deberá:

1. Ejecutar la $\mathcal{CC}_1$ en busca de los argumentos de soporte de la consulta DeLP Q.

$$\mathcal{CC}_1 = [\langle(\mathcal{P}_{delp}, +_{delp})\rangle, argumento(Q, \mathcal{A})]$$

de la cual se obtendrá la respuesta

$$\mathcal{R}_1 = \langle argumento(Q, \mathcal{A}_1), argumento(Q, \mathcal{A}_2), \ldots, argumento(Q, \mathcal{A}_n)\rangle.$$

2. Ejecutar la $\mathcal{CCMF}_2$ en busca de derrotadores para cada argumento obtenido en $\mathcal{R}_1$.

$$\mathcal{CCMF}_2 = \left[\begin{array}{c} \langle(\mathcal{P}_{delp}, +_{delp}), (\mathcal{P}_{crit}, *_{\succ})\rangle, \\ \langle derrotador(\mathcal{A}_1, \mathcal{D}_{1\_}), derrotador(\mathcal{A}_2, \mathcal{D}_{2\_}), \ldots, derrotador(\mathcal{A}_n, \mathcal{D}_{n\_})\rangle \end{array}\right]$$

de lo cual se obtendrá la respuesta

$$\mathcal{R}_2 = \left\langle \begin{array}{c} \langle derrotador(\mathcal{A}_1, \mathcal{D}_{11}), derrotador(\mathcal{A}_1, \mathcal{D}_{12}), \ldots, derrotador(\mathcal{A}_1, \mathcal{D}_{1m})\rangle, \\ \langle derrotador(\mathcal{A}_2, \mathcal{D}_{21}), derrotador(\mathcal{A}_2, \mathcal{D}_{22}), \ldots, derrotador(\mathcal{A}_2, \mathcal{D}_{2p})\rangle, \\ \ldots, \\ \langle derrotador(\mathcal{A}_n, \mathcal{D}_{n1}), derrotador(\mathcal{A}_n, \mathcal{D}_{n2}), \ldots, derrotador(\mathcal{A}_n, \mathcal{D}_{nr})\rangle. \end{array} \right\rangle$$

3. Por cada secuencia de derrotadores obtenida en $\mathcal{R}_2$ se generará una IC equivalente a una $CCMF$. Las distintas ICs conformarán una ICM buscando derrotadores para los derrotadores obtenidos previamente.

$$\mathcal{ICM}_3 = \left\langle \begin{array}{c} \left[\begin{array}{c} (\langle(\mathcal{P}_{delp}, +_{delp}), (\mathcal{P}_{crit}, *_{\succ}), (\{\mathcal{A}_1\}, -_{\mathcal{D}})\rangle, derrotador(\mathcal{D}_{11}, \mathcal{D}_{11\_})), \\ (\langle\rangle, derrotador(\mathcal{D}_{12}, \mathcal{D}_{12\_})), \ldots, (\langle\rangle, derrotador(\mathcal{D}_{1m}, \mathcal{D}_{1m\_})) \end{array}\right], \\ \left[\begin{array}{c} (\langle(\mathcal{P}_{delp}, +_{delp}), (\mathcal{P}_{crit}, *_{\succ}), (\{\mathcal{A}_2\}, -_{\mathcal{D}})\rangle, derrotador(\mathcal{D}_{21}, \mathcal{D}_{21\_})), \\ (\langle\rangle, derrotador(\mathcal{D}_{22}, \mathcal{D}_{22\_})), \ldots, (\langle\rangle, derrotador(\mathcal{D}_{2p}, \mathcal{D}_{2p\_})) \end{array}\right], \\ \ldots, \\ \left[\begin{array}{c} (\langle(\mathcal{P}_{delp}, +_{delp}), (\mathcal{P}_{crit}, *_{\succ}), (\{\mathcal{A}_n\}, -_{\mathcal{D}})\rangle, derrotador(\mathcal{D}_{n1}, \mathcal{D}_{n1\_})), \\ (\langle\rangle, derrotador(\mathcal{D}_{n2}, \mathcal{D}_{n2\_})), \ldots, (\langle\rangle, derrotador(\mathcal{D}_{nr}, \mathcal{D}_{nr\_})) \end{array}\right] \end{array} \right\rangle$$

de lo cual se obtendrá una respuesta $\mathcal{R}_3$ incluyendo una secuencia de respuestas análogas a $\mathcal{R}_2$.

4. Repetir el punto 3
   generando la $\mathcal{ICM}_i$ a partir de la $\mathcal{R}_{i-1}$
   hasta que la respuesta $\mathcal{R}_i$ no contenga derrotadores.

Como se mostrará en el algoritmo de la Sección 5.5, el cliente irá guardando los argumentos obtenidos, así como también los derrotadores, formando de esta manera los distintos árboles de dialéctica. De acuerdo al esquema de algoritmo presentado, estos árboles de dialéctica se calcularán por niveles, cada uno de los cuales estará asociado a una consulta contextual distinta. El paralelismo en el cálculo de la garantía surge de la ejecución de las *CCMF* y de las *ICM*, las cuales puede ser distribuidas automáticamente por el *GRE* en distintos *SRs* (Puntos 2 y 3 del esquema de algoritmo).

Ejemplo 5.3. *En este ejemplo mostraremos como el cliente Ag_{delpd}, introducido previamente en este capítulo, puede utilizar el GRE mostrado en la Figura 5.3 para determinar el estado de garantía de la consulta DeLP a en el programa DeLP $\mathcal{P}_{arb}$ implementando el esquema de algoritmo presentado previamente.*

El cliente Ag_{delpd} ejecutará, de acuerdo al Punto 1 del algoritmo, la consulta contextual $\mathcal{CC}$ mostrada en la Figura 5.8, buscando argumentos en favor de la consulta DeLP a. Luego, ejecutará la consulta contextual múltiple factorizada $\mathcal{CCMF}_{14}$ mostrada en la Figura 5.9, siguiendo el Punto 2 del algoritmo, en busca de derrotadores para cada uno de los argumentos $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ y $\mathcal{A}_4$ obtenidos en la respuesta $\mathcal{R}$ a la consulta anterior. Dichos argumentos serán incluidos en la CCMF como parte de la secuencia de consultas.

Como se puede observar en la Figura 5.10, el GRE tendrá la posibilidad de separar el cómputo de la respuesta a la consulta $\mathcal{CCMF}_{14}$ en los distintos SRLAs que conforman el GRE. Considerando que estos SRLAs son homogéneos, la decisión del GRE tendrá como principal objetivo reducir el tiempo requerido para computar la respuesta. En este escenario, la mejor alternativa corresponde a dividir la CCMF en cuatro consultas contextuales y enviarle una consulta a cada SRLA.

La $\mathcal{CCMF}_{14}$ muestra uno de los lugares donde se obtiene paralelismo, es decir, en el cómputo de la respuesta a la $\mathcal{CCMF}_{14}$. Una posible distribución de dicho cómputo se muestra en la Figura 5.14, donde el cálculo de los argumentos que derrotan a $\mathcal{A}_1$ se

realizó en el Nodo 1, los que derrotan a $\mathcal{A}_2$ en el Nodo 2 y así siguiendo. Por cada respuesta obtenida con al menos un argumento se generará una IC equivalente a una CCMF en busca de derrotadores a los argumentos encontrados. Las ICs se agruparán en la interrogación contextual múltiple $\mathcal{ICM}_{56}$ mostrada en la Figura 5.12, la cual será enviada al GRE siguiendo el Punto 3.

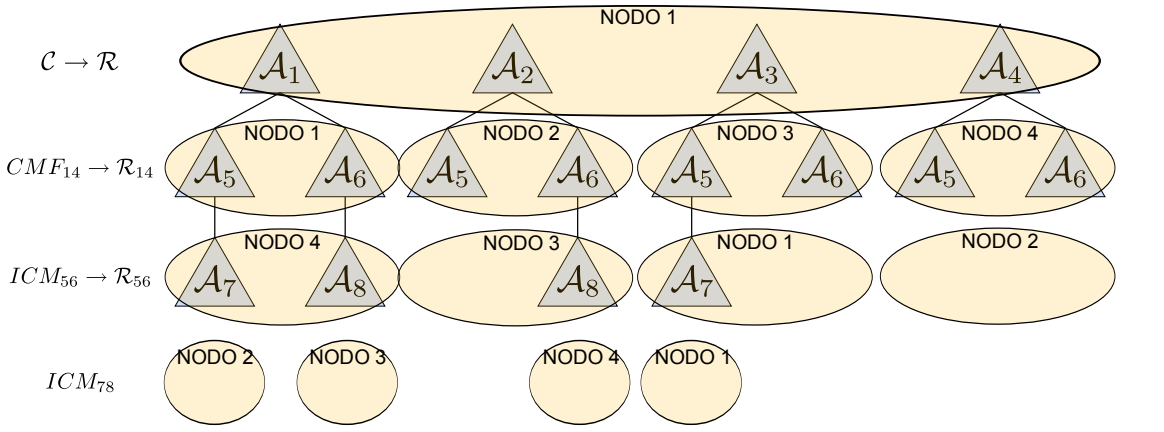

Figura 5.14: Distribución del cómputo de los argumentos y contraargumentos para el Ejemplo 5.5 en el *GRE* mostrado en la Figura 5.3

La generación de esta ICM presenta una nueva oportunidad para obtener paralelismo. El GRE es el encargado de distribuir el cómputo de la respuesta en los distintos SRLAs. La ICM generada incluye cuatro IC, cada una de ellas incluyendo dos argumentos a derrotar. Esta consulta es enviada al GRE en busca de derrotadores para los derrotadores de los argumentos en favor de a. Una posible distribución del cómputo de las respuestas correspondientes puede verse en la Figura 5.14, donde la separación de la interrogación contextual múltiple $\mathcal{ICM}_{56}$ tiene como resultado las cuatro ICs que la forman, considerando que se cuenta con la misma cantidad de SRLAs.

Finalmente, con la respuesta $\mathcal{R}_{56}$ se obtienen cuatro nuevos derrotadores, y por lo tanto, se repite el Punto 3 del esquema de algoritmo y el cliente Ag_{delpd} ejecuta la interrogación contextual múltiple $\mathcal{ICM}_{78}$, mostrada en la Figura 5.13. Esta nueva consulta estará formada nuevamente por cuatro ICs, las cuáles serán distribuidas por el GRE. En la Figura 5.14 se muestra una de las posibles distribuciones. Esta ICM no obtiene der-

rotadores y, por lo tanto, finaliza el cómputo de los árboles de dialéctica de la consulta "a". □

En esta sección se introdujo un esquema de algoritmo que, utilizando las consultas contextuales mencionadas en la sección anterior, computa los árboles de dialéctica que genera una consulta DeLP. Además, se mostró un ejemplo de las consultas contextuales que puede realizar un cliente y las respuestas que obtendrá del *GRE*. En la siguiente sección se presentará un algoritmo concreto, construido en base al esquema de algoritmo introducido previamente.

5.5. Algoritmo DeLP Paralelo

En esta sección se presentará el algoritmo propuesto para implementar un intérprete de DeLP en Paralelo utilizando las consultas contextuales y los *GREs*.

Datos de Entrada:

- Q (Consulta DeLP),
- $\mathcal{P}_{delp}$ (Programa DeLP),
- $\mathcal{P}_{crit}$ (Criterio de Comparación) y
- $\mathcal{GRE} = \langle \mathbf{SR}_{la}, \sigma, \alpha \rangle$ donde $\mathbf{SR}_{la}$ estará compuesto por varios $\mathcal{SR}_{la}$, siendo cada uno de estos $\mathcal{SR}_{la} = \langle \mathcal{I}_{lp}, \{+_{delp}, *_{\succ}, -_{\mathcal{D}}\}, \mathcal{P}_{Idelp} \rangle$.

Datos de Salida: $LArb$ (Lista de Árboles de dialéctica).

1) Primer Paso

   I) Armar la Consulta Contextual $\mathcal{CC}_1 = [\langle (\mathcal{P}_{delp}, +_{delp}) \rangle, arg(Q, \mathcal{A})]$.

   II) Enviar $\mathcal{CC}_1$ a $\mathcal{GRE}$.

   III) Esperar la Respuesta $\mathcal{R}_1 = \langle arg(Q, \mathcal{A}_1), arg(Q, \mathcal{A}_2), \ldots, arg(Q, \mathcal{A}_n) \rangle$

   IV) Agregar una raíz de un árbol a $LArb$ por cada $\mathcal{A}_i$ obtenido en $\mathcal{R}$, con $1 \leq i \leq n$.

2) Segundo Paso

I) A partir de $\mathcal{R}_1$, armar la Consulta Contextual Múltiple Factorizada
$\mathcal{CCMF}_2 = \left[\begin{array}{c} \langle(\mathcal{P}_{delp}, +_{delp}), (\mathcal{P}_{crit}, *_{\succ})\rangle, \\ \langle derr(\mathcal{A}_1, \mathcal{D}_{1\_}), derr(\mathcal{A}_2, \mathcal{D}_{2\_}), \ldots, derr(\mathcal{A}_n, \mathcal{D}_{n\_})\rangle \end{array} \right]$

II) Enviar $\mathcal{CCMF}_2$ a $\mathcal{GRE}$.

III) Esperar la Respuesta $\mathcal{R}_2 = \langle S_1, S_2, \ldots, S_n\rangle$, donde para cada S_i, $1 \leq i \leq n$ existe un m tal que $S_i = \langle derr(\mathcal{A}_i, \mathcal{D}_{i1}), derr(\mathcal{A}_i, \mathcal{D}_{i2}), \ldots, derr(\mathcal{A}_i, \mathcal{D}_{im})\rangle$.

IV) Para cada secuencia S_i, $1 \leq i \leq n$ tal que $m \neq 0$, siendo $m = |S_i|$ la cantidad de argumentos en la secuencia S_i, agregar los argumentos $\mathcal{D}_{ij}$ en S_i, $1 \leq j \leq m$ como hijos de $\mathcal{A}_i$ en *LArb*.

3) Tercer Paso

I) Armar la Interrogación Contextual Múltiple $\mathcal{ICM}_p = \langle \mathcal{IC}_1, \mathcal{IC}_2, \ldots, \mathcal{IC}_n\rangle$, con n la cantidad de secuencias de respuestas obtenidas en $\mathcal{R}_{p-1}$ y

$$\mathcal{IC}_i = \left[\begin{array}{c} (\langle(\mathcal{P}_{delp}, +_{delp}), (\mathcal{P}_{crit}, *_{\succ}), (LA_i, -_{\mathcal{D}})\rangle, derr(\mathcal{D}_{i1}, \mathcal{D}_{i1\_})), \\ (\langle\rangle, derr(\mathcal{D}_{i2}, \mathcal{D}_{i2\_})), \ldots, (\langle\rangle, derr(\mathcal{D}_{im}, \mathcal{D}_{im\_})) \end{array} \right]$$

con $1 \leq i \leq n$, siendo $|S_i| = m$ y S_i una de las secuencias de la respuesta anterior y LA_i la línea de argumentación correspondiente al argumento derrotado por dicha secuencia.

II) Enviar $\mathcal{ICM}_p$ a $\mathcal{GRE}$.

III) Esperar la Respuesta $\mathcal{R}_{pa} = \langle S_1, S_2, \ldots, S_n\rangle$, donde cada S_i, $1 \leq i \leq n$ es una secuencia $S_i = \langle S_{i1}, S_{i2}, \ldots, S_{im}\rangle$ de secuencias de respuestas para algún m.

IV) Aplanar la respuesta $\mathcal{R}_{pa}$ para generar una nueva respuesta $\mathcal{R}_p$ siendo una única secuencia de secuencias de respuestas tal que:

$$\mathcal{R}_p = \langle S_{11}, S_{12}, \ldots, S_{1m}, S_{21}, S_{22}, \ldots, S_{2p}, \ldots, S_{n1}, S_{n2}, \ldots, S_{nr}\rangle.$$

V) Para cada secuencia S_{ij} de $\mathcal{R}_p$ tal que $|S_{ij}| = m$, si $m \neq 0$, agregar los argumentos $\mathcal{D}_{ijk}$, $1 \leq k \leq m$ como hijos de $\mathcal{D}_{ij}$ en *LArb*.

VI) Mientras exista al menos un S_{ij} en $\mathcal{R}_p$ tal que $|S_{ij}| \neq 0$, repetir el punto 3.

5.6. Análisis del Algoritmo

El algoritmo presentado en este capítulo representa una alternativa a la hora de computar la garantía de una consulta DeLP en paralelo. Esta alternativa posee como principales ventajas que su implementación por parte del cliente resulta simple y, además, minimiza tanto el uso de la infraestructura de comunicación como la cantidad de mensajes intercambiados. Por otro lado, al recibir todas las consultas agrupadas en una consulta contextual extendida, el *GRE* tiene la posibilidad de distribuir mejor el cómputo de las mismas en los *SRLA* disponibles. Sin embargo, la utilización de una única consulta contextual extendida para el cálculo de los argumentos de un nivel completo de los árboles de dialéctica genera un efecto de barrera innecesario en el cálculo de las distintas líneas de argumentación, considerando que estas líneas son independientes entre sí.

Dentro de las posibilidades que ofrecen las consultas contextuales y los *GRE*, una posible modificación al algoritmo propuesto consiste en reemplazar cada ICM por diversas *CCMFs* ejecutadas de manera asincrónica. En esta nueva alternativa, la utilización de múltiples consultas de manera separada elimina, en parte, el efecto de barrera del esquema anterior permitiendo que el cálculo de las líneas de argumentación avance de manera independiente, en vez de por niveles. Sin embargo, la complejidad en la implementación de los clientes aumenta al requerir la ejecución de consultas asincrónicas y el manejo de respuestas en cualquier instante. Además, el *GRE* recibe consultas permanentemente, complicando su tarea de distribución del cómputo de las respuestas.

Por otro lado, el algoritmo presentado originalmente posee la propiedad de computar los árboles de dialéctica de manera completa. Sin embargo, como fue demostrado en [Gar01], para obtener el estado de garantía de una determinada consulta DeLP no es necesario computar todos los árboles de manera completa. Es decir, es posible optimizar el algoritmo presentado, incluyendo la poda de argumentos en la construcción de los árboles de dialéctica, puntualmente en el momento en que es posible determinar el estado de un nodo específico. Cuando no se encuentran derrotadores para un dado argumento, el nodo correspondiente se encuentra *"no derrotado"* y, por lo tanto, su padre adquiere el estado de *"derrotado"* y es por esta razón que se puede frenar el cómputo de la línea de argumentación cuya hoja es el argumento no derrotado.

Es claro que existen aún más alternativas a la hora de implementar DeLP de manera distribuida, incluso sin utilizar las consultas contextuales introducidas en en los capítulos

anteriores. Cualquier análisis y comparación de estas otras alternativas con la presentada en esta sección escapa al alcance de esta tesis. El principal objetivo de este capítulo es mostrar cómo las consultas contextuales y los *GRE* permiten distribuir, automáticamente y eficientemente, el cómputo de los árboles de dialéctica de DeLP, los cuales corresponden a la herramienta principal de prueba utilizada por el intérprete DeLP para determinar el estado de garantía de una consulta DeLP.

5.7. Conclusiones

En el presente capítulo se presentó un ejemplo concreto de aplicación donde se muestra cómo es posible obtener paralelismo utilizando las consultas contextuales y *GRE* Homogéneos. En particular, se introdujo un algoritmo que permite generar los árboles de dialéctica de una consulta DeLP de manera distribuida con el objetivo de calcular el estado de garantía de dicha consulta.

Una de las características que se puede resaltar del intérprete DeLP paralelo introducido corresponde a la simplicidad con la cual se obtiene el paralelismo. Esta simplicidad está directamente relacionada con la utilización de consultas contextuales extendidas y *GREs* formados por *SRs* especialmente diseñados para generar argumentos. Además, el hecho de utilizar consultas contextuales permite que distintos agentes empleen el mismo *GRE* simultáneamente para calcular el estado de garantía de distintas consultas DeLP.

Capítulo 6

Propuesta de Implementación

Los conceptos de Servicio de Razonamiento (*SR*) y Grupo de Servicios de Razonamiento (*GR*) introducidos en los capítulos anteriores permiten distintas alternativas a la hora de concretar la implementación de los mismos. En este capítulo se describirá una de las alternativas existentes, la cuál está relacionada con los conceptos de agentes [Bra97, Woo99, FG96], sistemas multiagente [JSW98] y protocolos de interacción entre agentes [ON98].

Se utilizará la noción de agente para implementar el concepto de *SR*. El empleo de agentes permitirá obtener una implementación lo suficientemente flexible como para adaptarse a los distintos dominios de aplicación donde pueden ser utilizados los *SRs*. Luego, se utilizará la noción de sistema multiagente para implementar los *GRs*. Los sistemas multiagente permiten obtener, por un lado, una implementación distribuida de los *GRs*, y a su vez, lograr el comportamiento especificado a través de las funciones de selección y agrupamiento introducidas en esta tesis.

Los protocolos de interacción permiten establecer cómo serán las interacciones tanto entre clientes y *SRs*, así como también entre los distintos *SRs* que conforman los *GRs*. Se introducirán algunos protocolos de interacción, ya sea estándar o definidos ad-hoc, que pueden ser utilizados por los clientes a la hora de consultar a los *SRs*. Además, se utilizará el protocolo Contract Net para lograr que los *GRs* puedan distribuir las consultas contextuales entre los *SRs* de acuerdo a funciones de selección y agrupamiento específicas.

6.1. Motivación

Los *SRs* tienen la capacidad de representar conocimiento y permiten realizar modificaciones temporales a dicho conocimiento para responder consultas de clientes. Un *SR* representa una clase especial de servicio que puede responder *CCs* de clientes, estableciendo una relación típica del estilo cliente-servidor. Los clientes que deseen consultar este tipo de servicios utilizarán las consultas contextuales definidas en esta tesis, las cuales brindan un mecanismo de interacción entre un cliente y un *SR*.

En esta tesis se introdujeron cinco tipos de consultas contextuales, con el objetivo de lograr eficiencia en el cálculo de las respuestas, permitir declaratividad y evitar la sobrecarga en el intercambio de mensajes y las incorporaciones de los contextos, así como también facilitar la distribución del cómputo de las respuestas. Los *GRs* permiten mejorar la eficiencia en el cálculo de la respuesta a consultas múltiples y abstraer al cliente de la decisión acerca de a que *SR* enviar su consulta contextual.

Al momento de concretar la implementación de *SRs* y *GRs*, es necesario proveer a los *SRs* la capacidad de interactuar con clientes y otros *SRs*, así como también la capacidad de determinar que acción realizar ante la llegada de *CCs*. Con respecto a los *GRs*, es necesario especificar la forma de interactuar con los clientes y cómo se distribuye la resolución de las consultas entre los *SRs* que los integran.

Los conceptos de agentes y sistemas multiagente proveen las herramientas necesarias para implementar los *SRs* y los *GRs*. Los agentes poseen habilidad social, lo cuál les permite interactuar con otros agentes (clientes) y pueden ser reactivos, respondiendo a cambios en el entorno para cumplir con sus objetivos, así como también proactivos, tomando la iniciativa y comportándose de acuerdo a sus objetivos de diseño. Los sistemas multiagente están compuestos por agentes autónomos, los cuales pueden ser cooperativos y coordinar sus acciones con el objetivo de llevar a cabo sus tareas de manera eficiente y robusta.

Por lo tanto, la utilización de agentes y sistemas multiagente proveen una forma de lograr la implementación de *SRs* y *GRs* de manera general. Las principales ventajas de utilizar estos conceptos corresponden a la posibilidad de obtener una implementación estándar, facilitando la interacción con otros sistemas, y a la vez flexible, permitiendo su adaptación a distintos dominios de aplicación.

Durante el desarrollo de esta tesis se implementaron diversos prototipos relacionados con la comunicación e interacción entre agentes [GTS05, TG05, TG06, TG07]. Dichos prototipos posibilitaron la concreción de una prueba de concepto en lo que respecta a la implementación de *SRs* y *GRs*. A lo largo de este capítulo se describirá la arquitectura de un agente *SR* y la forma de coordinar los mismos para obtener un *GR*, mostrando en cada caso versiones simplificadas de dicha implementación.

6.2. Implementación de *SRs* utilizando Agentes

El Capítulo 2 describe los componentes que conforman un *SR*, así como también el comportamiento del mismo ante la llegada de *CCs*. Al momento de implementar un *SR* existen diversas alternativas. En esta sección se analizará e introducirá una de estas alternativas, la cuál está relacionada con la noción de agente inteligente [Woo99]. Esto permite disponer de una arquitectura clara donde son separados los componentes básicos de un *SR*, de la comunicación y el comportamiento.

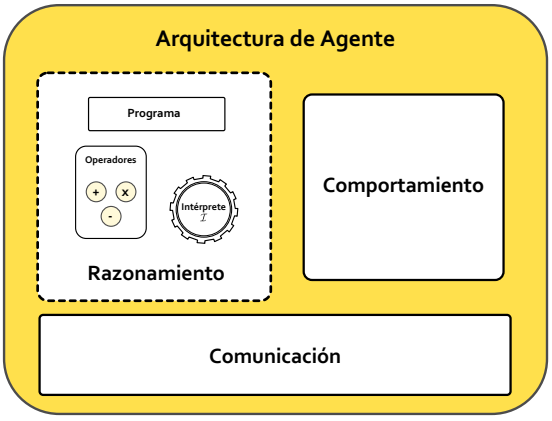

Figura 6.1: Arquitectura de Agente implementando un *SR*

A grandes rasgos, los objetivos de diseño de un *SR* son responder las consultas contextuales de los clientes eficientemente. Por lo tanto, un agente implementando un *SR* debería poseer habilidad social para poder interactuar con los clientes (otros agentes), recibiendo consultas contextuales y enviando las respuestas correspondientes, debería ser reactivo para computar las respuestas a las consultas contextuales recibidas, y finalmente, debería

ser proactivo para poder resolver dichas consultas eficientemente. La forma en la cuál la proactividad de un *SR* mejorará la eficiencia en el cómputo de una respuesta será explicada en la siguiente sección, en el momento en el cual se mencionen las alternativas a la hora de implementar los *GRs*.

6.2.1. Arquitectura de Agente para Implementar un *SR*

En la Figura 6.1 se introduce la arquitectura de agente propuesta para implementar un *SR*. Esta arquitectura está basada en el esquema de *SR* introducido en la Figura 2.2 del Capítulo 2 extendida para incluir los módulos de comunicación y comportamiento. El módulo de comunicación será el encargado de interactuar con los clientes, recibiendo consultas contextuales y enviando las respuestas correspondientes. En el caso del módulo de comportamiento, éste será el responsable, en un principio, de procesar las consultas recibidas y decidir que hacer con ellas. En particular, el módulo de comportamiento puede enviar las consultas al módulo de razonamiento para que las resuelva y luego utilizar el módulo de comunicación para enviar la respuesta correspondiente. En lo que al módulo de razonamiento respecta, éste estará compuesto por un programa, representando el conocimiento público del *SR*, los operadores de programa que permitirán modificar temporalmente dicho conocimiento a partir del contexto y el intérprete de programa que será el encargado de obtener las respuestas a las consultas.

La Figura 6.2 muestra cómo son procesadas las consultas por un agente que implementa un *SR* utilizando la arquitectura propuesta. Las consultas recibidas de los distintos clientes son almacenadas por el módulo de comunicación en una cola para ser procesadas por orden de llegada. El módulo de comportamiento es el encargado de procesar las consultas contextuales y determinar la forma en que son resueltas. Si decide resolver la consulta él mismo, entonces ésta es enviada al módulo de razonamiento. El módulo de razonamiento utilizará el contexto de la consulta y los operadores correspondientes para generar un programa temporal a partir de modificaciones sobre el programa almacenado. Luego, el intérprete de programa utiliza el programa temporal y las consultas enviadas para computar la respuesta correspondiente. Finalmente, el módulo de comportamiento enviará esta respuesta al cliente que envió la consulta, a través del módulo de comunicación.

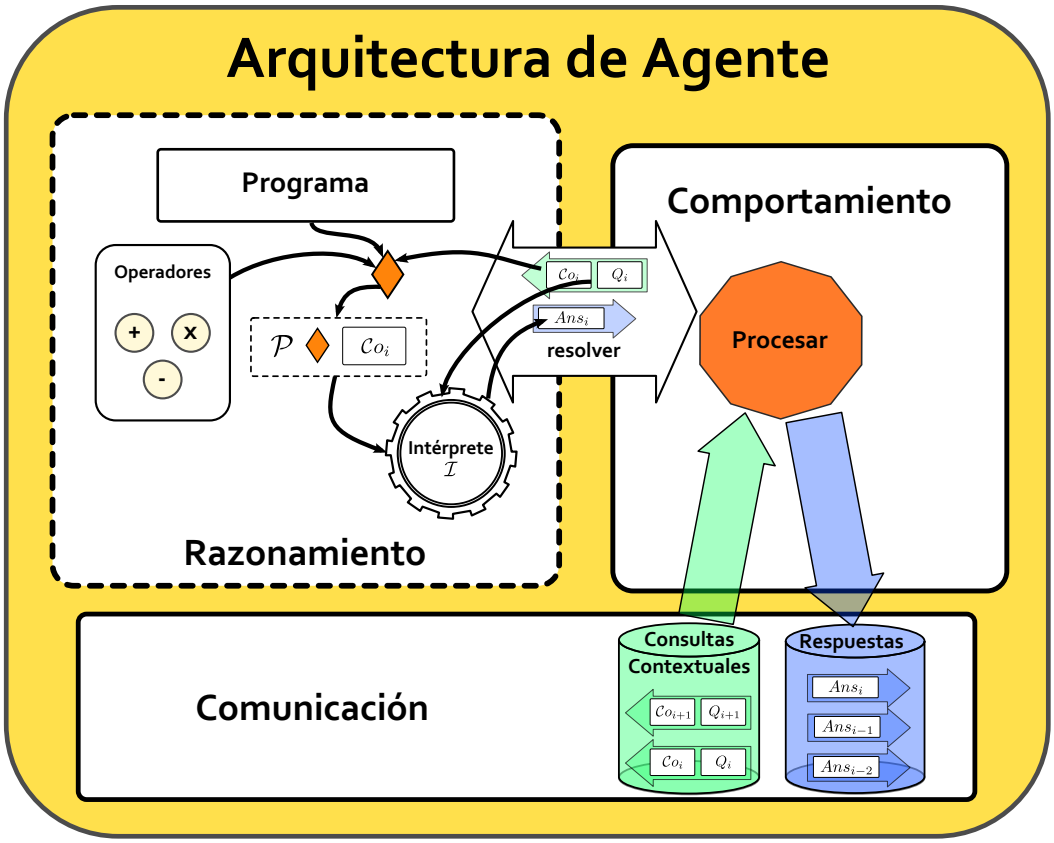

Figura 6.2: Funcionamiento de la Arquitectura de Agente

Módulo de Comportamiento

Esta arquitectura de agente representa una alternativa a la hora de implementar un *SR* que permite, al menos, dos enfoques distintos que pueden ser empleados dependiendo del dominio en que será utilizado este concepto de *SR*. Por un lado, permite la implementación de *SRs* generales, utilizando intérpretes estándares, como por ejemplo de Prolog o DeLP, para posibilitar el empleo de dichos mecanismos de inferencia como un servicio externo a otros agentes. Por otro lado, es posible extender dicha arquitectura de agente, en particular el módulo de comportamiento, para que modifiquen el estado interno del agente y permitan que las respuestas a las consultas contextuales se computen utilizando conocimiento actualizado y adecuado a las necesidades específicas del dominio.

En el caso de la primera alternativa, solamente se deben especificar los operadores y el

conocimiento que debe poseer el *SR* para poder responder las consultas contextuales de los clientes. Estos clientes pueden utilizar el contexto para obtener respuestas personalizadas y adecuadas a sus necesidades. De esta forma, un agente con la habilidad social como para interactuar con un *SR* puede utilizar un mecanismo de inferencia complejo, como por ejemplo DeLP, para razonar y tomar decisiones, sin la necesidad de implementar dicho mecanismo internamente.

Por otro lado, la segunda alternativa requiere la extensión de la arquitectura de agente presentada, ya sea modificando el módulo de comportamiento o incluyendo nuevos módulos especializados que se encarguen de plasmar los eventos relevantes del entorno en el conocimiento específico del agente. De esta forma, es posible utilizar el concepto de *SR* como un mecanismo de interacción complejo y a la vez transparente para la arquitectura del agente.

Por ejemplo, es posible modificar el módulo de comportamiento para que obtenga información de departamentos en alquiler y actualice el programa para incorporar datos relevantes y permita mantener actualizado a un *SR*. Dicho *SR* puede utilizar este conocimiento para generar recomendaciones a clientes que busquen alquilar departamentos con ciertas características y restricciones, como el ejemplo de aplicación mostrado en el Capítulo 2.

Módulo de Comunicación

Un cliente que desee realizar una consulta contextual a un *SR* o a un *GR*, deberá iniciar una interacción. La interacción entre agentes es llevada a cabo a través del intercambio de mensajes de acuerdo a una política de conversación. Estas políticas de conversación o protocolos de interacción, establecen patrones típicos de intercambio de mensajes entre agentes.

Diversos protocolos de interacción han sido ideados para los sistemas formados por agentes. En particular, FIPA [FIP02a, ON98] corresponde a un organización dedicada a la producción de especificaciones con el objetivo de que se conviertan en estándares para agentes heterogéneos que interactúan entre sí. FIPA ha jugado un papel crucial en el desarrollo de tecnologías para fomentar la implementación de agentes y sistemas multiagente.

Un protocolo de interacción estándar, definido por FIPA, que logra el comportamiento deseado y, además, es simple de implementar, corresponde al protocolo FIPA

Query [FIP02b]. Dicho protocolo, mostrado en la Figura 6.3, consiste en un cliente enviando una consulta (query) y esperando por la respuesta del receptor. Por otro lado, el *SR* (receptor), al recibir una consulta, la procesa y resuelve, ya sea de manera aislada o con la ayuda de otros *SRs*, y luego envía al cliente (quien inició el protocolo FIPA Query) la respuesta obtenida.

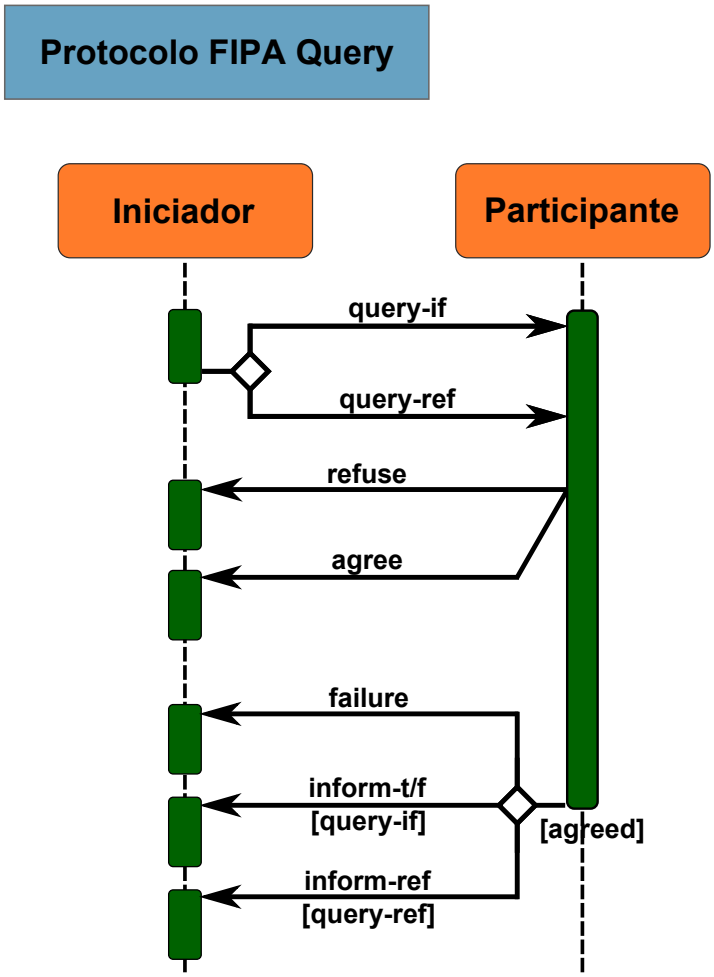

Figura 6.3: Protocolo FIPA Query

Esta alternativa de implementación posee las ventajas de ser simple y de abstraer al cliente de la forma en la cual el *SR* (o *GR*) resuelve su consulta. Por otro lado, el cliente debe esperar a la respuesta final del *SR* para obtener cualquier tipo de información, más allá de una posible notificación de que su consulta fue aceptada y está siendo procesada.

Es decir, en el caso de que la consulta enviada sea compleja, requiera mucho procesamiento para ser resuelta y genere múltiples soluciones, el cliente deberá esperar a que todas estas soluciones sean computadas y no podrá alterar en ningún punto el proceso.

Es posible definir un protocolo de interacción ad-hoc con el objetivo de permitir que el cliente no tenga que esperar a que se procese enteramente su consulta para obtener información. Dicho protocolo debería permitir al *SR* encargado de resolver la consulta, enviar soluciones parciales a medida que son generadas. Dependiendo del dominio de aplicación, más específicamente de las necesidades del cliente y del tiempo en que se tarden en computar las respuestas, podrá variar la cantidad de mensajes enviados por el *SR*.

Una alternativa aún más compleja pero que a la vez resulta más flexible corresponde a la posibilidad de que el cliente subdivida una consulta contextual extendida para ir enviando partes de la misma a medida que va obteniendo respuestas del *SR*. De esta forma, el cliente no sólo obtendrá respuestas a sus consultas a medida que se van generando sino que además podrá modificar las siguientes consultas a resolver por el *SR* a partir de las respuestas obtenidas. La principal diferencia de esta alternativa comparada con la posibilidad de enviar distintas *CCs* de manera secuencial corresponde a permitir que las consultas se realicen manteniendo un contexto, el cual a su vez puede ir variando, todo esto sin la necesidad de repetirlo en cada consulta.

Este protocolo de interacción resulta adecuado para su utilización tanto con las *CCMFs* como con las *ICs*. En el caso de las *CCMFs*, el cliente envía un único contexto y, luego, pueden ir enviando distintas consultas a medida que va obteniendo respuestas. Por otro lado, en el caso de las *ICs*, el cliente envía consultas contextuales continuadas, las cuales serán resueltas a partir del programa utilizado para computar la respuesta a la anterior consulta de la misma *IC*.

Los protocolos de interacción mencionados previamente corresponden a algunas de las alternativas existentes para implementar la interacción entre clientes y *SRs*. Estas alternativas van desde las más simples de implementar, que permiten obtener el comportamiento deseado, hasta las más complejas, que además permiten obtener ciertas ventajas, como la posibilidad de ir generando dinámicamente parte de la consulta contextual dependiendo de las respuestas obtenidas previamente para la misma consulta.

6.2.2. Implementación utilizando primitivas de mensajes

La concreción de la arquitectura de agente introducida anteriormente, y en especial el módulo de comunicación, requiere de una herramienta que permita implementar interacción entre agentes. Durante el desarrollo de las investigaciones que originaron esta tesis, se diseñó e implementó un conjunto de primitivas de interacción [GTS05], con el objetivo de facilitar la implementación de sistemas multiagente para entornos dinámicos y distribuidos.

El conjunto de primitivas mencionado fue diseñado como una extensión al lenguaje de programación lógica Prolog, debido a que dicho lenguaje es uno de los más utilizados a la hora de implementar agentes inteligentes. Un objetivo de diseño de las primitivas propuestas corresponde a tratar de conservar el espíritu de Prolog, es decir, proveer una especificación de la solución y ocultar lo más posible los detalles de implementación. En esta tesis utilizaremos una versión extendida de las primitivas de interacción originalmente publicadas en [TG05] y en [TG06].

Del conjunto de primitivas de interacción propuestas, solamente el subconjunto asociado con las primitivas de mensajes y eventos serán utilizadas en esta sección. Es decir, aquellas primitivas que permiten enviar mensajes, esperar a recibir mensajes y asociar a la llegada de un determinado mensaje la ejecución de un predicado específico. Estas primitivas permiten a los agentes intercambiar mensajes utilizando un formato compatible al utilizado por FIPA y, a su vez, le permiten abstraerse de implementaciones de bajo nivel a través de la asociación de predicados específicos a la llegada de cierto tipo de mensajes.

Las primitivas de mensajes y eventos propuestas son:

- $send\_msg(+Mensaje, -Resultado)$
- $recv\_msg(?Mensaje, -Resultado)$
- $recv\_msg(?Mensaje, +Espera, -Resultado)$
- $bind\_msg(+Mensaje, +Predicado)$

Las primitivas $recv\_msg/2$ y $recv\_msg/3$ son bloqueantes, es decir, una vez ejecutadas se quedan a la espera de un mensaje que cumpla con las características requeridas. En el caso de $recv\_msg/3$, el parámetro $Espera$ indica el tiempo máximo en segundos que

espera dicha primitiva por el arribo de un mensaje. Una vez finalizado dicho tiempo, si no se recibió ningún mensaje, la primitiva devuelve en *Resultado* el mensaje correspondiente.

El formato de los mensajes intercambiados entre los agentes corresponde a una lista de parámetros y valores, donde para enviar un mensaje se debe incluir obligatoriamente el parámetro *receiver* (receptor en ingles), indicando el agente a cuál está destinado dicho mensaje. Mientras que, al momento de esperar por la llegada de un mensaje o asociar determinados mensajes a predicados específicos, es posible especificar los parámetros e incluso los valores que debe tener el/los mensaje/s correspondiente/s.

El conjunto de primitivas diseñado e implementado en Prolog permitirá procesar todos los mensajes recibidos de otros agentes así como también enviar mensajes a estos agentes u otros. De esta forma, para lograr el comportamiento deseado por el módulo de comunicación, simplemente se debe asociar la llegada de mensajes de un tipo especifico que se encuentre relacionado con las consultas contextuales (habiendo sido previamente definido y acordado con el cliente) a la llamada de un predicado responsable de procesarlas. Por ejemplo, si se ejecuta la sentencia:

$$\begin{aligned} bind(\quad & [sender(Cliente), protocol(fipa\_query), ontology(contextual\_query), \\ & \; performative(query\_ref), content(\mathcal{CC}), conversation\_id(Id) \;], \\ & procesar\_cc(Id, \mathcal{CC}, Cliente) \;). \end{aligned}$$

se estará asociando la llegada de mensajes de un determinado *Cliente* perteneciente al protocolo *fipa_query* con la ontología *contextual_query* y cuyo acto comunicativo es *query_ref* a la ejecución del predicado *procesar_cc* responsable de procesar la consulta contextual $\mathcal{CC}$.

El predicado *procesar_cc*, incluido dentro del módulo de comportamiento, será el encargado de procesar la consulta contextual y, en caso de determinar que puede resolverla, enviar la aceptación correspondiente al cliente e iniciar la resolución de la misma. Finalmente, una vez obtenida la respuesta a la consulta, ésta es enviada al cliente a través del módulo de comunicación. A continuación se muestra una versión simplificada de una posible implementación de dicho predicado:

$$
\begin{aligned}
procesar\_cc(\quad & Id, \quad \mathcal{CC}, \quad Cliente \ ) \ :- \\
& puedo\_resolver(\mathcal{CC}), \\
& enviar\_aceptación(Id, Cliente), \\
& resolver(\mathcal{CC}, Respuesta), \\
& enviar\_respuesta(Id, Cliente, Respuesta).
\end{aligned}
$$

El predicado *puedo_resolver* utilizará el módulo de razonamiento para determinar si el *SR* es capaz de resolver la consulta. En el caso del predicado *resolver*, éste decidirá como resolver la consulta. Una alternativa es utilizando el módulo de razonamiento, pero como se mostrará en la siguiente sección, existe otra posibilidad. Finalmente, los predicados *enviar_aceptación* y *enviar_respuesta* utilizan el módulo de comunicación para enviar los mensajes correspondientes al cliente. Una versión simplificada del predicado *enviar_respuesta* es la siguiente:

$$
\begin{aligned}
enviar\_respuesta(\quad & Id, \quad Cliente, \quad Respuesta \ ) :- \\
& send\_msg \quad (\ [\ receiver(Cliente), performative(inform\_result), \\
& \quad protocol(fipa\_query), ontology(contextual\_query), \\
& \quad content(Respuesta), conversation\_id(Id) \ ] \ ).
\end{aligned}
$$

La arquitectura de agente introducida en esta sección con el objetivo de implementar *SRs* es general y a la vez concreta. Es general porque permite su utilización en distintos dominios de aplicación, y es concreta porque establece claramente los componentes y las características que posee un agente o *SR*. En esta subsección se mostraron fragmentos de código que representan una versión simplificada de la implementación de un *SR*.

En la siguiente sección se analizará el comportamiento de estos agentes (*SRs*) dentro de un *GR* y se introducirá un mecanismo que permite obtener el funcionamiento deseado de los *GRs* asociado a las funciones de selección y agrupamiento. Se extenderá, además, esta versión simplificada de la implementación, con el objetivo de permitir la interacción entre los *SRs* y la descomposición y distribución de las consultas contextuales extendidas.

6.3. Sistemas Multi-Agente para Implementar *GRs*

Teniendo en cuenta la alternativa mencionada en la sección anterior que utiliza el concepto de agente para implementar *SRs*, en esta sección emplearemos la noción de Sistema

Multiagente (*SMA*) para implementar el concepto de *GR* introducido en el Capítulo 4. En este contexto, los agentes (*SRs*) que conforman un *SMA* (*GR*) poseen objetivos y tareas (consultas contextuales) comunes. El término Sistema Multiagente ha adquirido un significado general, siendo utilizado para hacer referencia a todos los tipos de sistemas compuestos de múltiples componentes autónomos.

6.3.1. Coordinación entre Agentes

La coordinación es una propiedad de sistemas en donde un grupo de agentes desarrollan alguna actividad en un entorno común. Los agentes de un *SMA* deben coordinar sus actividades para cumplir con sus propios intereses y satisfacer los objetivos del grupo [HS99]. Las acciones de los agentes necesitan ser coordinadas para permitir entregar a otros agentes (clientes) respuestas a sus consultas en tiempo y forma, asegurando sincronización entre los agentes del *SMA* que calculan dichas respuestas y evitando, a su vez, la resolución de problemas de manera redundante.

Con el objetivo de lograr coordinar agentes en un *SMA*, una posibilidad es utilizar la técnica de distribución de control y datos entre los propios agentes. Utilizando esta técnica, los agentes poseerán un grado de autonomía en la decisión de qué objetivos o acciones perseguir y en la generación de nuevas acciones. En el caso particular de un agente implementando un *SR*, éste recibirá consultas contextuales y tendrá la posibilidad de decidir si descompone la misma en consultas más simples o no, y en caso afirmativo, distribuirá el cómputo de estas nuevas consultas entre los agentes del *SMA*.

Al momento de decidir cómo descomponer una consulta contextual o, incluso, si hacerlo o no, el *SR* debe considerar diferentes aspectos relacionados con, por un lado, el tipo y tamaño de la consulta contextual a procesar y, por otro lado, la cantidad y los distintos tipos de *SRs* disponibles. Una vez dividida la consulta contextual en nuevas consultas contextuales más simples, éstas pueden ser distribuidas en los distintos *SRs* que conforman el *GR*, utilizando distintos criterios, como por ejemplo, evitando la sobrecarga de recursos, asignado consultas a *SRs* con las capacidades o información vinculada a la consulta, entre otras alternativas.

Uno de los mecanismos más conocidos y utilizados por los *SMAs* para distribuir las tareas corresponde al protocolo Contract Net [Smi81]. El protocolo Contract Net, mostrado en la Figura 6.4, es un protocolo de interacción para la solución de problemas de manera

cooperativa entre agentes, el cual está modelado en base al mecanismo de contratación utilizado en las empresas para el intercambio de bienes y servicios. Este protocolo soluciona el problema de encontrar el *SR* adecuado para resolver una determinada consulta contextual permitiendo el empleo de los criterios antes mencionados.

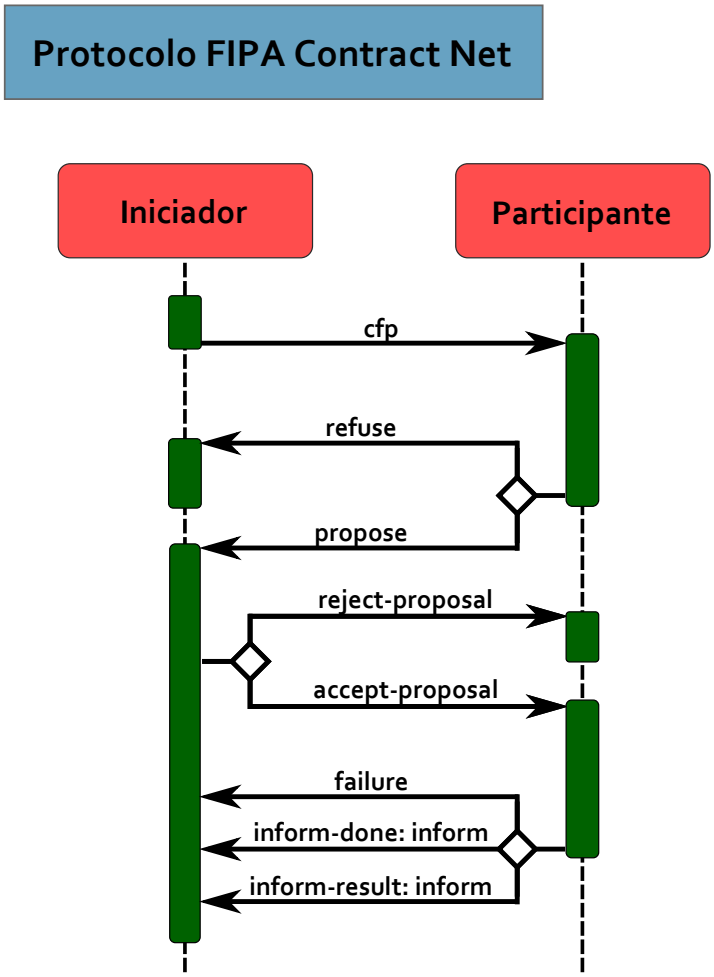

Figura 6.4: Protocolo FIPA Contract Net

Un *SR* que necesita resolver una consulta contextual es denominado *administrador*, mientras que los *SRs* que pueden resolver las consultas son denominados potenciales *contratistas*. Desde el punto de vista del administrador, el proceso es:

- Anunciar la consulta contextual que se necesita resolver,

- Recibir y evaluar las ofertas de potenciales contratistas,
- Seleccionar el contratista adecuado para resolver la consulta contextual y
- Recibir y procesar las respuestas.

Por otro lado, desde la perspectiva del contratista, el proceso es el siguiente:

- Recibir y procesar los anuncios de consultas contextuales,
- Evaluar la capacidad para resolver la misma,
- Responder ya sea declinando u ofertando,
- En el caso de que sea aceptada la oferta, resolver la consulta contextual y
- Enviar la respuesta al administrador.

Los roles de los agentes no están preestablecidos, por lo tanto, un *SR* puede actuar tanto como administrador realizando anuncios, así como también como contratista, respondiendo a los anuncios de consultas contextuales de otros *SRs*. Esta flexibilidad permite que cualquier *SR* dentro de un *GR* pueda recibir consultas contextuales y que, a partir del empleo del protocolo Contract Net (ver Figura 6.5), estas consultas sean resueltas por los *SRs* más idóneos dentro del *GR* sin la necesidad de que cada *SR* tenga información respecto de los demás *SRs* dentro del *GR*.

Para lograr esto, un *SR* puede separar cualquier tipo de consulta contextual extendida recibida en *CCs* e iniciar el protocolo Contract Net. Una vez recibidas las ofertas para las distintas *CCs*, el administrador puede seleccionar para cada una de ellas el *SR* adecuado e incluso, podría agrupar aquellas consultas que vayan a ser resueltas por el mismo contratista o *SR*. De esta forma, se obtiene una implementación general del concepto de *GR* que permite tanto la distribución de las consultas compuestas en consultas más simples así como también el agrupamiento de las consultas simples de acuerdo a los *SRs* que estarán encargados de resolverlas.

Por otro lado, también permite una mayor descomposición y/o distribución de las tareas por parte de los contratistas, los cuales pueden a su vez iniciar un nuevo protocolo Contract Net entre los mismos o distintos agentes. En particular, este mecanismo puede utilizarse transparentemente en la implementación de grupos de *GRs*, donde una consulta

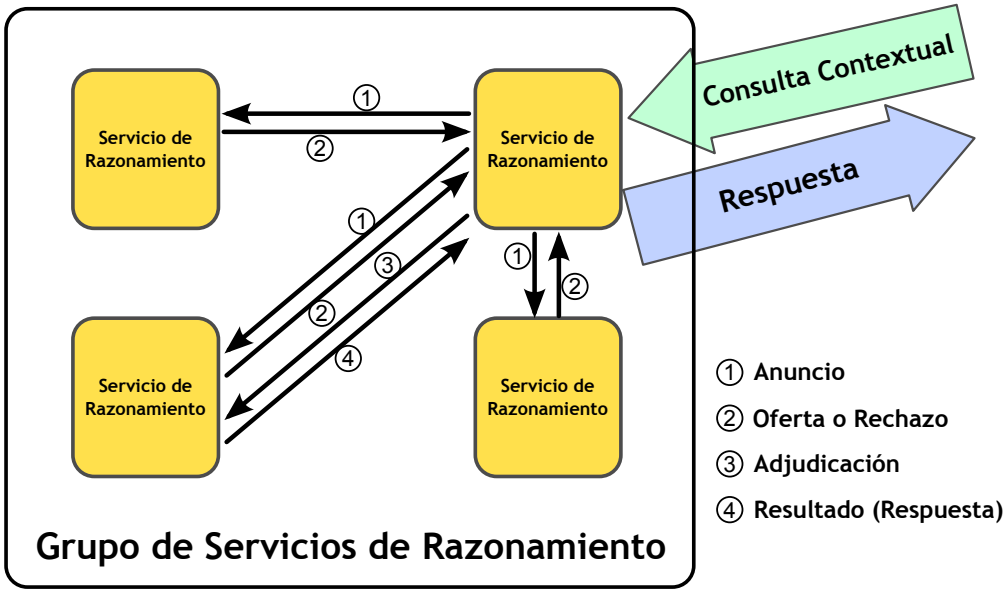

Figura 6.5: Protocolo Contract Net para distribuir una Consulta Contextual

que es recibida por un grupo de *GRs* puede descomponerse en consultas más simples y distribuirse entre los distintos *GRs* que forman el grupo general, y a su vez, cada consulta más simple puede nuevamente descomponerse y distribuirse entre los *SRs* de los distintos *GRs*.

Esta alternativa de implementación de los *GRs* tiene la ventaja de ser distribuida y, por lo tanto, no tiene un único punto de falla o cuello de botella como si ocurriría con una opción centralizada. Esta característica permite no sólo que las consultas contextuales puedan arribar al *GR* por cualquier *SR* sino que, además, posibilita una composición dinámica del *GR* en cuanto a la cantidad y capacidad de los *SRs* que lo conforman. Por otro lado, debido a que esta alternativa es distribuida y no requiere que los agentes posean información de los demás integrantes del *GR*, resulta escalable e, incluso, permite ser extendido de manera transparente para la implementación de grupos de *GRs*.

Una de las desventajas de la utilización de este protocolo corresponde a la posibilidad de seleccionar para la resolución de una determinada consulta contextual a un contratista con capacidades limitadas debido a que uno mejor calificado se encuentra ocupado al momento de recibir el anuncio. Por otro lado, un administrador no está obligado a informar

a los potenciales contratistas que ya fue seleccionado un *SR* para resolver la tarea, lo cual puede complicar y retrasar la tarea de selección de los contratistas con respecto a los diferentes anuncios existentes. Además, un administrador puede no recibir ninguna oferta por diversas razones, entre las cuales se encuentran, por ejemplo, el hecho de que todos los potenciales contratistas estén ocupados o, bien, que estén disponibles pero seleccionen otro anuncio de acuerdo a sus preferencias y capacidades. En cualquiera de los casos antes mencionados existen ciertas modificaciones posibles al protocolo o, incluso, al comportamiento de los agentes que permiten minimizar y hasta evitar las distintas desventajas.

Este mecanismo permite implementar la distribución de las tareas en un *GR*, siendo lo suficientemente flexible como para adaptarse a distintos dominios de aplicación y lograr el comportamiento deseado. En lo que va de este capítulo se introdujo la alternativa de utilizar agentes y sistemas multiagentes a la hora de implementar los conceptos de *SR* y *GR* introducidos en esta tesis. En la siguiente sección se analizarán algunas alternativas que surgen a la hora de implementar la interacción entre los clientes y los *SRs*.

6.3.2. Implementación de *SMAs*

Una habilidad que necesita poseer un agente que es parte de un *SMA* corresponde a la capacidad de localizar otros agentes dentro del propio *SMA* y comunicarse con ellos. A partir del momento en que los agentes pueden localizar y comunicarse con sus pares dentro de un mismo *SMA* es que resulta posible la implementación de protocolos de interacción como el protocolo Contract Net antes mencionado. Como parte de los trabajos realizados en el desarrollo de esta tesis, se diseñó e implementó un modelo de comunicación basado en la arquitectura y los protocolos de interacción de FIPA [TG07], que permite a un agente encontrar otros agentes de acuerdo a sus características e intercambiar mensajes con ellos.

En el modelo de comunicación propuesto, los agentes se registran en los *SMAs* en los cuales desean participar con el objetivo de permitir a otros agentes que los localicen fácilmente. Los agentes pueden agruparse en los distintos *SMAs* de acuerdo a sus características o servicios provistos. En el caso particular de esta tesis, un *SMA* representará un *GR*. El modelo incluye un agente especial, denominado *YPA* (Yellow Pages Agent en in-

glés o Agente Páginas Amarillas), el cuál es responsable de mantener toda la información de los agentes registrados en los distintos *SMAs*.

Con el objetivo de facilitar el desarrollo de agentes utilizando el modelo antes mencionado, se propuso un conjunto reducido de primitivas que simplifican la interacción con el *YPA*. Estas primitivas posibilitan a los agentes conectarse a un *YPA*, registrarse en uno o varios *SMAs*, y obtener la lista de *SMAs* existentes o de agentes registrados en un *SMA* específico. De esta forma, un agente implementando un *SR* puede registrarse en el *GR* correspondiente y localizar a los demás agentes dentro del mismo *GR* (*SMA*) .

Una vez resuelto el problema de localización de agentes dentro de un mismo *GR* y el problema de interacción entre agentes, a partir de la utilización de las primitivas de mensajes introducidas en la sección anterior, un agente se encuentra en condiciones de implementar el protocolo Contract Net. En este caso en particular, un *SR* debe implementar los dos roles del protocolo de interacción, siendo que podrá iniciar el protocolo como administrador para distribuir el cómputo de una consulta, y también podrá participar del mismo como contratista, determinando si quiere ofrecerse para resolver la consulta o no.

En lo que respecta al rol de administrador, un *SR*, al momento de procesar una consulta contextual recibida, puede decidir iniciar el protocolo Contract Net. De acuerdo a la implementación mostrada en la sección anterior, esta decisión la debe tomar en el predicado $resolver(CC, Respuesta)$ (ver Subsección 6.2.2). A continuación se muestra una versión simplificada del predicado que podría utilizar un *SR* para iniciar el protocolo Contract Net:

$$
\begin{aligned}
iniciar\_cn(\ \mathcal{CC}, \quad & Respuesta\ ) :- \\
& obtener\_SRs(SRs), \\
& enviar\_anuncio(SRs, \mathcal{CC}, Id), \\
& obtener\_ofertas(Id, Ofertas), \\
& procesar\_ofertas(Ofertas, Contratista), \\
& enviar\_adjudicación(Id, Contratista), \\
& esperar\_respuesta(Id, Contratista, Respuesta).
\end{aligned}
$$

Inicialmente, el predicado *iniciar_cn* obtiene los *SRs* que conforman el *GR*, probablemente consultando al agente *YPA*. Luego, envía el anuncio a todos los *SRs* y espera a recibir las ofertas de los mismos, todo esto utilizando el módulo de comunicación. Una vez obtenidas todas las ofertas, las procesa y determina cuál de los *SRs* resulta seleccionado para responder la consulta contextual. Finalmente, envía un mensaje al contratista

informándole que ha sido seleccionado para resolver la consulta y espera a recibir la respuesta correspondiente.

Los predicados encargados de intercambiar mensajes con los posibles contratistas utilizan el módulo de comunicación, en especial, las primitivas de mensajes introducidas en la sección anterior. Los mensajes intercambiados incluirán el nombre del protocolo, la ontología correspondiente y un identificador de la conversación, además de la información relativa al emisor y los receptores del mensaje y el contenido propiamente dicho. Finalmente, el predicado $procesar\_ofertas/2$ es el encargado de decidir a que SR adjudicar la consulta contextual y su implementación dependerá del criterio que se utilice para la distribución de las mismas.

Por otro lado, para implementar el rol del contratista, un SR debe asociar la llegada de anuncios de otros SRs a la ejecución de un predicado que se encargue de procesar dichos anuncios y decidir que oferta realizar. Para lograr esto, el SR puede ejecutar el siguiente predicado:

$$\begin{aligned} bind(\quad & [performative(cfp), protocol(contract\_net), ontology(contextual\_query), \\ & conversation\_id(Id), content(\mathcal{CC}), sender(Administrador)\ ], \\ & procesar\_cn(Id, \mathcal{CC}, Administrador)\ ). \end{aligned}$$

De esta forma, cuando el SR reciba un anuncio de otro SR, se ejecutará automáticamente el predicado $procesar\_cn/3$. Dicho predicado será el encargado de determinar si puede resolver la consulta contextual y que oferta está dispuesto a hacer para resultar adjudicado. A continuación se muestra una posible implementación de dicho predicado:

$$\begin{aligned} procesar\_cn(\ Id, \quad & \mathcal{CC}, \quad Administrador\ ) :- \\ & puedo\_resolver(\mathcal{CC}), \\ & quiero\_resolver(\mathcal{CC}, Oferta), !, \\ & enviar\_oferta(Id, Administrador, Oferta), \\ & esperar\_adjudicación(Id, Administrador), \\ & resolver(\mathcal{CC}, Respuesta), \\ & enviar\_respuesta(Id, Administrador, Respuesta). \end{aligned}$$

$$\begin{aligned} procesar\_cn(\ Id, \quad & \mathcal{CC}, \quad Administrador\ ) :- \\ & enviar\_rechazo(Id, Administrador). \end{aligned}$$

Inicialmente, el predicado determina si puede y quiere resolver la consulta contextual. En un caso negativo, es decir, que no quiere o no puede resolver la consulta, envía el mensaje de rechazo correspondiente. En cambio, si desea resolver la consulta, envía la oferta al *Administrador* y espera por la posible adjudicación. Este predicado, esperará un tiempo fijo por la llegada del mensaje correspondiente, y en caso de que obtenga la adjudicación esperada, resuelve la consulta (con el predicado $resolver/2$) y envía la respuesta al *Administrador*.

El predicado $quiero\_resolver/2$ es el encargado de determinar si un *SR* desea resolver una consulta contextual determinada. Este predicado puede tener en cuenta, a la hora de decidir, no sólo la consulta contextual recibida, sino también el estado del *SR*, en particular, la carga del mismo. Por otro lado, es importante aclarar que, cuando el *SR* resulta seleccionado para resolver la consulta, se llama al predicado $resolver/2$, el cuál, a su vez, puede descomponer la consulta nuevamente e iniciar un nuevo protocolo Contract Net para distribuir la resolución de la misma.

6.4. Conclusiones

En este capítulo se describió una alternativa a la hora de implementar los conceptos de *SR* y de *GR*, así como también las interacciones entre clientes y *SRs*. Se utilizaron agentes para implementar *SRs*, Sistemas Multi-Agente para los *GRs* y se describieron y analizaron algunos protocolos de interacción que permiten que los clientes puedan enviar sus consultas contextuales a los *SRs*.

Al momento de implementar un *SR*, los agentes representan una alternativa general y flexible, permitiendo adaptarse a distintos dominios de aplicación. En este capítulo se describió la arquitectura de agente desarrollada para implementar *SRs*, la cuál está compuesta por un módulo de comunicación, encargado de manejar las interacciones con los demás agentes del sistema, un módulo de comportamiento, encargado de determinar el accionar del *SR* y un módulo de razonamiento, responsable de resolver las consultas contextuales.

Los protocolos de interacción establecen patrones típicos de intercambios de mensajes entre agentes. En este capítulo se analizaron tres alternativas que poseen los clientes a la hora de interactuar con los *SRs*. Una de esas alternativas corresponde al protocolo estándar

FIPA Query, que resultó ser la opción más simple. Además, se introdujeron dos protocolos ad-hoc que permiten a los clientes, por un lado, obtener respuestas lo antes posible, y además, modificar dinámicamente las consultas y los contextos enviados. Utilizando un conjunto de primitivas de interacción especialmente diseñadas para el desarrollo de agentes, se mostró una versión simplificada de un *SR* implementando el protocolo FIPA Query y procesando las consultas de los clientes.

En el caso particular de los *GRs*, los *SMAs* representan una alternativa distribuida de implementación. La coordinación entre los *SRs* que conforman el *GR* cumple un rol fundamental en el procesamiento de las consultas contextuales recibidas. El protocolo Contract Net permite coordinar las acciones de los *SRs* incluidos en el *GR*, facilitando la distribución del cómputo de las consultas contextuales de acuerdo a las funciones de selección y agrupamiento. En lo que respecta a la concreción del *SMA*, se introdujo un modelo de comunicación que permite localizar *SRs* dentro de un mismo *GR* y se mostró una posible implementación del protocolo Contract Net.

Capítulo 7

Conclusiones

En esta tesis se presentó una clase particular de *Servicios de Razonamiento*, para entornos con múltiples agentes, y de *Grupos de Servicios de Razonamiento*, para permitir el tratamiento de diferentes tipos de consultas en paralelo. El concepto introducido de *Consulta Contextual* permite a los clientes incluir en sus consultas conocimiento privado y especificar que tratamiento darle a dicho conocimiento, al momento de integrarse con el conocimiento público almacenado en el Servicio de Razonamiento.

Los Servicios de Razonamiento permiten representar información o conocimiento público y responder consultas de clientes utilizando este conocimiento público junto con información privada provista por el cliente, la cuál es denominada *contexto*. Para lograr esto, los Servicios de Razonamiento poseen un mecanismo de inferencia que les permite obtener respuestas a consultas utilizando el conocimiento público modificado temporalmente por la información de los clientes utilizando operadores de tratamiento de contexto específicos.

Un Servicio de Razonamiento puede disponer de distintos operadores para el tratamiento del contexto enviado por los clientes. En esta tesis se definieron ejemplos de posibles operadores para distintos dominios de aplicación específicos, que permiten, por ejemplo, agregar o quitar temporalmente información del conocimiento público del Servicio de Razonamiento, así como también dar prioridad ya sea al conocimiento público o al privado al momento de agregar información.

Los conceptos de Servicio de Razonamiento (SR) y Consulta Contextual (CC) permiten, por un lado, que el SR pueda contestar de manera personalizada las consultas de los clientes, incluso, simultáneamente, sin que por esto se vean afectadas las respuestas

obtenidas por éstos. Además, desde el punto de vista del cliente, le permite obtener una respuesta de acuerdo a sus necesidades, utilizando información privada en la resolución de la misma. Estos conceptos fueron definidos en forma abstracta, asumiendo un *marco de representación de conocimiento* que incluye un lenguaje de representación que permite realizar inferencias y un lenguaje de representación de consultas y respuestas acorde al lenguaje de representación de conocimiento elegido.

A lo largo de la tesis se introdujeron diferentes ejemplos y dominios de aplicación utilizando formalismos concretos para representar conocimiento, consultas y respuestas. Se introdujeron *SRs* utilizando la Programación Lógica que sugieren tratamientos médicos y que recomiendan noticias deportivas, y *SRs* utilizando Lógica Rebatible y Argumentación que proponen departamentos para alquilar o incluso que aconsejan acerca de qué acciones de la bolsa comprar.

En escenarios donde existan gran cantidad de clientes y de *SRs*, y además el nivel de interacción entre ellos sea alto, generalmente se requerirá del intercambio de grandes cantidades de información. Los distintos tipos de Consultas Contextuales definidos permiten mejorar la eficiencia en el cómputo de las respuestas de múltiples *CCs*, reduciendo la sobrecarga en el intercambio de mensajes y las actualizaciones del conocimiento en el *SR* y, a su vez, posibilitan el procesamiento paralelo de las mismas entre distintos *SRs*. Los distintos tipos de Consultas Contextuales definidos tratan de abarcar todos los posibles escenarios en los cuales un cliente necesita ejecutar múltiples *CCs*.

Las *Consultas Contextuales Múltiples* agrupan múltiples *CCs* en una única consulta disminuyendo la sobrecarga en el intercambio de mensajes y ofreciendo una oportunidad de paralelismo. Las *Consultas Contextuales Múltiple Factorizadas* incluyen un único contexto en el cuál se responderán las consultas incluidas. Este tipo de consulta permite al *SR* encargado de resolver las consultas minimizar las modificaciones temporales requeridas sobre el conocimiento público, además de ofrecer una nueva oportunidad de paralelismo.

Otro tipo de interacción propuesta es la *Interrogación Contextual*, que permite intercalar consultas y modificaciones temporales al conocimiento del *SR*, siendo estas modificaciones aplicadas gradualmente. Este tipo de consulta permite disminuir tanto el intercambio de mensajes como las modificaciones necesarias para el procesamiento de las consultas. Finalmente, la *Interrogación Contextual Múltiple* representa una consulta donde se agrupan múltiples Interrogaciones Contextuales, incluyendo las ventajas de las Interrogaciones Contextuales y, a su vez, ofreciendo oportunidades de paralelismo.

Existen ciertas propiedades que relacionan los distintos tipos de consultas definidos en esta tesis. Por ejemplo, la Consulta Contextual resulta ser un caso particular de los otros cuatro tipos de Consultas Contextuales. Las Consultas Contextuales Múltiples Factorizadas pueden ser emuladas por las Interrogaciones Contextuales y tanto éstas últimas como las Consultas Contextuales Múltiples pueden ser reemplazadas por Interrogaciones Contextuales Múltiples. De esta forma, se puede decir que las Interrogaciones Contextuales Múltiples representan el tipo de consulta más general, abarcando a todas las consultas definidas en esta tesis.

Los Grupos de Razonamiento (*GR*) tienen como principal objetivo permitir una distribución automática del cómputo de la respuesta a las consultas contextuales entre los distintos *SRs*. Estos grupos simplifican la interacción de un cliente frente a un grupo de *SRs* y facilitan la selección del *SR* más indicado para resolver cada consulta recibida, posibilitando incluso, cuando esto sea factible, el cómputo en paralelo de la respuesta a una consulta por distintos *SRs*.

Los conceptos de *Función de Selección* y *Función de Agrupamiento* representan los mecanismos que poseen los Grupos de Razonamiento para distribuir las consultas recibidas entre los *SRs* incluidos en el grupo. Para lograr un correcto desempeño, se definieron las propiedades de *Sensatez* y *Completitud* de las Funciones de Agrupamiento, que aseguran que estas funciones cumplen con ciertas restricciones y características deseables. Se mencionaron distintas alternativas a la hora de concretar dicho mecanismo y se mostró una posible implementación utilizando el protocolo *Contract Net*.

La definición del concepto de Grupo de Razonamiento permitió distinguir entre *GR* homogéneos y heterogéneos. Los *GRs* homogéneos son aquellos que están formados por *SRs* con iguales características y permiten distribuir y paralelizar el cómputo de los diferentes tipos de consultas contextuales. Los *GRs* heterogéneos están formados por *SRs* que pueden generar distintas respuestas ante las mismas consultas contextuales. Estos grupos tiene la posibilidad de utilizar el contexto de la consulta para elegir el *SR* más *indicado* para calcular la respuesta.

Como ejemplo de aplicación de los *GRs* homogéneos se implementó un Intérprete de DeLP en Paralelo. El *GR* utilizado está compuesto por *SRs* específicos (denominados *SRLA*s) con la capacidad de generar (como respuesta a consultas contextuales específicas) argumentos y contra-argumentos que posibilitan el cálculo en paralelo del estado de garantía de una consulta DeLP. En este ejemplo de aplicación se muestra un uso concreto

de los conceptos de *CC*, *SR* y *GR* introducidos en esta tesis, obteniendo una implementación de propósito general que a su vez puede ser utilizada o extendida para resolver problemas de dominios específicos.

Finalmente, se propuso una alternativa de implementación concreta, utilizando agentes para representar los *SRs*, Sistemas MultiAgente para los Grupos de Razonamiento y protocolos de interacción para determinar cómo se lleva a cabo el intercambio de consultas entre clientes y *SRs* y entre *SRs* dentro de un Grupo. Se presentó una arquitectura de agente especialmente diseñada para concretar los *SRs* y se mostró una posible implementación en Prolog utilizando primitivas de intercambios de mensajes.

Como trabajo futuro a corto plazo es posible completar algunas implementaciones específicas. Por ejemplo, una implementación de Grupo de Razonamiento para DeLP que permitirá mejorar la implementación existente conocida como DeLP Server (una implementación particular de *SR*). Además, sería posible disponer de una implementación de DeLP en paralelo. Otro trabajo futuro a corto plazo sería implementar un *GR* heterogéneo que permita interactuar con agentes en algún lenguaje estandar como KQML o FIPA ACL.

Como trabajo futuro a largo plazo sería posible estudiar en detalle como aprovechar los conceptos propuestos para desarrollar aplicaciones con otros formalismos de razonamiento que se adapten a esta forma de interactuar entre servicios y agentes, como por ejemplo Description Logics o Bases de Datos. Otro posible trabajo futuro a largo plazo corresponde al análisis de una semántica diferente para las Consultas Contextuales, en la cual las modificaciones realizadas por el contexto de las mismas fuera *permanente*.

Apéndice A

Programación Lógica

La programación lógica corresponde a un formalismo simple y, a la vez, poderoso, que posibilita la representación de conocimiento y la programación. La programación lógica, introducida en 1974 por R. Kowalski [Kow74], ofrece un nuevo paradigma de programación, el cuál fue originalmente implementado por el lenguaje de programación Prolog. Dicho lenguaje de programación fué creado en los principios de los 70 por un grupo liderado por A. Colmerauer. [CR93].

La programación lógica permite escribir programas y consultarlos para obtener respuestas. Existen dos formas naturales de interpretar un programa lógico. La primera, denominada *interpretación declarativa*, está relacionada con *qué* está siendo computado, mientras que la segunda, denominada *interpretación procedural*, explica *cómo* se realiza el cómputo. Es decir, la interpretación declarativa está asociada a la semántica, mientras que la interpretación procedural al mecanismo utilizado para llevar a cabo el cómputo.

La programación lógica, al ser declarativa, permite que los programas lógicos resulten más simples de entender y desarrollar. Básicamente, la programación lógica permite describir relaciones, posiblemente infinitas, entre objetos para luego utilizar un mecanismo de inferencia que se ajuste a la interpretación procedural del lenguaje y permita obtener conclusiones [Apt97]. Este apéndice mostrará cómo es la sintaxis de los programas lógicos válidos y cuál es el significado de las distintas sentencias que se incluyen en un programa. Además, se definirán las consultas que se pueden realizar sobre estos programas lógicos, junto con su significado.

Por último, se definirá la semántica de los programas lógicos y de las derivaciones que se pueden obtener a partir de estos, para finalmente definir cómo es una respuesta válida

para una consulta lógica determinada. Todas estas definiciones son necesarias para poder entender el significado de los programas lógicos que son utilizados en los ejemplos de *SRs* y consultas contextuales introducidos en esta tesis.

A.1. Sintaxis

Las definiciones incluidas en esta sección fueron tomadas y adaptadas de [Llo87]. Lo primero que se asumirá es un alfabeto del lenguaje, el cuál debe incluir símbolos para denotar individuos (constantes) y símbolos para denotar relaciones (símbolos de predicado). Ejemplos de constantes son *mariano*, *graciela* y *ruben*, mientras que ejemplos de símbolos de predicado son *hijo_de*, *padre* y *madre*. Los símbolos de predicado tienen una aridad fija, es decir, un número de argumentos asociados con ellos.

Con el alfabeto de constantes, los símbolos de predicado y algunos caracteres auxiliares, sentencias del lenguaje natural tales como "mariano es hijo de graciela" y "graciela es madre" pueden ser formalizadas con fórmulas tales como *hijo_de(mariano,graciela)* y *madre(graciela)*.

Además, el alfabeto debe permitir expresar sentencias referidas a todos los elementos del mundo descripto. Por ejemplo, es necesario poder formalizar la expresión "para todos los individuos M y H, si M es madre y H es un hijo de M, entonces M ama a H". Para lograr esto, se introducirán las variables, las cuales se notarán con la primera letra en mayúsculas. Una variable es un símbolo que se refiere a individuos no especificados, como M y H en la sentencia anterior. Para formalizar dicha sentencia se utilizará el siguiente expresión: $ama(M, H) \leftarrow madre(M), hijo\_de(H, M)$.

Hasta el momento, los individuos han sido representados únicamente a través de constantes. Sin embargo, puede ocurrir que en el mundo que se desea modelar, algunos "individuos" sean "objetos compuestos". Por ejemplo, en algunos casos puede ser necesario establecer relaciones entre familias así como también entre las personas que la componen. En este caso, resulta deseable poder referirse a una dada familia utilizando una construcción compuesta por las constantes que identifican a sus miembros.

Para lograr esto se utilizarán los símbolos de funciones, representando funciones entre los objetos del dominio y teniendo una aridad mayor que cero. Por ejemplo, asumiendo que se tiene un símbolo de función ternaria *familia* (con aridad 3), un símbolo de función

binaria *hijo* (con aridad 2) y la constante *nulo*. Luego, la familia compuesta por los padres Ruben y Graciela y los hijos Sebastián y Mariano puede ser representada por la construcción: *familia(ruben,graciela,(hijo(sebastian,hijo(mariano,nulo))))*.

Desde un punto de vista sintáctico, las sentencias son secuencias finitas de símbolos primitivos. Luego, lo primero que se define es el alfabeto del lenguaje, el cuál consiste de las siguientes clases de símbolos:

Definición A.1 (Alfabeto)**.** *Un alfabeto consiste de cuatro clases de símbolos:*

1) variables,
2) constantes,
3) símbolos de funciones, y
4) símbolos de predicados.

En el lenguaje formalizado, los objetos serán representados por cadenas de caracteres denominados *términos* y cuya sintaxis se define a continuación:

Definición A.2 (Término)**.** *Un* término *es definido por inducción como sigue:*

1) Una variable es un término,
2) Una constante es un término,
3) Si f es un símbolo de función n-aria y $t_1, \ldots, t_n$ son términos, entonces, $f(t_1, \ldots, t_n)$ es un término.

Se denominará término fijo *a aquel término que no contenga variables.*

En la programación lógica, las variables representan valores desconocidos, de forma similar a las variables matemáticas. Los valores asignados a las variables son términos (expresiones). Estos valores son asignados a través de las substituciones, las cuales ligan variables a términos. Por lo tanto, una substitución es un mapeo finito de variables a términos la cuál asigna a cada variable V en su dominio un termino t distinto de V:

Definición A.3 (Substitución)**.** *Una* substitución θ *es un conjunto finito de la forma* $\{v_1/t_1, \ldots, v_n/t_n\}$*, donde cada v_i es una variable, cada t_i es un término distinto de v_i y las variables $v_1, \ldots, v_n$ son distintas. A cada elemento v_1/t_1 se lo denomina* ligadura *para v_i. Se denominará* substitución fija *(grounded substitution en ingles) a θ si los t_i son todos términos fijos.*

A continuación se extenderá el lenguaje de los términos para llegar al lenguaje de programa. Primero, se definirá a los átomos como predicados n-arios compuestos por términos, para luego definir a los literales como átomos y átomos negados.

Definición A.4 (Átomo)**.** *Si p es un símbolo de predicado n-ario y $t_1, \ldots, t_n$ son términos, entonces $p(t_1, \ldots, t_n)$ es un* átomo*. Se denominará* átomo fijo *a aquél átomo cuyos términos $t_i, 1 \leq i \leq n$ sean fijos.*

Definición A.5 (Literal)**.** *Un* literal *es un átomo o la negación de un átomo. Un* literal positivo *es un átomo. Un* literal negativo *es la negación de un átomo. Sea* A *un átomo, el literal positivo correspondiente se notará* A*, mientras que el literal negativo correspondiente se notará con* not A*.*

A partir de este momento, es posible definir como son las cláusulas de programa lógico, las cuales están compuestas por un átomo en la cabeza y una secuencia de literales en el cuerpo. El hecho de incluir literales en el cuerpo de la cláusula permite el uso de átomos negados. Es decir, es posible emplear la negación por falla en las cláusulas y, por ende, en los programas.

Definición A.6 (Cláusula de Programa Lógico)**.** *Una* cláusula de programa lógico *es una cláusula de la forma $A \leftarrow B_1, \ldots, B_n$ donde A es un átomo, y $B_1, \ldots, B_n$ son literales. Al átomo A se lo denomina la cabeza y a $B_1, \ldots, B_n$ se lo denomina el cuerpo de la cláusula de programa lógico.*

Luego, se definirá las consultas lógicas como una secuencia de literales. Como se verá a continuación, la sintaxis de la consulta resulta muy similar al de una cláusula de programa lógico sin cabeza, pese a que su significado es distinto.

Definición A.7 (Consulta Lógica)**.** *Una* consulta lógica *es una cláusula de la forma $\leftarrow B_1, \ldots, B_n$, es decir, una cláusula con el consecuente vacío. Cada $B_i (i = 1, \ldots, n)$ es llamada una subconsulta de la consulta.*

Una cláusula de programa lógico $A \leftarrow B_1, \ldots, B_n$ puede ser interpretada en la lógica de primer orden como la implicación $\forall x_1 \ldots \forall x_k (B_1 \wedge \ldots \wedge B_n \rightarrow A)$. Mientras que, una consulta lógica $\leftarrow B_1, \ldots, B_n$ significa, en la lógica de primer orden, la fórmula $\exists x_1 \ldots \exists x_k (B_1 \wedge \ldots \wedge A_n)$.

Finalmente, para completar la sintaxis de la programación lógica, se introducirá la definición de programa lógico, el cuál estará compuesto por un conjunto de cláusulas de programa lógico.

Definición A.8 (Programa Lógico)**.** *Un* programa lógico *es un conjunto finito de cláusulas de programa lógico.*

A.2. Semántica

Con el objetivo de entender el significado de un programa lógico, en esta sección se definirá su semántica. La semántica declarativa de un programa lógico está dada por la semántica de las fórmulas de la lógica de primer orden. A continuación se introducirá los conceptos de interpretaciones y modelos, concentrándonos particularmente en una clase importante de interpretaciones como lo son las interpretaciones de Herbrand.

Una interpretación consiste simplemente de un dominio de discurso sobre el cual toman valores las variables y asigna a cada constante un elemento del dominio, a cada símbolo de función un mapeo al dominio y a cada símbolo de predicado una relación del dominio. Es decir, una interpretación especifica un significado para cada símbolo de una cláusula. Resultan de especial interés aquellas interpretaciones para las cuales la cláusula toma un valor de verdad positivo, en cuyo caso dicha interpretación corresponde a un modelo de la cláusula.

Definición A.9 (Interpretación)**.** *Una* interpretación *I para un programa lógico P consiste de lo siguiente:*

- *Un conjunto no vacío D, llamado* dominio *de la interpretación,*
- *Para cada constante en P, la asignación de un elemento en D,*
- *Para cada símbolo de función n-ario en P, la asignación de un mapeo de D^n a D, y*
- *Para cada símbolo de predicado n-ario en P, la asignación de un mapeo de D^n en el conjunto $\{verdadero, falso\}$.*

Definición A.10 (Modelo)**.** *Sea I una interpretación de un programa lógico P, y sea F una cláusula de P. La interpretación I será un* modelo *de F si F es* `verdadero` *en I.*

Definición A.11 (Modelo de Programa)**.** *Sea P un programa lógico y sea I una interpretación para P. La interpretación I será* modelo de programa *para P si es modelo de cada cláusula de programa de P.*

En esta tesis se utilizará la alternativa de modelos canónicos para definir la semántica declarativa de los programas lógicos, es decir, para cada programa lógico P, se utilizará uno de sus modelos como el modelo canónico "$CM(P)$". Este modelo determina el valor de verdad de una respuesta a una consulta dada. En particular, una consulta sin variables puede tener éxito, en el caso de que sea verdadera en $CM(P)$, o fallar en caso contrario.

El modelo canónico será seleccionado entre los modelos de `Herbrand` de P, es decir, entre los modelos cuyo universo es el conjunto de términos fijos del lenguaje de P, y cuyos objetos y funciones constantes son interpretados de forma tal que cada término fijo se denota a sí mismo. Un modelo de Herbrand está completamente determinado por los átomos fijos verdaderos, y puede ser identificado con este conjunto de átomos. Un modelo de Herbrand de P es minimal si no existe ningún subconjunto propio de dicho modelo que sea modelo de Herbrand de P. Teniendo en cuenta que un programa sin átomos negados tiene exactamente un modelo de Herbrand mínimo, se seleccionará a este modelo como el modelo canónico $CM(P)$.

En cambio, los programas con negación por falla pueden tener varios modelos de Herbrand mínimos. Por lo tanto, se definirá el modelo canónico en base a la semántica de modelo estable definida por Gelfond et. al [GL88]. Esta semántica se basa en asumir hipotéticamente que algunos literales del programa son verdaderos y otros falsos. Con esta asunción, es posible construir un programa lógico sin átomos negados y obtener el modelo canónico correspondiente. Si este modelo canónico concuerda con las asunciones hechas, entonces éstas asunciones constituyen un modelo estable del programa.

Definición A.12 (Operador de Gelfond-Lifschitz)**.** *Sea P un programa lógico e I una interpretación. La transformación-GL de P módulo I es el programa P_I obtenido a partir de P luego de aplicarle las siguientes operaciones:*

- *Remover de P todas las reglas conteniendo un literal negado $notA$ tal que $A \in I$,*
- *Remover de todas las reglas restantes, todos los átomos negados.*

Es claro que el programa resultante P_I es un programa lógico sin átomos negados, y por lo tanto, tiene un único modelo de Herbrand mínimo. En el caso que este modelo coincida con la interpretación I, entonces I será un modelo estable de P.

Definición A.13 (Modelo Estable)**.** *Una interpretación I de un programa lógico P es un* modelo estable *de P sssi I es el mínimo modelo de Herbrand de P_I.*

La semántica de modelo estable es definida para un programa lógico P si P tiene exactamente un modelo estable, y define a dicho modelo como el modelo canónico de P. En esta tesis se utilizará la semántica de modelo estable para definir el significado de los programas lógicos.

Los programas lógicos se computan utilizando un método de resolución denominado resolución-SLD. Esto corresponde a la interpretación computacional de los programas lógicos, también denominado interpretación procedural, y su explicación está fuera del alcance de esta tesis. De todas formas, la unificación es una parte principal en este método de resolución y corresponde al proceso por el cual se computan las substituciones, es decir, las respuestas.

Definición A.14 (Instancia de Consulta Lógica)**.** *Sea $\theta = \{v_1/t_1, \ldots, v_n/t_n\}$ una substitución y C una consulta lógica de la forma $\leftarrow B_1, \ldots, B_m$. Luego, se notará a la* instancia *de C por θ con $C\theta$, siendo, dicha expresión, obtenida a partir de C luego de reemplazar simultáneamente cada ocurrencia de la variable v_i en C por el término t_i ($i = 1, \ldots, n$). Si $C\theta$ es fija, entonces será denominada* instancia fija *de C.*

Definición A.15 (Respuesta para una Consulta Lógica)**.** *Sea C una consulta lógica de la forma $\leftarrow B_1, \ldots, B_n$ y P un programa lógico. Sea $CM(P)$ el modelo canónico de P. Una respuesta para C en P es una substitución θ tal que la instancia $C\theta$ de la consulta lógica C es una instancia fija y para cada $B_i\theta$ ($i = 1, \ldots, n$):*

- *si $B_i\theta$ es un* literal positivo *L, entonces $L \in CM(P)$.*
- *si $B_i\theta$ es un* literal negativo *not L, entonces $L \notin CM(P)$.*

Esta última definición establece cómo es una respuesta válida para una consulta. Es claro que, a partir de esta definición, puede existir más de una respuesta válida para una misma consulta. En esta tesis se utilizarán las definiciones introducidas en este anexo de

programa lógico, consulta lógica y respuesta para una consulta lógica como base para definir el marco de representación de conocimiento para la programación lógica. A partir de este marco de representación de conocimiento se definen casos particulares de los conceptos de consultas contextuales y *SRs* introducidos en esta tesis.

Apéndice B

Programación Lógica Rebatible (DeLP)

La Programación Lógica Rebatible (DeLP por su sigla en inglés) puede ser utilizada básicamente en cualquier aplicación que requiera representación de conocimiento. Sus características le permiten modelar conocimiento que involucre información incompleta o potencialmente contradictoria. El mecanismo de inferencia sobre el cual esta basado permite decidir entre conclusiones contradictorias y adaptarse fácilmente a entornos cambiantes.

Un programa lógico rebatible esta formado por un conjunto de *hechos*, *reglas estrictas* y *reglas rebatibles*. Estas últimas permiten la representación de información tentativa. Toda conclusión del programa deberá ser sustentada por algún *argumento* que pueda construirse utilizando las reglas y los hechos del programa. Cuando se utilicen reglas rebatibles para derivar una conclusión C, esta conclusión será tentativa y podrá ser refutada por información que la contradiga. Para decidir cuando una conclusión C puede aceptarse a partir de un programa lógico rebatible, se realizará un *análisis dialéctico*, construyendo argumentos a favor y en contra de la conclusión C.

Un argumento que sustenta una conclusión C podrá ser atacado por otros argumentos *derrotadores* que lo contradigan y que pueden construirse a partir del programa. Dichos derrotadores podrán a su vez ser atacados, y así sucesivamente, generando una secuencia de argumentos llamada *línea de argumentación*. Cada línea de argumentación deberá satisfacer ciertas propiedades, a fin de evitar que se generen argumentaciones falaces. El

proceso completo considera para cada argumento todos sus posibles argumentos derrotadores, lo cual, en lugar de una única línea de argumentación, genera un conjunto de líneas representadas con un *árbol de dialéctica*. Este análisis dialéctico determinará si la conclusión en cuestión está *garantizada* o no, a partir del programa.

En este apéndice se definirán formalmente los conceptos de programa lógico rebatible, consulta lógica rebatible y respuesta para una consulta lógica rebatible. Para estas definiciones se utilizarán ciertos conceptos y terminología estándar de la programación lógica introducidos en el apéndice anterior. Primero se definirá la sintaxis de los programas lógicos rebatibles para luego introducir las derivaciones rebatibles que permitirán realizar una análisis dialéctico. Este análisis dialéctico será la base para la definición de la respuesta a una consulta lógica rebatible.

B.1. Sintaxis

Un programa lógico rebatible estará compuesto por *hechos*, *reglas estrictas* y *reglas rebatibles*, cuya sintaxis se define a continuación:

Definición B.1 (Hecho)**.** *Un hecho es un literal fijo L. Esto es, un átomo fijo, o un átomo fijo negado.*

Definición B.2 (Regla Estricta)**.** *Una* regla estricta *es un par ordenado, denotado "Cabeza* $\leftarrow$ *Cuerpo", donde el primer elemento, Cabeza, es un literal fijo, y el segundo elemento, Cuerpo, es un conjunto finito no vacío de literales fijos. Una regla estricta con cabeza* L_0 *y cuerpo* $\{L_1, \ldots, L_n\}$ *(n>0) se escribirá también como:* $L_0 \leftarrow L_1, \ldots, L_n$.

Definición B.3 (Regla Rebatible)**.** *Una* regla rebatible *es un par ordenado, denotado "Cabeza* $-\!\prec$ *Cuerpo", donde el primer elemento Cabeza, es un literal fijo, y el segundo elemento, Cuerpo, es un conjunto finito no vacío de literales fijos. Una regla rebatible con cabeza* L_0 *y cuerpo* $\{L_1, \ldots, L_n\}$ *(n>0) se escribirá también como:* $L_0 -\!\prec L_1, \ldots, L_n$.

Sintácticamente, el símbolo "$-\!\prec$ " es lo único que distingue a una regla rebatible de una estricta. Pragmáticamente, las *reglas rebatibles* se utilizarán para representar conocimiento rebatible, esto es, información tentativa que puede utilizarse en la medida que no exista información que la contradiga. Las *reglas estrictas* representarán conocimiento seguro y libre de excepciones, esto es, siempre que se crea en los literales que forman el cuerpo, se podrá creer con la misma seguridad en el literal que corresponde a la cabeza.

Definición B.4 (Programa Lógico Rebatible)**.** *Un programa lógico rebatible $\mathcal{P}_{delp}$ es un conjunto de hechos, reglas estrictas y reglas rebatibles. En un programa $\mathcal{P}_{delp}$ se identificarán con Θ al conjunto de hechos, con Ω al conjunto de reglas estrictas y con Δ al conjunto de reglas rebatibles. Por conveniencia, se notará con Π al conjunto $\Theta \cup \Omega$. Por lo tanto, en general se denotará a un programa lógico rebatible $\mathcal{P}_{delp}$ con el par (Π, Δ).*

Es importante destacar, que aunque las reglas de un programa utilizan únicamente literales fijos, de acuerdo a la Observación 2.1 de [Gar01], en algunos ejemplos se utilizarán variables solamente como una forma de denotar *esquemas de reglas*. A continuación se introducirá el concepto de consulta lógica rebatible, el cual corresponderá a un literal.

Definición B.5 (Consulta Lógica Rebatible)**.** *Una* consulta lógica rebatible *corresponde a un literal. Si el literal es fijo, la consulta será una* consulta fija lógica rebatible*. En caso contrario, será una* consulta esquemática lógica rebatible*.*

El objetivo de una consulta fija lógica rebatible es saber si el literal fijo correspondiente está garantizado a partir de un programa lógico rebatible. En la siguiente sección se describirá cómo se calcula el estado de garantía para un literal fijo. En el caso de una consulta esquemática lógica rebatible, el objetivo es determinar si existe algún literal fijo que sea instancia de la consulta esquemática y que esté garantizado a partir del programa lógico rebatible correspondiente.

B.2. Argumentación Rebatible

En esta sección se introducirá el mecanismo por el cuál se computan las garantías para las consultas lógicas rebatibles, siendo éste el componente principal del cálculo de la respuesta a una consulta. En un principio, se incluirá la definición de derivación rebatible para una consulta fija lógica rebatible definida originalmente en [Gar01].

Definición B.6 (Derivación Rebatible de un literal fijo)**.** *Sea $\mathcal{P}_{delp} = (\Pi, \Delta)$ un programa lógico rebatible y L un literal fijo. Una derivación rebatible para L a partir de $\mathcal{P}_{delp}$, consiste de una secuencia finita de literales fijos $L_1, L_2, \ldots, L_n = L$, provisto de que exista una secuencia de reglas estrictas o rebatibles $R_1, R_2, \ldots, R_n$ del programa $\mathcal{P}_{delp}$, de tal forma que para cada literal L_i en la secuencia:*

(a) L_i *es un hecho, o*

(b) existe en $\mathcal{P}_{delp}$ *una regla* R_i *con cabeza* L_i *y cuerpo* $B_1, B_2, \ldots, B_k$ *donde todo literal fijo* B_j *del cuerpo* $(1 \leq j \leq k)$ *es un elemento de la secuencia que precede a* L_i*.*

La derivación se dice *rebatible*, porque aunque un literal L pueda ser derivado, puede existir en el programa información que contradiga a L, y entonces L no será aceptado como una creencia válida del programa. Existe un caso especial de derivación, el cuál es denominado derivación estricta y corresponde al caso donde todas las reglas utilizadas en la derivación son estrictas. Este tipo de derivaciones no pueden ser refutadas ya que se basan en reglas que no son rebatibles. A continuación, se introduce la definición de derivación estricta, tomada de [Gar01]

Definición B.7 (Derivación Estricta de un literal fijo)**.** *Sea* $\mathcal{P}_{delp} = (\Pi, \Delta)$ *un programa lógico rebatible y* L *un literal fijo para el cual existe una derivación rebatible* $L_1, L_2, \ldots, L_n = L$*. El literal* L *tienen una* derivación estricta *si todas las reglas de programa utilizadas para obtener la secuencia* $L_1, L_2, \ldots, L_n = L$ *son reglas estrictas.*

Teniendo en cuanta que las reglas de programa lógico rebatible permiten utilizar literales negados en la cabeza, es posible derivar literales complementarios. Por lo tanto, puede ocurrir que un conjunto de reglas sea contradictorio, es decir, que a partir de dicho conjunto de reglas se puedan obtener derivaciones para un literal L como para su complemento. De acuerdo a la Observación 2.4 de [Gar01], se asumirá que todo programa lógico rebatible posee un conjunto de hechos y reglas estrictas no contradictorio.

DeLP utiliza como procedimiento de prueba un mecanismo de análisis global para decidir que literales fijos serán aceptados a partir de un programa. Este procedimiento de prueba permitirá definir argumentos para un literal fijo y contra-argumentos para los argumentos obtenidos. Luego, un análisis dialéctico permitirá decidir cuando un literal fijo está aceptado. A continuación se muestra la noción de estructura de argumento para programas lógicos rebatibles.

Definición B.8 (Estructura de argumento)**.** *Sea* h *un literal fijo y* $\mathcal{P}_{delp} = (\Pi, \Delta)$ *un programa lógico rebatible, una* estructura de argumento *para* h *es un par* $\langle \mathcal{A}, h \rangle$*, donde* $\mathcal{A}$ *es un conjunto de reglas rebatibles de* Δ*, tal que:*

1) existe una derivación rebatible para h *a partir de* $\Pi \cup \mathcal{A}$*.*

2) $\Pi \cup \mathcal{A}$ *es no contradictorio, y*

3) $\mathcal{A}$ *es minimal, es decir, no existe un subconjunto propio* $\mathcal{A}'$ *de* $\mathcal{A}$ *tal que* $\mathcal{A}'$ *satisface las condiciones (1) y (2).*

Dos literales complementarios representan información estrictamente contradictoria. La siguiente definición generaliza este concepto:

Definición B.9 (Literales en Desacuerdo)**.** *Sea* $\mathcal{P}_{delp} = (\Pi, \Delta)$ *un programa lógico rebatible y* Π *el conjunto de reglas estrictas y hechos del programa. Dos literales* h *y* h_1 *están en* desacuerdo*, si y sólo si el conjunto* $\Pi \cup \{h, h_1\}$ *es contradictorio.*

A continuación se definirá una noción de conflicto entre dos estructuras de argumento basada en la negación fuerte. Esto es, dos argumentos estarán en conflicto cuando sustenten conclusiones en desacuerdo.

Definición B.10 (Contra-argumento o ataque)**.** *Sean* $\langle \mathcal{A}_1, h_1 \rangle$ *y* $\langle \mathcal{A}_2, h_2 \rangle$ *dos estructuras de argumentos obtenidas a partir de un programa* $\mathcal{P}_{delp}$*. La estructura de argumento* $\langle \mathcal{A}_1, h_1 \rangle$ contra-argumenta *a* $\langle \mathcal{A}_2, h_2 \rangle$ *en el literal fijo* h*, si y sólo si, existe una estructura de argumento* $\langle \mathcal{A}, h \rangle$ *tal que* $\mathcal{A} \subseteq \mathcal{A}_2$ *y, además,* h *y* h_1 *están en desacuerdo. El argumento* $\langle \mathcal{A}, h \rangle$ *se llama* sub-argumento de desacuerdo*, y el literal fijo* h *será el* punto de contra-argumentación*.*

Un problema central en los formalismos de argumentación rebatible es la definición de un criterio que permita comparar argumentos. En el caso de los sistemas más abstractos de argumentación rebatible, usualmente se asume un orden pre-establecido entre todos los argumentos posibles. Otros formalismos proponen asumir un orden definido explícitamente dentro las reglas del programa (prioridades), con lo cual, las reglas de mayor prioridad son preferidas. Otra alternativa es utilizar el criterio de especificidad definido originalmente por David Poole [Poo85]. La especificidad permite comparar reglas o argumentos sin la necesidad de especificar explícitamente las prioridades.

Un análisis más detallado acerca de los criterios de comparación de argumentos puede encontrarse en [Gar01], En el caso de DeLP, éste permite la implementación de distintos criterios de comparación. En particular, utiliza por defecto especificidad, pero podría definirse cualquier otro mecanismo de comparación de argumentos. Es por esto que, en lo que sigue, se asumirá que existe un orden parcial "$\succ$" que permite distinguir cuando una

estructura de argumento es mejor que otra. Por ejemplo, si $\langle \mathcal{A}_1, h_1 \rangle$ es mejor que $\langle \mathcal{A}, h \rangle$, denotado $\langle \mathcal{A}_1, h_1 \rangle \succ \langle \mathcal{A}, h \rangle$.

Definición B.11 (Derrotador). *Sean $\langle \mathcal{A}_1, h_1 \rangle$ y $\langle \mathcal{A}_2, h_2 \rangle$ dos estructuras de argumento. La estructura $\langle \mathcal{A}_1, h_1 \rangle$ es un derrotador para $\langle \mathcal{A}_2, h_2 \rangle$ en el literal fijo h si existe una estructura de argumento $\langle \mathcal{A}, h \rangle$ tal que, $\mathcal{A} \subseteq \mathcal{A}_2$, $\langle \mathcal{A}_1, h_1 \rangle$ contra-argumenta a $\langle \mathcal{A}_2, h_2 \rangle$ en el literal fijo h y, o bien:*

- *$\langle \mathcal{A}_1, h_1 \rangle$ es mejor que $\langle \mathcal{A}, h \rangle$ con respecto al criterio de comparación que se utilice ($\langle \mathcal{A}_1, h_1 \rangle \succ \langle \mathcal{A}, h \rangle$), o*
- *ni $\langle \mathcal{A}_1, h_1 \rangle$ es mejor que $\langle \mathcal{A}, h \rangle$, ni $\langle \mathcal{A}, h \rangle$ es mejor que $\langle \mathcal{A}_1, h_1 \rangle$, con respecto al criterio de comparación que se utilice ($\langle \mathcal{A}_1, h_1 \rangle \not\succ \langle \mathcal{A}, h \rangle$, y $\langle \mathcal{A}, h \rangle \not\succ \langle \mathcal{A}_1, h_1 \rangle$).*

Dado el conjunto de estructuras de argumento que pueden obtenerse de un programa $\mathcal{P}_{delp}$, la relación de derrota entre argumentos sólo puede establecer un orden entre dos argumentos en conflicto. Sin embargo, el estado de un argumento $\langle \mathcal{A}_1, h_1 \rangle$ con respecto al programa $\mathcal{P}_{delp}$ dependerá de la interacción de $\langle \mathcal{A}_1, h_1 \rangle$ con el resto de los argumentos que puedan obtenerse.

Por ejemplo, a partir de un programa $\mathcal{P}_{delp}$ puede obtenerse una estructura de argumento que no tenga derrotadores, en cuyo caso, un agente con este programa podría creer en el literal fijo que soporta el argumento antes mencionado. Sin embargo, puede ocurrir que exista un derrotador para el argumento antes mencionado, invalidando dicha creencia. También puede ocurrir que el argumento derrotador tenga a su vez otro argumento que lo derrote, restaurando la creencia inicial. Esta secuencia podría ir aún más lejos, lo cual originará una secuencia de argumentos, denominada *línea de argumentación*, donde cada elemento es un derrotador de su predecesor.

Definición B.12 (Línea de Argumentación). *Sea $\mathcal{P}_{delp}$ un programa lógico rebatible, y $\langle \mathcal{A}_0, h_0 \rangle$ una estructura de argumento obtenida a partir de $\mathcal{P}_{delp}$. Una* línea de argumentación *a partir de $\langle \mathcal{A}_0, h_0 \rangle$ es una secuencia de estructuras de argumentos (obtenidas a partir de $\mathcal{P}_{delp}$) denotada $\Lambda = [\langle \mathcal{A}_0, h_0 \rangle, \langle \mathcal{A}_1, h_1 \rangle, \langle \mathcal{A}_2, h_2 \rangle, \langle \mathcal{A}_3, h_3 \rangle, \ldots]$, donde para cada elemento de la secuencia $\langle \mathcal{A}_i, h_i \rangle$, el elemento siguiente $\langle \mathcal{A}_{i+1}, h_{i+1} \rangle$ es un derrotador para $\langle \mathcal{A}_i, h_i \rangle$.*

El primer elemento de una línea de argumentación $[\langle \mathcal{A}_0, h_0 \rangle, \langle \mathcal{A}_1, h_1 \rangle, \langle \mathcal{A}_2, h_2 \rangle, \langle \mathcal{A}_3, h_3 \rangle, \ldots]$ es una estructura que sustenta a un literal

fijo h_0. El segundo elemento de la línea de argumentación derrota al primero y por consiguiente interfiere con la creencia en h_0. El tercer elemento derrota al segundo y por consiguiente, indirectamente sustenta a h_0, y así siguiendo. De esta manera, en una línea de argumentación se distinguirán dos conjuntos disjuntos de argumentos: argumentos de soporte para h_0 y de interferencia para h_0.

Es claro que una línea de argumentación no es suficiente para analizar la interacción entre argumentos y decidir sobre la creencia de un dado literal fijo a partir de un determinado programa. Esto se debe a que para cada estructura de argumento pueden existir más de un derrotador, dando origen a un *árbol de derrotadores*. Además, existen ciertas restricciones que deben cumplir las líneas de argumentación para que sean aceptables, como por ejemplo, que no contengan argumentos que se derrotan a sí mismos, o que no existan derrotadores recíprocos. En [Gar01] se desarrolla análisis mas detallado acerca de las restricciones de las lineas de argumentación.

La presencia de múltiples derrotadores para una estructura de argumento produce una ramificación de líneas de argumentación, dando origen a los *árboles de derrotadores* o *árboles de dialéctica*. En estos árboles, cada camino desde la raíz hasta una hora corresponde a una línea de argumentación.

Definición B.13 (Árbol de dialéctica)**.** *Sea $\langle \mathcal{A}_0, h_0 \rangle$ una estructura de argumento obtenida a partir de un programa $\mathcal{P}_{delp}$. Un árbol de dialéctica para $\langle \mathcal{A}_0, h_0 \rangle$, a partir de $\mathcal{P}_{delp}$, se denota $\mathcal{T}_{\langle \mathcal{A}_0, h_0 \rangle}$, y se construye de la siguiente forma:*

- *La raíz del árbol es etiquetada con $\langle \mathcal{A}_0, h_0 \rangle$.*
- *Sea N un nodo del árbol etiquetado $\langle \mathcal{A}_n, h_n \rangle$, y $[\langle \mathcal{A}_0, h_0 \rangle, \langle \mathcal{A}_1, h_1 \rangle, \langle \mathcal{A}_2, h_2 \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle]$ la secuencia de etiquetas del camino que va desde la raíz hasta el nodo N.*
  Sean $[\langle \mathcal{B}_1, q_1 \rangle, \langle \mathcal{B}_2, q_2 \rangle, \ldots, \langle \mathcal{B}_k, q_k \rangle]$ todos los derrotadores de $\langle \mathcal{A}_n, h_n \rangle$.
  Para cada derrotador $\langle \mathcal{B}_i, q_i \rangle$ $(1 \leq i \leq k)$, tal que la línea de argumentación $[\langle \mathcal{A}_0, h_0 \rangle, \langle \mathcal{A}_1, h_1 \rangle, \langle \mathcal{A}_2, h_2 \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle, \langle \mathcal{B}_i, q_i \rangle]$ sea aceptable, existe un nodo hijo N_i de N etiquetado con $\langle \mathcal{B}_i, q_i \rangle$.
  Si no existe ningún derrotador $\langle \mathcal{B}_i, q_i \rangle$ en tales condiciones, entonces el nodo N es una hoja.

Es claro que los nodos hoja del árbol de dialéctica corresponden a argumentos no derrotados. En cambio, un nodo interno que tiene como hijo a un nodo hoja, corresponderá a un argumento derrotado. Siguiendo con este análisis, desde las hojas hacia la raíz, los nodos de un árbol de dialéctica se pueden marcar como "D" *derrotado*, o "U" *no derrotado* (en inglés, undefeated). Este marcado representa el análisis dialéctico en el cual todos los argumentos construibles a partir de un programa $\mathcal{P}_{delp}$ son considerados a fin de decidir el estado de un argumento $\langle \mathcal{A}, h \rangle$.

A partir de un árbol de dialéctica marcado se puede definir qué es un literal fijo garantizado. Decir que un literal fijo "h" está garantizado a partir de un programa $\mathcal{P}_{delp}$ significa que un razonador que utilice el programa $\mathcal{P}_{delp}$ podrá creer en "h". La respuesta para una consulta lógica rebatible estará directamente relacionada al estado de garantía de las instancias fijas de la misma.

Definición B.14 (Literales fijos garantizados)**.** *Sea $\mathcal{P}_{delp} = (\Pi, \Delta)$ un programa lógico rebatible y h un literal fijo. Sea $\langle \mathcal{A}, h \rangle$ una estructura de argumento para h y $\mathcal{T}^*_{\langle \mathcal{A}, h \rangle}$ el árbol de dialéctica marcado asociado a $\langle \mathcal{A}, h \rangle$. El literal fijo h está* garantizado *si la raíz de $\mathcal{T}^*_{\langle \mathcal{A}, h \rangle}$ está marcada como* U*.*

Definición B.15 (Respuesta a una Consulta Lógica Rebatible)**.** *Una respuesta para la consulta lógica rebatible Q en el programa lógico rebatible $\mathcal{P}_{delp}$ es el par:*

- (SI, q)*, si $\exists q$ tal que q es una instancia fija de Q y está garantizado a partir de $\mathcal{P}_{delp}$;*
- $(\text{INDECISO}, q)$*, si no existe una instancia fija de Q que esté garantizada a partir de $\mathcal{P}_{delp}$ y $\exists q$ tal que es instancia fija de Q y ni q ni su complemento están garantizados a partir de $\mathcal{P}_{delp}$;*
- (NO, Q)*, si $\forall q$ tal que q es una instancia fija de Q, el complemento de q está garantizado a partir de $\mathcal{P}_{delp}$ o*
- $(\text{DESCONOCIDO}, Q)$*, si Q no pertenece al lenguaje del programa $\mathcal{P}_{delp}$.*

En el caso que la consulta lógica rebatible Q_F sea fija, existe una única instancia fija para esa consulta lógica rebatible y es Q_F. Por lo tanto, la respuesta a dicha consulta corresponde al estado de garantía de la misma, es decir, SI en el caso de que Q_F esté garantizada, NO en el caso de que el complemento de Q_F esté garantizada, $INDECISO$ si no

se da ninguno de los casos anteriores. Finalmente, la respuesta será $DESCONOCIDO$ si la consulta no pertenece al lenguaje del programa.

Por otro lado, si la consulta lógica rebatible Q_e es esquemática, una respuesta válida a dicha consulta indicará una de las siguientes tres posibilidades: (a) la existencia de una instancia fija de Q_e que esté garantizada, (b) ante la inexistencia de una instancia fija garantizada, una cuyo complemento tampoco esté garantizado o, finalmente, (c) el hecho de que para todos las instancias fijas de Q_e, su complemento esta garantizado. De esta forma, es posible utilizar una consulta lógica rebatible esquemática para obtener información acerca de la existencia de literales fijos que sean instancia de dicha consulta y su estado de garantía.

Las consultas esquemáticas lógicas rebatibles pueden tener más de una respuesta válida, siendo que pueden existir más de una instancia fija de la consulta garantizadas, o bien, pueden existir más de una instancia fija no garantizadas y tampoco sus respectivos complementos. Las definiciones incluidas en este apéndice de programa lógico rebatible, consulta lógica rebatible y respuesta para una consulta lógica rebatible permiten la definición de un marco de representación de conocimiento para la programación lógica rebatible. A partir de este marco de representación de conocimiento es posible definir, como se muestra en los Capítulos 2 y 3, consultas contextuales y *SRs* específicos para la programación lógica rebatible.

Bibliografía

[ALP+94] Alferes, J. J., Leite, J. A., Pereira, L. M., Przymusinska, H., and Przymusinski, T. C. Dynamic logic programming. In *Proc. of the International Conference on Principles of Knowledge Representation and Reasoning (KR'98)* (1994), Morgan Kauffmann, pp. 98–111.

Este trabajo presenta el paradigma de la Programación Lógica Dinámica, el cuál facilita la modularización de la Programación Lógica, así como también la modularización del razonamiento no monótono. Este paradigma permite una composición de actualizaciones de bases de conocimiento representadas con Programas Lógicos Generalizados.

[Apt97] Apt, K. R. *From logic programming to Prolog*. Prentice Hall International series in computer science. Prentice Hall, 1997.

[AT07] Altman, A., and Tennenholtz, M. Incentive compatible ranking systems. In *AAMAS* (2007), E. H. Durfee, M. Yokoo, M. N. Huhns, and O. Shehory, Eds., IFAAMAS, p. 84.

El ranking de agentes basado en el feedback obtenido de otros agentes es fundamental en Sistemas Multi-Agentes. A partir de la información obtenida de los distintos agentes se computa un ranking social de las distintas alternativas existentes. Este trabajo analiza el problema de los incentivos en los sistemas de ranking, donde los agentes actúan en pos de maximizar su posición en el ranking, sin priorizar la obtención del resultado correcto. El trabajo considera dos nociones diferentes de compatibilidad de incentivos y diversas propiedades básicas de los sistemas de ranking, mostrando que en general no existen sistemas de ranking compatibles con incentivos que satisfagan dichas propiedades.

[Bar03] Baral, C. *Knowledge Representation, Reasoning, and Declarative Problem Solving*. Cambridge University Press, Cambridge, England, 2003.

En este libro, el autor propone el uso del lenguaje no monótono AnsProlog (Programación Lógica con semántica de Answer Set) para la representación de conocimiento y razonamiento, y la resolución de problemas declarativamente.

[BBM95] Buvač, S., Buvač, V., and Mason, I. Metamathematics of contexts. *Fundamenta Mathematicae 23*, 3 (1995). Available from http://www-formal.stanford.edu/buvac.

Este artículo estudia las propiedades lógicas de los contextos, completando la formalización iniciada en el trabajo [BM93]. Incluye, además, una comparación de la semántica resultante con la semántica de Kripke.

[BCM+03] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., Eds. *The Description Logic Handbook: Theory, Implementation, and Applications* (2003), Cambridge University Press.

Description Logics corresponden a un conjunto de lenguajes de representación de conocimiento que han sido estudiados en los últimos tiempos en inteligencia artificial. Este libro cubre todos los aspectos de la investigación en este campo, incluyendo teoría, implementación y aplicaciones.

[BM93] Buvač, S., and Mason, I. Propositional logic of context. In *Proceedings of the Eleventh National Conference on Artificial Intelligence* (Menlo Park, California, 1993), R. Fikes and W. Lehnert, Eds., American Association for Artificial Intelligence, AAAI Press, pp. 412–419.

Este artículo analiza las propiedades lógicas de los contextos, describiéndo la sintaxis y semántica de un lenguaje general de tipo proposicional para trabajar con ellos. Incluye, además, un sistema de prueba de estilo Hilbert para el lenguaje definido. Los principales resultados del trabajo consisten en la demostración de la sanidad y la completitud de este sistema de prueba.

[Bra97] Bradshaw, J. M. *An Introduction to Software Agents*. Bradshaw (editor), 1997, ch. 1, pp. 3–46.

En este capítulo se tratan dos marcados enfoques relacionados a la definición de agente: uno basado en la noción de *agenthood*, y el otro basado en la descripción

de los atributos que los agentes de software deben poseer. Estas perspectivas complementarias son resumidas en este capítulo. Luego se discute "porque" los agentes de software están relacionados a dos asuntos prácticos: 1) simplificar la complejidad de la computación distribuida y 2) sobrellevar las limitaciones de interface de usuario actual.

[Bre05] Brezillon, P. Task-realization models in contextual graphs. In *Modeling and Using Context: 5th International and Interdisciplinary Conference*, A. Dey, B. Kokinov, D. Leake, and R. Turner, Eds. Springer-Verlag, Berlin, 2005, pp. 55–68.

Este artículo presenta un marco basado en un formalismo de representación para el modelado de la realización de tareas por usuarios, denominado gráficos contextuales. Este formalismo se basa en una definición de contexto presentada en este trabajo, el cuál es considerado la suma de dos tipos de conocimiento, uno relevante para dicho enfoque en particular, y uno no relevante.

[CR93] Colmerauer, A., and Roussel, P. The birth of prolog. In *Proceedings of the Conference on History of Programming Languages* (New York, NY, USA, April 1993), R. L. Wexelblat, Ed., vol. 28(3) of *ACM Sigplan Notices*, ACM Press, pp. 37–52.

[DLC87] Durfee, E. H., Lesser, V. R., and Corkill, D. D. Coherent cooperation among communicating problem solvers. *IEEE Trans. Comput. 36* (November 1987), 1275–1291.

Este trabajo aborda la problemática de cooperación coherente entre agentes de un SMA distribuido cuyo objetivo es el de resolver problemas de manera conjunta. La propuesta resalta la importancia de un control local sofisticado por el cual cada agente integra su conocimiento del problema con el conocimiento de la red de coordinación. Los autores describen tres mecanismos para mejorar la coherencia en la red y presentan una variedad de situaciones donde muestran los beneficios y las limitaciones de estos mecanismos.

[Dur99] Durfee, E. H. Distributed problem solving and planning. In *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, G. Weiss, Ed. The MIT Press, Cambridge, MA, USA, 1999, ch. 3, pp. 121–164.

[FG96] Franklin, S., and Graesser, A. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *ECAI '96: Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages* (London, UK, 1996), Springer-Verlag, pp. 21–35.

En este trabajo, los autores proponen una definición formal de agentes autónomos la cual distingue claramente un agente de software de un programa. También ofrecen el comienzo de una taxonomía de agentes autónomos, y discuten posibilidades para una futura clasificación. Finalmente, discuten subagentes y sistemas multiagentes.

[FIP02a] Foundation for Intelligent Physical Agents. *FIPA Abstract Architecture Specification*, December 2002. Status : Standard, version L , http://www.fipa.org/specs/fipa00001/.

El objetivo principal de FIPA es crear estándares de agentes para promover la interoperabilidad entre aplicaciones de agentes y sistemas de agentes. El trabajo realizado por FIPA incluye la especificacion de un lenguaje de comunicacion entre agnetes, servicios de agentes y ontologías que permitan el manejo de agentes.

[FIP02b] Foundation for Intelligent Physical Agents. *FIPA Query Interaction Protocol Specification*, December 2002. Status : Standard, version H , http://www.fipa.org/specs/fipa00027/.

El protocolo de interacción FIPA Query permite a un agente requerir la ejecución de un determinado tipo de acción a otro agente. El iniciador le requiere al participante que ejecute algún tipo de acción de tipo *inform*, usando un acto comunicativo específico. El participante procesa el requerimiento y decide si desea aceptar o rechazar el requerimiento. Finalmente, si el participante aceptó el requerimiento, ejecuta la acción requerida y envía el último mensaje del protocolo indicando el resultado obtenido o que ocurrió una falla, si este fuera el caso.

[FL05] Ferraris, P., and Lifschitz, V. Mathematical foundations of answer set programming. In *We Will Show Them! (1)* (2005), S. N. Artëmov, H. Barringer, A. S. d'Avila Garcez, L. C. Lamb, and J. Woods, Eds., College Publications, pp. 615–664.

Este trabajo aborda el diseño de programas ASP probablemente correctos y la teoría matemática en que se basan estos programas. Los programas ASP son

sintáctiamente similares a los programas Prolog, pero el mecanismo computacional usado para resolver sus consultas es distinto y tiene como principal objetivo lograr procedimientos de prueba más rápidos.

[Gar01] Garcia, A. J. *Programación en Lógica Rebatible: Lenguaje, Semántica Operacional y Paralelismo*. Tesis de doctorado, 2001.

[Giu93] Giunchiglia, F. Contextual reasoning. *Epistemologica 16* (1993), 345–364. Also IRST-Technical Report 9211-20, IRST, Trento, Italy.

Este trabajo presenta un análisis teórico del razonamiento basado en contextos, mencionándosee sus características y posibilidades de aplicación. Presenta, ademas, un modelo de razonamiento a metanivel utilizando contextos.

[GL88] Gelfond, M., and Lifschitz, V. The stable model semantics for logic programming. In *ICLP/SLP* (1988), pp. 1070–1080.

Este trabajo propone una nueva semántica declarativa para programas lógicos con negación. Su formulación es simple y a la vez más general que la semántica iterativa de punto fijo para programas estratificados, siendo aplicable para algunos programas que no son estratificables.

[GRTS07] García, A. J., Rotstein, N. D., Tucat, M., and Simari, G. R. An argumentative reasoning service for deliberative agents. In *KSEM* (2007), Z. Zhang and J. H. Siekmann, Eds., vol. 4798 of *Lecture Notes in Computer Science*, Springer, pp. 128–139.

En este trabajo se presenta el concepto de Servicio de Razonamiento Argumentativo para responder consultas con Contexto, en el mismo sentido al propuesto en esta tesis. En particular, este trabajo propone tres posibles operadores de contexto, priorizado, no priorizado y restrictivo, aplicables a Servicios de Razonamiento utilizando el formalizmo DeLP.

[GTS05] García, A. J., Tucat, M., and Simari, G. R. Interaction Primitives for Implementing Multi-agent Systems. In *Proceedings of the VII Argentine Symposium on Artificial Intelligence (ASAI 2005)* (Rosario, Argentina, 2005), pp. 24–35.

Las primitivas de interacción presentadas en este trabajo fueron motivadas por la implementación de sistemas multi-agente para entornos dinámicos y distribuidos,

donde un grupo de agentes se comunican y colaboran. Estas primitivas permiten, entre otras características, la creación de distintos sistemas multi-agente independientes y asociar a la recepción de mensajes específicos la ejecución de predicados Prolog. Además, estas primitivas posibilitan la implementación de lenguajes de comunicación entre agentes estándares y provee herramientas para el desarrollo de protocolos de conversación de agentes.

[Guh91] Guha, R. V. *Contexts: A Formalization and some applications*. PhD thesis, Stanford University, 1991.

En esta tesis doctoral se analiza la noción de contexto aplicada a la resolucion de un problema real, la base de conocimiento del sistema CYC. Se presenta una lógica de contextos y se analizan distintos usos de contextos para abordar problemas clásicos de Inteligencia Artificial.

[HS99] Huhns, M. N., and Stephens, L. M. Multiagent systems and societies of agents. In *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, G. Weiss, Ed. The MIT Press, Cambridge, MA, USA, 1999, ch. 2, pp. 79–120.

En este capítulo, los autores analizan y describen entornos en los cuales los agentes pueden operar de manera efectiva e interactuar entre ellos aumentando la productividad. En la sección 2.3, los autores presentan el concepto de protocolo de interacción, los cuales son utilizados para intercambiar una serie de mensajes entre agentes. Respecto de esta tesis, resultan interesantes las distintas estrategias presentadas relacionadas con los protocolos de cooperación en la subsección 2.3.2.

[HZB+06] Hull, D., Zolin, E., Bovykin, A., Horrocks, I., Sattler, U., and Stevens, R. Deciding semantic matching of stateless services. In *AAAI* (2006), AAAI Press.

Este trabajo propone un nuevo enfoque a la hora de describir y razonar acerca de servicios de procesamiento de información sin estado. Esta propuesta puede verse como una extensión de las descripciones estándares de los servicios, que hacen explícita la relación entre entrada y salida y considera las ontologías OWL para obtener el significado de los términos utilizados en la descripción del servicio.

[JFAS07] Jureta, I., Faulkner, S., Achbany, Y., and Saerens, M. Dynamic task allocation within an open service-oriented MAS architecture. In *AAMAS* (2007), E. H. Durfee, M. Yokoo, M. N. Huhns, and O. Shehory, Eds., IFAAMAS, p. 206.

Este trabajo propone un SMA cuya arquitectura consiste de centros de servicio. Cada centro de servicio contiene un mediador encargado de distribuir las tareas, asignando las mismas a los correspondientes agentes especialistas dependiendo de su performance anterior y, en el caso de agentes nuevos, anticipando su performance. Los mediadores utilizan un criterio novedoso basado en múltiples criterios, aplicando un algoritmo de aprendizaje por refuerzo.

[JSW98] Jennings, N., Sycara, K., and Wooldridge, M. A roadmap of agent research and development. In *Autonomous Agents and Multi Agent Systems* (Boston, 1998), Kluwer Academic, pp. 275–306.

Este artículo provee una perspectiva general de investigación y desarrollo de actividades en el campo de agentes autónomos y sistemas multi-agentes. El objetivo es identificar las aplicaciones y los conceptos claves, e indicar como se relacionan entre ellos. Además ofrecen un contexto histórico del área.

[Kow74] Kowalski, R. Predicate logic as programming language. In *Information Processing 74, Proceedings of IFIP congress 74* (Stockholm, Sweden, 1974), J. L. Rosenfeld, Ed., North-Holland, pp. 569–574.

[LAWG05] Lord, P. W., Alper, P., Wroe, C., and Goble, C. A. Feta: A lightweight architecture for user oriented semantic service discovery. In *ESWC* (2005), A. Gómez-Pérez and J. Euzenat, Eds., vol. 3532 of *Lecture Notes in Computer Science*, Springer, pp. 17–31.

Las arquitecturas web basadas en servicios posibilitan un alto grado de flexibilidad a la hora de buscar nuevos servicios, facilitando su descubrimiento y composición. Este trabajo propone ciertos requerimientos para la descripción de servicios en el dominio de bioinformática, demandando descripciones técnicamente simples.

[Liu99] Liu, M. Deductive database languages: Problems and solutions. *ACM Comput. Surv 31*, 1 (1999), 27–62.

Las bases de datos deductivas son el resultado de la integración de las bases de datos relacionales y las técnicas de programación Lógica. A partir de esta simple integración surgen distintos problemas, los cuales son discutidos en este trabajo, desde cuatro aspectos distintos: valores complejos, orientación a objetos, órdenes superiores, y actualizaciones. En cada caso, se examinan cuatro lenguajes típicos que buscan solucionar estos aspectos.

[Llo87] Lloyd, J. W. *Foundations of Logic Programming*, 2 ed. Springer, Berlin, 1987.

[LMR92] Lobo, J., Minker, J., and Rajasekar, A. *Foundations of Disjunctive Logic Programming*. The MIT Press, Cambridge, Massachusetts, 1992.

La programación lógica disjunctiva permite la descripción de información indefinida o incompleta a través de átomos disjuntos en la cabeza de la cláusula. Los autores describen distintos tipos de semánticas y presentan la teoría de la negación. Finalmente, incluyen algunas aplicaciones en bases de datos de conocimiento.

[MPM+04] Martin, D. L., Paolucci, M., McIlraith, S. A., Burstein, M. H., McDermott, D. V., McGuinness, D. L., Parsia, B., Payne, T. R., Sabou, M., Solanki, M., Srinivasan, N., and Sycara, K. P. Bringing semantics to web services: The OWL-S approach. In *SWSWPC* (2004), J. Cardoso and A. P. Sheth, Eds., vol. 3387 of *Lecture Notes in Computer Science*, Springer, pp. 26–42.

Las tecnologías de servicios para la web semántica, tales como OWL-S, están evolucionando de forma tal de permitir que los servicios pueden especificarse utilizando semánticas más completas. Estas semánticas más completas permiten un uso y una selección automática de servicios más flexible, a la vez que soportan la construcción de herramienntas y metodologías más poderosas. Este trabajo muestra como utilizar OWL-S en conjunción con los estándares de los servicios Web, y explica e ilustra el valor agregado de las semánticas expresadas en OWL-S.

[MR01] Marek, V. W., and Remmel, J. B. On the foundations of answer set programming. In *Answer Set Programming* (2001), A. Provetti and T. C. Son, Eds.

Este trabajo aborda uno de los aspectos fundacionales de Answer Set Programming, el cuál corresponde a caracterizar qué es lo que teóricamente pueden computar este tipo de sistemas. Este trabajo se focaliza en un formalizmo de ASP específico, el trabajo de Gelfond y Lifschitz de Semánticas Estables para Programas Lógicos.

[ON98] O'Brien, P. D., and Nicol, R. C. FIPA — Towards a Standard for Software Agents. *BT Technology Journal 16* (1998), 51–59.

Los Sistemas Multi-Agente requieren de un alto grado de interactividad entre los agentes y, por lo tanto, necesitan de la existencia de consenso en las interfaces de los mismos para soportar la interoperabilidad entre los distintos sistemas de agentes. Este trabajo describe FIPA y sus organizaciones, incluyendo un resumen y una guía de la especificación FIPA97.

[Poo85] Poole, D. On the comparison of theories: Preferring the most specific explanation. In *IJCAI* (1985), pp. 144–147.

[RTK07] Reches, S., Talman, S., and Kraus, S. A statistical decision-making model for choosing among multiple alternatives. In *AAMAS* (2007), E. H. Durfee, M. Yokoo, M. N. Huhns, and O. Shehory, Eds., IFAAMAS, p. 205.

Los agentes posseen diversas alternativas para seleccionar al momento de resolver un problema. Generalmente, el agente no conoce de antemano cuál es la mejor opción y, por lo tanto, se requiere de algún tipo de exploración. Este trabajo se enfoca en los casos en los cuales los agentes poseen algún tipo de información inicial de cada posible alternativa, proponiendo un modelo estático para determinar cuál es el mejor enfoque que permite resolver el problema asumiendo que, en un principio, se cuenta con poca información.

[Smi81] Smith, R. G. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers C-29*, 12 (1981), 1104–1113.

El protocolo Contract Net fue desarrollado para especificar cómo es la comunicación entre un grupo de agentes que tienen como principal objetivo distribuir la resolución de tareas. Ésta distribución es llevada a cabo en un proceso de negociación, en el cual intervienen agentes con tareas a ser ejecutadas y agentes que

estan capacitados para resolverlas. Este trabajo presenta una especificación del protocolo y demuestra su uso en la solución de un problema de sensado distribuido.

[TG05] Tucat, M., and García, A. J. An Extended Set of Interaction Primitives for Multi-agent Systems Development. In *Actas del 7mo Workshop de Investigadores en Ciencias de la Computación (WICC 2005)* (Córdoba, Argentina, 2005), Universidad Nacional de Río Cuarto, pp. 218–222.

En este trabajo se propone la extensión de las primitivas de interacción presentadas en [GTS05]. En particular se proponen extensiones para permitir que un agente pertenezca a distintos sistemas mutli-agente simultáneamente y para permitir mayor flexibilidad a la hora de asociar la llegada de mensajes de distintos tipos a la ejecución automática de predicados Prolog.

[TG06] Tucat, M., and García, A. J. Primitives for Building Interaction Protocols. In *Actas del XII Congreso Argentino de Ciencias de la Computación (CACIC 2006)* (San Luis, Argentina, 2006), Universidad Nacional de San Luis.

[TG07] Tucat, M., and García, A. J. Design and Implementation of a FIPA based Agent Communication Model for a Logic Programming Framework. In *Actas del XIII Congreso Argentino de Ciencias de la Computación (CACIC 2007)* (Corrientes, Argentina, 2007), Universidad Nacional del Nordeste, pp. 1562–1572.

En este trabajo se diseñó un modelo de comunicación entre agentes basado en la arquitectura FIPA y en los Protocolos de Interacción. El objetivo de este modelo de comunicación es permitir la comunicación entre agentes de acuerdo a sus características e intercambiar mensajes utilizando un lenguaje de comunicación de agentes estándar. La implementación del modelo de comunicación propuesto corresponde a una extensión de Prolog, siguiendo el espíritu de la Programación Lógica.

[TGS09] Tucat, M., Garcia, A., and Simari, G. Using Defeasible Logic Programming with Contextual Queries for Developing Recommender Servers. In *AAAI Fall Symposium Series* (2009).

En este trabajo se proponen y formalizan las consultas contextuales múltiples para los Servicios de Razonamiento basados en DeLP. El dominio de aplicación de sistemas de recomendación es utilizado para motivar y ejemplificar posibles usos de los distintos tipos de consultas.

[Vou07] Vouros, G. A. Information searching and sharing in large-scale dynamic networks. In *AAMAS* (2007), E. H. Durfee, M. Yokoo, M. N. Huhns, and O. Shehory, Eds., IFAAMAS, p. 49.

Encontrar el agente adecuado para que provea, a tiempo, los recursos necesarios en una red de gran tamaño y a la vez dinámica, es un problema aún no resuelto. Este trabajo presenta una metodología para la búsqueda y el compartimiento de la información que combina índices de ruteo con métodos basados en tokens. El método propuesto permite a los agentes buscar efectivamente a través de obtener los intereses de sus vecinos, publicitar sus habilidades de provisión de información y mantener índices para retrasmitir consultas, todo integrado en el mismo proceso.

[WD07] Wu, J., and Durfee, E. H. Sequential resource allocation in multiagent systems with uncertainties. In *AAMAS* (2007), E. H. Durfee, M. Yokoo, M. N. Huhns, and O. Shehory, Eds., IFAAMAS, p. 114.

Este trabajo aborda el problema del intercambio de recursos limitados en ejecución entre un grupo de agentes, teniendo como objetivo mejorar la performance del sistema multiagente. Lograr un secuenciamiento optimal de la asignación de recursos no es trivial en sistemas complejos con incertidumbre. El trabajo presenta un algoritmo que permite separar automáticamente en múltiples fases una tarea compleja y lograr una reasignación de recursos de manera optimal al inicio de cada fase.

[Woo99] Wooldridge, M. Intelligent agents. In *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, G. Weiss, Ed. The MIT Press, Cambridge, MA, USA, 1999, ch. 1, pp. 27–78.

[ZJ07] Zheng, L., and Jin, Z. Requirements driven agent collaboration. In *AAMAS* (2007), E. H. Durfee, M. Yokoo, M. N. Huhns, and O. Shehory, Eds., IFAAMAS, p. 196.

Este trabajo propone lograr colaboración entre agentes basándose en los requerimientos. Esta propuesta asume la existencia de diversos agentes que proveen servicios y se encuentran distribuidos en Internet. Cuando ocurre un requerimiento para llevar a cabo una tarea en particular, estos agentes autónomos pueden reconocer los nuevos requerimientos emergentes y formar grupos de agentes dinámicamente buscando proveer soluciones para estos requerimientos.