Universidad del Bío-Bío
Facultad de Ciencias Empresariales
Departamento de Sistemas de Información

Profesora Guía
Mónica Alejandra
Caniupán Marileo

"Peer Consistent Answers System"

TRABAJO DE TITULACIÓN PRESENTADO EN CONFORMIDAD A LOS REQUISITOS
PARA OBTENER EL TÍTULO DE INGENIERO CIVIL EN INFORMÁTICA

Marzo 12 de 2009

Ariela Alejandra Carrera Clavería

Resumen

La programación lógica ha contribuído al desarrollo de las bases de datos deductivas, sistemas cuya entrada es la instancia de la base de datos, y su principal salida es, por medio de reglas lógicas, inferir nuevo conocimiento, sin necesidad de almacenarlo.

Las bases de datos relacionales no siempre son consistentes con respecto a sus restricciones de integridad y los sistemas administradores de bases de datos no siempre pueden mantener la consistencia. Para instancias de bases de datos inconsistentes, es posible obtener reparaciones, por medio de la computación de programas en lógica disyuntiva, llamados programas de reparación. Basados en la semántica de los modelos estables, los programas de reparación tienen como entrada la instancia de la base de datos y sus restricciones de integridad. Para obtener respuestas consistentes, se computan los programas de reparación en conjunto con el programa de consulta. Es posible obtener las respuestas consistentes a las consultas desde los modelos estables del nuevo programa. Para obtener tales modelos, DLV System [16] es de suma utilidad, ya que despliega modelos estables a partir de un programa DATALOG y permite conectarse con bases de datos.[1]

Cuando se consulta por datos que dependen no sólo de una base de datos, sino que de muchas relacionadas entre sí, la obtención de respuestas consistentes a una consulta se torna más compleja. Es el caso de los sistemas de intercambio de datos punto a punto, también llamados sistemas P2P. Un sistema P2P es un conjunto de bases de datos relacionales (nodos), que intercambian datos de manera descentralizada. En un sistema P2P, la consistencia no sólo obedece a las restricciones de integridad locales de cada nodo, sino que además está condicionada por restricciones de intercambio de datos y relaciones de confianza, entre pares de nodos. En un sistema P2P, un nodo intercambia datos con otros nodos, por medio de consultas y respuestas consistentes de nodos, solamente en el caso que éste confíe en los datos de los otros tanto o más que en los propios.

[1] DLV es un software libre, cuyo nombre significa DATALOG with disjunction.

Bajo las condiciones expuestas, surge un nuevo problema, que si bien teóricamente tiene solución, no ha sido implementado hasta hoy: la obtención de respuestas consistentes para una consulta hecha a un nodo que participa de un sistema P2P.

La semántica de las respuestas consistentes a consultas para un nodo, en un sistema P2P, está dada en términos de todas las reparaciones de la base de datos local, las cuales son mínimas, virtuales y satisfacen las restricciones de intercambio de datos con otros nodos, y de éstos últimos con otros, y así, sucesivamente. De esta forma, el problema de obtener respuestas consistentes de nodos que son posiblemente inconsistentes, se puede solucionar bajo un enfoque recursivo, por medio del cómputo de programas de reparación y programas de consulta.

En esta Habilitación Profesional se implementan programas en lógica disyuntiva para responder consistentemente a consultas, expresadas en DATALOG, formuladas a una base de datos relacional y evaluadas sobre un sistema P2P, con respecto a restricciones de integridad locales, relaciones de confianza y restricciones de intercambio de datos. Estos programas son implementados en una aplicación computacional, que interactúa con DLV System y con nodos (bases de datos relacionales) que participan de un sistema P2P especificado.

Dedicatoria

Son los sueños los que mueven al humano.

A mi familia

A lo largo del camino hay sacrificios que si bien en un principio son amargos a la larga se transforman en poesía, energía que alimenta compensando.

Si llegaras a flaquear en un momento, no te olvides que un legado me has dejado: que el camino no es más malo que el atajo, lo barato muchas veces sale caro.

Cómo ignorar el sudor de tu frente, y la espalda adolorida de cansancio; son jornadas de lo más agotadoras, son cariño, son entrega, son trabajo.

Te regalo mi vitalidad naciente y toda la expectativa que has posado; por amor, por convicción, por el futuro, por tu esfuerzo que ya sabes, no es en vano.

Si tu cielo no se encuentra despejado, caminando por senderos alargados, compañero la intención es la que prima, son los sueños los que mueven al humano.

Herederos de una tierra prometida, aprendices de un amor interesado: no confíen al dinero lo importante, sociedad es la que integra sin dudarlo.

Seas blanco, seas negro o amarillo, de la China, de Bolivia o de Tobago, el rebelde, el sometido o el anarco, lo esencial no es entendernos, hay que amarnos. [2]

[2] *Son los sueños los que mueven al humano* por *Ariela Carrera* está licenciada bajo Licencia Creative Commons Atribución-No Comercial-Sin Derivadas 2.0 Chile. Fue escrita en el verano de 2001, inspirada por su padre y reservada especialmente para esta ocasión. Ha permanecido inédita hasta hoy.

Agradecimientos

Escogí llegar hasta acá sin dejar de cargar mis convicciones y principios, aunque muchos se empeñaron en dejar caer los propios y no pocas veces, derribar los ajenos. Conmigo no pudieron, y mientras mi cansancio aumentaba, aumentaba mi valor y deseo de surgir por mis propios medios. Como dicen los refranes, de lo bueno no hay abundancia, en épocas de oscuridad, sólo pocos dieron luz y tomaron relevancia. Por compartir la vida juntos, les doy gracias. Ellos hasta hoy, siguen brillando con luz propia, y les agradezco estar ahí, en las buenas y en las malas. Éste no fue un camino fácil y mis últimos pasos fueron tardíos. Si en un momento me cubrió el miedo o el hastío, ellos no dejaron de ser correctos, a pesar de avanzar más lento. Después de todo puedo decir que lo logré, que lo logramos, de la mejor manera en que lo pude hacer, con la frente en alto, con los ojos en el cielo, con los pies en la tierra, con una mano en la razón, y la otra, en el corazón.

Este trabajo no hubiera sido posible sin la presencia de estos seres de luz. *Mononi*: sensatez, optimismo, valentía, autoestima. *Willy*: esfuerzo, perseverancia, honradez, candidez. *Bebé*: objetividad, consecuencia, determinación, honor. *Mónica Caniupán*: gracias por ser como es, por su rigurosidad, responsabilidad, autoexigencia y exactitud. De seguro que tengo mucho que aprender de Ud. *Pedro Gerónimo Campos Soto*, muchas gracias por su buena voluntad, por la clase magistral de Servlets y de Jsp, y por todos los consejos para esta implementación. *Germán Poo*: gracias por compartir tus conocimientos conmigo, por la ayuda con DB2 y por aquella oportunidad en Sistemas Distribuidos. *Karina Rojas*: gracias por confiar en mí y por darme todas las oportunidades que me ha dado. A mis *amiguis* y *compañeros:* Vivi, Divo, Churraza, Perro, Jerssy, Overcross, Compadre, Patricia Herminia, Hiro, Nati Oyarce, Daniel, Je, Verito, Rusia, Ramiro, mAldito, Marcyta, Eimer, CheKa, Viejo Lobo de Mar, Valita, Jorge, Richy, Pilow, Rorrotech, Sole, Jimbo, Nati Contreras, Zeta, Joselito, Liz, Yose, Estepario, Lechu, Chanchi Mala, Martini, Mito y Víctor. Les agracedezco sus buenas intenciones, no saben cuánto bien me han hecho su compañía y/o sus palabras de aliento. A todos

ellos les debo una parte de mí, que se ha reflejado en este resultado. Gracias a todos ustedes y gracias a Dios. ¡Por fin, lo logramos!

Índice general

Índice de figuras

Capítulo 1

Introducción

Los sistemas de gestión de bases de datos (SGBD) poseen, por definición, una serie de reglas que expresan la semántica de los datos almacenados, que garantizan su integridad y validez, denominadas restricciones de integridad, denotadas por *IC* (Integrity Constraints). La evolución de los sistemas de intercambio de datos punto a punto ha sido extensa, en cuanto a redes y comunicaciones se refiere. Variados son los protocolos que rigen las transmisiones, hardware, software y middleware, cada vez más avanzados, hacen de la conectividad algo cotidiano y eficiente. A pesar de esto, han dejado de lado la preocupación por que los datos sigan siendo un volumen de información válida con una semántica, satisfaciendo las restricciones de integridad.

Bajo el escenario de los sistemas de intercambio de datos punto a punto, cada uno de los participantes (bases de datos conectadas), es igual a los demás, es decir, no existen clientes ni servidores predeterminados y parte de los datos de una base de datos están relacionados con los datos de otra. Los sistemas de intercambio de datos punto a punto, son más conocidos como sistemas *P2P* (Peer to Peer). Como nodos conectados para compartir datos, los sistemas P2P deben normar de alguna manera este fuerte intercambio de información. Esto lo logran mediante una serie de restricciones de carácter semántico que rigen la forma en que sus datos se interrelacionan, denominadas *restricciones de intercambio de datos*, denotadas por *DEC* (Data Exchange Constraints), y

un conjunto de *relaciones de confianza*. Un nodo sólo intercambia datos con otros en los que él confía tanto o más que en sí mismo.

Entonces, un sistema P2P se conforma, a grandes rasgos, por un conjunto $\mathcal{P}$ de n nodos, con $\mathcal{P} = \{\mathtt{P}_1, \ldots, \mathtt{P}_n\}$, cada uno con un esquema, instancia y restricciones de integridad propios. Además, se tienen el conjunto de relaciones de confianza y de DECs entre pares de nodos distintos $\Sigma(\mathtt{P}_i, \mathtt{P}_j)$, con $i \neq j$. Por cada DEC se tiene una correspondiente tripleta en la relación de $confianza \subseteq \mathcal{P} \times \{menos, igual\} \times \mathcal{P}$ [7], esto es, cada DEC tiene asociada una relación de confianza. De esta forma, los nodos en un sistema P2P tienen una doble labor: permanecer íntegros conforme a ICs locales, lo que por sí solo resulta complejo, y además, transar sus datos de acuerdo a restricciones de intercambio y relaciones de confianza con otros nodos.

Cuando se hace una consulta a un nodo `P` y los datos de éste dependen de DECs con otros nodos en $\mathcal{P}$ y de las ICs de cada uno, garantizar una respuesta consistente a la consulta se hace un problema cuya solución no se ha implementado. Para responder, el nodo `P` debería consultar a su vez, los datos de los nodos en los que él confía, involucrados en las DEC's y en la consulta. ¿Cómo entonces podría garantizarse que tal información intercambiada no transgreda todas las restricciones a las que está sometida, de manera que la instancia del nodo sea consistente, y por ende, la respuesta a la consulta sea consistente? Es así como surge la problemática de obtener respuestas consistentes a consultas hechas a un nodo que participa de un sistema P2P.

Ejemplo 1 Se tiene un sistema P2P para una universidad, y se desea consultar por todos los alumnos que registran recetas de medicamentos con el departamento de salud. Sean los nodos `A` y `B` parte de este sistema P2P, representando el registro académico de los alumnos y el departamento de salud, respectivamente.

El esquema del nodo `A` se compone de la relación *Alumno*, cuya clave primaria es *Rut*, que es la única restricción de integridad. La instancia es la siguiente:

Alumno	*Rut*	*Nombre*	*Estado*
	152	*Berta*	*Pendiente*
	121	*Pedro*	*Regular*
	132	*Mario*	*Regular*
	110	*Omar*	*Adscrito*

El nodo `B` representa el departamento de salud de la universidad, donde se atienden los alumnos. El esquema de `B` se compone de las relaciones *Atencion* y *Receta*. Las restricciones de integridad para este nodo indican que los atributos *Codigo* y *Folio* son claves primarias y que por cada tupla con atributo *Codigo* en la relación *Receta* debe existir una tupla con el mismo valor en el atributo *Codigo* de la relación *Atencion*.

Atencion	*Codigo*	*Fecha*	*Rut*
	10	$2-3-7$	121
	20	$12-4-7$	132
	30	$25-8-7$	147
	40	$28-8-7$	110

Receta	*Folio*	*Codigo*	*Medicamento*	*Costo*
	01	10	*Ambroxol*	758
	02	10	*Viadil*	2485
	03	50	*Paracetamol*	398

La relación de confianza entre estos nodos es $\{(\texttt{B}, menos, \texttt{A})\} \in confianza$. Su correspondiente restricción de intercambio de datos indica que sólo se atienden en el departamento de salud las personas que sean alumnos registrados. Esto es, por cada tupla en la relación *Atencion* debe existir una tupla en la relación *Alumno* con el mismo valor en el atributo *Rut*.

A nivel local, se pueden apreciar que la tupla $Receta(03, 50, Paracetamol, 398)$, es inconsistente, ya que no existe una correspondencia en la relación *Atencion* con $Codigo = 50$. Por otro lado, a nivel del sistema P2P, se tiene $Atencion(30, 25-8-7, 147)$ pero no existe ninguna tupla en la relación *Alummo* con $Rut = 147$. Dada una consulta

$\mathcal{Q}$: $Atencion(X, Y, Z)$, ¿qué tuplas se deben desplegar para responder satisfaciendo todas las restricciones de integridad y de intercambio? ¿Bajo qué criterio debiéramos omitir o incluir ciertas respuestas? □

Sea un sistema P2P, y un nodo B en él. Concretamente, dado un conjunto de restricciones, de integridad y de intercambio de datos, no siempre una instancia $D(\mathtt{B})$ lo satisface. Debido a esto, dada una consulta, la entrega de respuestas consistentes constituye un problema no menor. Surgen interrogantes como ¿De los datos que están en la instancia $D(\mathtt{B})$, cuáles constituyen *respuestas consistentes*? Ésta es una de las interrogantes que este proyecto intenta resolver.

El panorama que los sistemas administradores de base de datos presentan no es alentador en cuanto a garantizar que las respuestas a consultas sean consistentes. Las IC son transgredidas por muchas razones, como sistemas heredados e integración de datos. La complejidad del problema de hace mayor cuando para responder las consultas es necesario comprobar la consistencia de los datos en otros nodos. En este escenario, la consistencia se vuelve un problema con más variables que al principio. Los datos de un nodo pueden estar dependiendo de los datos de otro, por medio de las DECs y de relaciones de confianza, y éstos, de otros nodos a su vez.

La consistencia en sistemas de intercambio de datos P2P no ha pasado de la etapa conceptual. Extensos son los estudios a nivel nacional e internacional relacionados con bases de datos y programación lógica, abarcando temas como la consistencia, los modelos estables y los programas de reparación, los que dan soporte a un sólido marco teórico y lógico que describe posibles formas de solucionar el problema de la consistencia en sistemas P2P. Los vastos estudios hechos, han expuesto sólidos y organizados marcos teóricos. A pesar de ello, sí se ha avanzado bastante en el manejo de consistencias a nivel de base de datos locales. Existen herramientas que colaboran en la obtención de respuestas consistentes a nivel local, tales como ConsEx [17], Queca [22], Hippo [21] y ConQuer [23]. No obstante, no existe una aplicación que asegure que los datos intercambiados en un sistema P2P, mediante consultas y respuestas, satisfagan las DECs y las

relaciones de confianza. No se ha implementado un programa que compute respuestas consistentes a consultas, con respecto a DECs, confianza e ICs, en sistemas P2P.

Por lo expuesto, la importancia de desarrollar este proyecto radica, fundamentalmente, en implementar programas de reparación para obtener respuestas consistentes a consultas, en sistemas P2P que pueden ser inconsistentes con respecto a DECs e ICs. En virtud de esto, desarrollar una aplicación que permita la obtención de respuestas consistentes a consultas expresadas en DATALOG, formuladas a un nodo que participa de un sistema P2P, se vuelve una necesidad y un desafío, y constituye el principal objetivo de esta Habilitación Profesional. En particular, se persigue diseñar una aplicación que permita la comunicación de diferentes bases de datos por medio de consultas, mejorar la lógica de los algoritmos y la teoría existentes, generando programas de reparación y evaluándolos sobre instancias reales, obtener respuestas consistentes a consultas expresadas en DATALOG e implementar la aplicación, mostrando la correctitud de los algoritmos implementados.

Este trabajo contribuye con el diseño, la implementación y las pruebas necesarias para desarrollar una aplicación computacional, que despliega respuestas consistentes a consultas expresadas en DATALOG, a partir de datos en sistemas P2P que posiblemente no son consistentes, dando así cumplimiento a los objetivos señalados anteriormente. Este trabajo no contempla en el desarrollo de la aplicación, la realización de control de versiones, ayuda en línea ni mantención.

El presente informe se compone de cinco capítulos. Este primer capítulo fue introductorio. Aquí se expusieron los antecedentes del problema y la naturaleza del proyecto. En el capítulo 2 se analizan aspectos teóricos, como bases de datos relacionales y programas DATALOG. Se define qué es la consistencia y las reparaciones en una base de datos local. El capítulo 3 ahonda en el tema de cómo manejar la consistencia en un sistema P2P. Se expone un análisis teórico de la semántica desarrollada en [6, 7], para obtener respuestas consistentes a consultas en sistemas P2P. Se finaliza presentando el programa de reparación para obtener respuestas consistentes de nodos. El capítulo 4 explica los modelos utilizados en el desarrollo de la aplicación. Se muestra en qué con-

siste la aplicación, su funcionalidad, información que maneja y se especifica el diseño de los módulos. Se muestran los detalles de la implementación y los principales resultados obtenidos. El capítulo 5 sintetiza las ideas más importantes de este trabajo, recalcando los logros y presentando las conclusiones.

Capítulo 2

Preliminares

En este capítulo se analizan aspectos de la programación en lógica disyuntiva y del esquema relacional. Se describe en detalle la teoría de la consistencia de bases de datos, aplicada localmente. Este capítulo finaliza presentando la problemática de obtener respuestas consistentes a una consulta formulada a un nodo que participa de un sistema P2P.

2.1. Programas en Lógica Disyuntiva DATALOG

Los *programas en lógica disyuntiva* DATALOG, denotados por *DLP* (Disjunctive Logic Program), fueron introducidos por [12]. Un DLP es un conjunto finito de reglas de la siguiente forma:

$$A_1(\bar{x}_1) \vee \cdots \vee A_n(\bar{x}_n) \leftarrow B_1(\bar{y}_1), \ldots, B_k(\bar{y}_k), not\ B_{k+1}(\bar{y}_{k+1}), \ldots, not\ B_m(\bar{y}_m) \qquad (2.1)$$

En cada regla r, cada A_i y B_i son nombres de predicados; cada $A_i(\bar{x}_i)$ y $B_i(\bar{y}_i)$ son átomos en un lenguaje relacional de primer orden; cada $\bar{x}_i$ y $\bar{y}_i$ son listas de variables y constantes, que coinciden con la aridad de sus respectivos predicados; un literal es un átomo, es decir, $P(\bar{c})$, o bien, su negación $not\ P(\bar{c})$.[1] Para un literal L, $not\ L$ representa el literal complementario de L. La disyunción se represen-

[1]La aridad es la cantidad de atributos que posee una relación; por ejemplo, $Alumno(Rut, Nombre)$ tiene aridad 2.

ta con el símbolo "$\vee$", mientras que el símbolo "," representa la conjunción. La negación débil es representada por el prefijo *not*. Este tipo de negación, también conocida como negación por defecto, asume como falso todo átomo que no sea inferido por el programa, conforme a la suposición del mundo cerrado, propuesta por [20]. La cabeza de la regla es denotada por $H(r)$, y está formada por la disyunción $A_1(\bar{x}_1) \vee \cdots \vee A_n(\bar{x}_n)$. El cuerpo de la regla es denotado por $B(r)$, y conformado por $B_1(\bar{y}_1), \ldots, B_k(\bar{y}_k), not\ B_{k+1}(\bar{y}_{k+1}), \ldots, not\ B_m(\bar{y}_m)$. Aquí se identifica una parte positiva, denotada por $B^+(r)$, conformada por $B_1(\bar{y}_1), \ldots, B_k(\bar{y}_k)$ y una parte negativa, denotada por $B^-(r)$, conformada por $not\ B_{k+1}(\bar{y}_{k+1})$, $\ldots$, $not\ B_m(\bar{y}_m)$. También pueden existir en el cuerpo átomos built-in, que son átomos que incluyen algunos de los predicados de comparación $\{<, >, =, \neq\ \}$, como por ejemplo, $Edad > 0$.

En particular, si $m = k = 0$, entonces la regla es llamada *hecho* y se puede omitir el símbolo "$\leftarrow$". Si $n = 0$, entonces la regla es una *restricción de negación del programa*, denotada por *PDC* (Program Denial Constraint). Si para cada regla se cumple que $m = 0$, entonces el programa es denominado *positivo*. Si para cada regla se cumple que $n = 1$, entonces el programa es llamado *normal*. La *Base de Datos Extensional*, denotada por *EDB* (Extensional Database), la componen los predicados definidos por los hechos del programa, mientras que los definidos a partir de las reglas restantes del programa, constituyen la *Base de Datos Intensional*, denotada por *IDB* (Intensional Database).

Ejemplo 2 Sea el siguiente programa en lógica disyuntiva:

$Aprueba(X, Y) \vee Reprueba(X, Y) \leftarrow Estudia(X, Y).$ (r_1)

$Estudia(1, gloria).$ (r_2)

La regla (r_2) es un hecho. *Estudia* es un predicado extensional. *Aprueba* y *Reprueba* son predicados intensionales. □

Sea Π un programa en lógica. El *universo de Herbrand* [8], denotado por $\mathcal{U}_\Pi$, está formado por todas las constantes del programa Π. La *base de Herbrand*, denotada por $\mathcal{B}_\Pi$, es el conjunto de todos los átomos que se pueden generar con los predicados en

Π y las constantes en $\mathcal{U}_\Pi$, esto es, el conjunto de todos los átomos instanciados. La *instanciación de Herbrand* de Π corresponde a todas las instancias de las reglas de Π, en las que se reemplazan las variables por las constantes en $\mathcal{U}_\Pi$; tal *programa instanciado* o *programa ground*, es denotado por $Ground(\Pi)$.[2]

Sea una *interpretación* de Π, denotada por I, un subconjunto de $\mathcal{B}_\Pi$. Una regla en Π, de la forma (2.1), se satisface en I, si alguno de los átomos de la cabeza de la regla $a_i(\bar{x}_i) \in I$, o si alguno de los literales positivos del cuerpo de la regla $b_j(\bar{y}_j)$ es falso en I. Un *modelo de Herbrand* para un programa Π, corresponde a alguna interpretación que satisface todas las reglas de $Ground(\Pi)$. Sea $\mathcal{M}$ un modelo de Herbrand de Π. $\mathcal{M}$ es un *modelo minimal* [9] si posee el mínimo de literales ground, es decir, no existe ningún subconjunto de $\mathcal{M}$ que sea modelo de $Ground(\Pi)$. Un modelo minimal está contenido en todos los modelos de Herbrand. Todos los literales presentes en el modelo minimal son consecuencias lógicas del programa, osea, son valores verdaderos. Un literal negativo del tipo $not\ b_j(\bar{y}_j)$ es verdadero en el programa si su correspondiente positivo $b_j(\bar{y}_j)$ no pertenece al modelo minimal. Para denotar los modelos minimales, se emplea $\mathcal{MM}$.

Ejemplo 3 Sea Π el siguiente programa Π: $G(X) \vee P(X) \leftarrow J(X),\ not\ T(X).\ J(e)$. La base de Herbrand es $\mathcal{B}_\Pi = \{\ J(e),\ P(e),\ G(e),\ T(e)\}$. Uno de los modelos de Herbrand es $\{\ J(e),\ G(e)\}$ □

La semántica de *modelos estables* fue introducida por [13] y es utilizada en los programas en lógica disyuntiva [14, 19]. Esta semántica señala que los DLP pueden tener más de un modelo minimal y que todo modelo estable es un modelo minimal. Los modelos estables son libres de negación [14, 19] y proporcionan literales que son o verdaderos o falsos, pero nunca desconocidos [11], debido a que los modelos estables obedecen a la suposición del mundo cerrado [20], es decir, todo literal que no pertenece al modelo es considerado como falso. El conjunto de todos los modelos estables de un programa Π, denotado por SM_Π (Stable Models).

[2] También recibe el nombre de *programa proposicional*

Para que una interpretación I constituya un modelo estable, I debe ser un modelo minimal de la *reducción Gelfond Lifschitz* de $Ground(\Pi)_I$, llamada *reducción GL* [14], que es un programa instanciado, con respecto a I, que no tiene negación, y que se obtiene de la siguiente manera: (1) eliminando todas las reglas de $Ground(\Pi)^I$ que tengan literales instanciados de la forma *not* $p(\bar{c})$, si $p(\bar{c}) \in I$ y (2) de las reglas restantes de $Ground(\Pi)^I$, eliminando todos los literales instanciados, de la forma *not* $p(\bar{c})$.

Ejemplo 4 (Ejemplo 3 cont.) Sea una interpretación I para el programa Π. $I = \{G(e), J(e)\}$. La reducción GL resultante del programa Π con respecto a I es $\{G(e) \vee P(e) \leftarrow J(e)\}$. I justamente corresponde a un modelo minimal del programa y además, es un modelo estable. El programa tiene dos modelos estables: $\mathcal{M}_1 = \{J(e), P(e)\}$ y $\mathcal{M}_2 = \{G(e), P(e)\}$ □

Si los DLP pueden tener más de un modelo estable, hay que determinar qué literales son verdaderos. Para decidir esto, la semántica de modelos estables proporciona dos razonamientos, el *cauteloso* y el *riesgoso*. El primero indica que un literal $p(\bar{c})$ es verdadero si es verdadero en todos los modelos estables de Π. En cambio, el razonamiento riesgoso considera como verdadero a $p(\bar{c})$ si éste es verdadero en al menos un modelo estable del programa.

Ejemplo 5 (Ejemplo 4 cont.) Se puede concluir que $J(e)$ es el único literal verdadero bajo el razonamiento cauteloso, dado que es el único literal presente en todos los modelos estables. Bajo el razonamiento arriesgado, se tienen como verdaderos a $J(e)$, $P(e)$ y a $G(e)$. □

2.2. El Esquema Relacional

Los *sistemas de intercambio de datos punto a punto*, denotados por *Sistemas P2P* (Peer to Peer), constituyen un conjunto de sistemas interconectados que intercambian datos de manera descentralizada. Cada uno de los sistemas componentes es denominado

nodo. Un nodo, para efectos de este trabajo, constituye una *base de datos relacional*, con un esquema, instancia y restricciones propias, que condicionan la consistencia de la base de datos, a un nivel local. Todos estos conceptos se definen en esta sección.

Sea Σ el *esquema* de una base de datos relacional [6, 11], denotada por: $\Sigma = (\mathcal{U}, \mathcal{R} \cup \mathcal{B})$. Σ está compuesto por un dominio $\mathcal{U}$, un conjunto fijo $\mathcal{R}$ de predicados o relaciones, y por $\mathcal{B}$, que es un conjunto fijo de predicados built-in, no sujetos a cambio, como los signos de comparación $\{<, >, =, \neq\}$ y el átomo proposicional **false**, el que siempre es falso en la base de datos. Cada predicado P tiene un conjunto de atributos finito y ordenado, denotado por $\mathcal{A}_P$. Se denota por $P[i]$ al iésimo atributo de P. La cantidad de atributos de un predicado se denomina *aridad* del predicado. El esquema determina el lenguaje de lógica de predicados de primer orden $\mathcal{L}(\Sigma)$. Las instancias de la base de datos del esquema relacional Σ, o tuplas, denotadas por D, son conjuntos finitos de átomos instanciados, esto es, predicados $P \in \mathcal{R}$ que contienen constantes $c_i \in \mathcal{U}$ y tienen la forma $P(c_1, c_2, \ldots, c_n)$. Una consulta es una fórmula, escrita en $\mathcal{L}(\Sigma)$, que se aplica sobre una instancia de la base de datos. La respuesta a la consulta serán todas aquellas tuplas o atributos de la instancia que satisfacen la consulta. Si la consulta sólo contiene constantes, entonces la respuesta a la consulta es booleana, osea verdadero o falso. Además de las instancias y el esquema, una base de datos está conformada por un conjunto finito de restricciones de integridad, denotado por IC (Integrity Constraint). Las restricciones de integridad son sentencias escritas en $\mathcal{L}(\Sigma)$. Ellas capturan la semántica del dominio de la aplicación y ayudan a que la información almacenada tenga un significado válido. Se espera que las restricciones de integridad sean satisfechas por cualquier instancia de Σ, pero no siempre lo son. Si una instancia D satisface a IC, se denotará por $D \models IC$.

Formalmente, una *restricción de integridad* [6], en su forma más general, es una sentencia $\psi \in \mathcal{L}(\Sigma)$ definida por:

$$\forall \bar{x}(\bigwedge_{i=1}^{m} P_i(\bar{x}_i) \rightarrow \exists \bar{z}(\bigvee_{j=1}^{n} Q_j(\bar{y}_j, \bar{z}_j) \vee \varphi)), \tag{2.2}$$

donde P_i y $Q_j \in \mathcal{R}$, $\bar{x} = \bigcup_{i=1}^{m} \bar{x}_i$, $\bar{z} = \bigcup_{j=1}^{n} \bar{z}_j$, $\bar{y}_j \subseteq \bar{x}$, $\bar{x} \cap \bar{z} = \emptyset$, $\bar{z}_i \cap \bar{z}_j = \emptyset$, con $i \neq j$ y

$m \geq 1$, tales que $\bar{z}_j$ no tiene variables repetidas para $j = 1, \ldots, n$. φ es una disyunción de átomos built-in de $\mathcal{B}$. Para una regla del tipo (2.2), el antecedente es el cuerpo de la fórmula (lado izquierdo), que precede al símbolo "$\rightarrow$". El consecuente es la cabeza de la fórmula (lado derecho del símbolo "$\rightarrow$"). Se denota por $Ant(\psi)$ al antecedente de ψ, mientras que su consecuente es denotado por $Con(\psi)$.

Las restricciones de integridad universales [6], denotadas por *UIC* (Universal Integrity Constraint), son definidas como:

$$\forall \bar{x}(\bigwedge_{i=1}^{m} P_i(\bar{x}_i) \rightarrow \bigvee_{j=1}^{n} Q_j(\bar{y}_j) \vee \varphi) \tag{2.3}$$

que es una fórmula de la forma (2.2) con $\bar{z} = \emptyset$, donde no existen variables cuantificadas existencialmente. Las restricciones de integridad referenciales [6], denotadas por *RIC* (Referential Integrity Constraint), son definidas en (2.2), con $n = m = 1$, $\bar{y} \subseteq \bar{x}$ y $\varphi = \emptyset$:

$$\forall \bar{x}(P(\bar{x}) \rightarrow \exists \bar{z} Q(\bar{y}, \bar{z})) \tag{2.4}$$

Las restricciones de integridad no null, son denotadas por *NNC* (Not Null Constraint) [6]. Este tipo de restricción permite prevenir que ciertos atributos tomen el valor **null** y tienen la forma:

$$\forall \bar{x}(P(\bar{x}) \wedge IsNull(x_i) \rightarrow \textbf{false}) \tag{2.5}$$

donde es introducido el predicado $IsNull$ por [5], con $IsNull(c)$ verdadero ssi c tiene el valor **null**, y $x_i \in \bar{x}$ está en la posición del atributo que no puede tomar valores nulos.

Definición 1 Sea IC el conjunto de restricciones de integridad de una base de datos. Su *grafo de dependencias* [10, 11] se denota por $\mathcal{G}(IC)$ y está compuesto por: (a) Un conjunto de vértices, que representan los predicados $P \in IC$. (b) El conjunto de los vértices de un grafo $\mathcal{G}$ es denotado por $\mathcal{V}(\mathcal{G})$. (c) Sean P_i y P_j dos predicados en ic, con $ic \in IC$. Los arcos desde P_i hasta P_j representan que $P_i \in Ant(ic)$ y $P_j \in Con(ic)$. (d) Un vértice es llamado *fuente* si solamente tiene arcos salientes. (e) Un vértice es llamado *destino*, si sólo tiene arcos entrantes. □

Si un vértice tiene arcos salientes y entrantes, se denomina *intermedio*. Si $P_i \in Ant(ic)$ y solamente se tiene $\varphi \in Con(ic)$, con φ como fórmula built-in, entonces el arco sale de P_i y llega hasta P_i, y tal vértice se denomina *destino* [10]. Un *componente*

conectado [11] en un grafo, es un subgrafo maximal, tal que para cada par (A, B) de vértices, existe un camino desde A hasta B o viceversa. El conjunto de componentes conectados para un grafo $\mathcal{G}$ es denotado por $\mathcal{C}(\mathcal{G})$. Dado un conjunto IC de UICs y RICs, IC_U denota al conjunto de UICs en IC. El *grafo de dependencias contraído* [10] de IC, denotado por $\mathcal{G}^C(IC)$ se obtiene de $\mathcal{G}(IC)$, reemplazando los vértices en $\mathcal{V}(\mathcal{G})$ por un único vértice, y borrando todos los arcos asociados a los elementos de IC_U, para cada componente conectado $c \in \mathcal{C}(\mathcal{G}(IC_U))$. Se dice que un conjunto IC *es acíclico RIC* [10] si su grafo de dependencias contraído no tiene ciclos. Esto siempre se cumple si $IC = IC_U$, es decir, si sólo se tienen restricciones del tipo UIC. Si por el contrario, existen ciclos en el grafo que involucren a restricciones del tipo RIC, se dice que IC es *es cíclico RIC*, al igual que cuando existe un auto-ciclo.

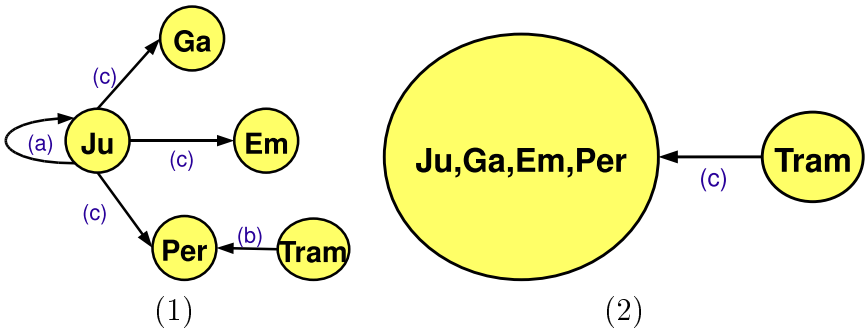

Figura 2.1: Grafos de Dependencia IC_1 Ejemplo 6

Ejemplo 6 Sea IC_1 el conjunto compuesto por las siguientes restricciones de integridad:

NNC $\forall xy(Jugar(x, y) \wedge IsNull(x) \rightarrow false)$. (a)

RIC $\forall xy(Tramposa(x) \rightarrow \exists y(Perder(x, y)))$. (b)

UIC $\forall xy(\ Jugar(x, y) \rightarrow Ganar(x, y) \vee Perder(x, y) \vee\ Empatar(x, y))$. (c)

La Figura 2.1(1) muestra el grafo de dependendencias de IC_1, mientras que la Figura 2.1(2) muestra su grafo contraído. Cada predicado ha sido representado por su primera sílaba. Vértice origen: $Tram$. Vértices destino: Ga y Em. Vértices intermedios: Ju y Per. Componentes conectados: $\mathcal{C}(\mathcal{G}) = \{Ju, Per, Em, Ga, Tram\}$, osea todos. Este conjunto es *acíclico RIC*. Se considera un nuevo conjunto de restricciones de integridad,

IC_2, al añadir al conjunto IC_1 nuevos predicados y tres restricciones:

RIC $\forall x(Apostador(x) \rightarrow \exists y(Cuota(x, y))$. (f)

RIC $\forall xy(Cuota(x, y) \rightarrow \exists z(Premio(x, z)))$. (g)

UIC $\forall x(Premio(x, z) \rightarrow Apostador(x))$. (h)

Las Figuras 2.2(1) y 2.2(2) muestran los grafos $\mathcal{G}(IC_2)$ y $\mathcal{G}^C(IC_2)$, respectivamente. Notar que el subgrafo construído a partir de (f), (g) y (h) es cíclico RIC. Componentes conectados: $\mathcal{C}_a(\mathcal{G}) = \{Ju, Per, Em, Ga, Tram\}$, y $\mathcal{C}_b(\mathcal{G}) = \{A, Cuo \text{ y } Pre\}$. □

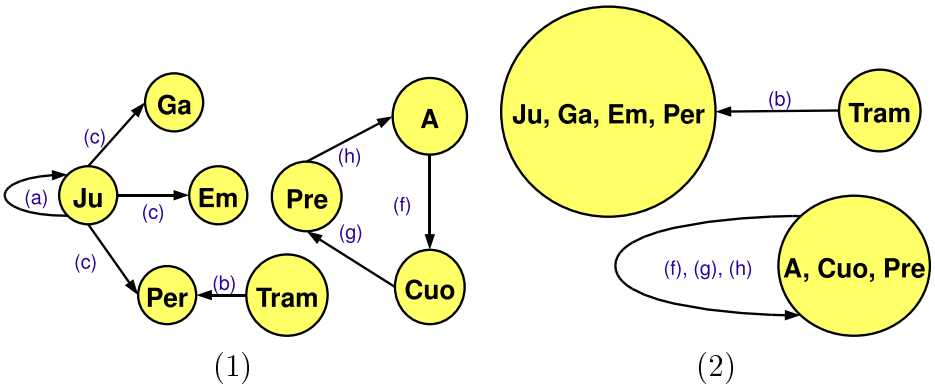

Figura 2.2: Grafos de Dependencias IC_2 Ejemplo 6

2.3. Consistencia y Reparaciones en Bases de Datos

Una instancia D es *consistente* si satisface su conjunto de restricciones de integridad dadas IC. De otra forma, es *inconsistente*. En esta sección se define la semántica de satisfacción de restricciones de integridad en términos de considerar cuáles son los atributos relevantes [6] y cuáles tuplas proveen más información que otras [18]. Se definen los conceptos de distancia [1], reparación y se describe el uso de valores nulos para reparar introducido por [5].

Definición 2 [6] Sea ψ una IC de la forma (2.2). Sea t un término, es decir, una variable o una constante del dominio. Sea $pos^R(\psi, t)$ el conjunto de posiciones en el predicado $R \in \mathcal{R}$ donde el término t aparece en ψ. El conjunto de *variables relevantes* para ψ es

$\mathcal{V}(\psi) = \{x \mid x$ es una variable repetida en $\psi\}$. El conjunto de *atributos relevantes* para ψ es denotado por $\mathcal{A}(\psi)$, y está formado por aquellos atributos R[i] tales que:

$\mathcal{A}(\psi) = \{R[i] \mid x \in \mathcal{V}(\psi)$, con $i \in pos^R(\psi, x)\} \cup$

$\{R[i] \mid c$ es una constante en ψ, con $i \in pos^R(\psi, c)\}$

Una IC ψ, de la forma (2.7) es *satisfecha* [5] en la instancia de la base de datos D, denotada por $D \models_N \psi$ ssi $D^{\mathcal{A}} \models \psi^N$, donde ψ^N es:

$$\forall \bar{x}(\bigwedge_{i=1}^{m} P_i^{\mathcal{A}(\psi)}(\bar{x}_i) \rightarrow (\bigvee_{v_j \in A(\psi) \cap \bar{x}} IsNull(v_j) \vee \exists \bar{z}(\bigvee_{j=1}^{n} Q_j^{\mathcal{A}(\psi)}(\bar{y}_j, \bar{z}_j) \vee \varphi))) \qquad (2.6)$$

y $\bar{x} = \bigcup_{i=1}^{m} \bar{x}_i$ y $\bar{z} = \bigcup_{j=1}^{n} \bar{z}_j$. Aquí, $D^{\mathcal{A}(\psi)} \models \psi^N$ se refiere a la satisfacción de primer orden, donde el valor *null* es tratado como cualquier constante del dominio $\mathcal{U}$. □

De esta fórmula, se pueden observar dos casos en los que una restricción es satisfecha: (a) Si *null* está en cualquiera de los atributos relevantes del antecedente. (b) Si ningún valor nulo aparece en los atributos relevantes, se debe cumplir la segunda disyunción de la fórmula (2.6), donde el consecuente está restringido a los atributos relevantes.

Para un conjunto de atributos $\mathcal{A}$ y un predicado $P \in \mathcal{R}$, se denota por $P^{\mathcal{A}}$ al predicado restringido o proyectado en los atributos de $\mathcal{A}$. $D^{\mathcal{A}}$ denota a la base de datos D con todos sus átomos proyectados en los atributos de $\mathcal{A}$, es decir, $D^{\mathcal{A}} = \{P^{\mathcal{A}}(\Pi_{\mathcal{A}}(\bar{t})) \mid P(\bar{t}) \in D\}$, donde $\Pi_{\mathcal{A}}(\bar{t})$ es la proyección de la tupla $\bar{t}$ en $\mathcal{A}$. $D^{\mathcal{A}}$ tiene el mismo dominio subyaciente $\mathcal{U}$ que D.

Ejemplo 7 Considere una UIC ψ : $\forall xyz(Medalla(x, y, z) \rightarrow Jugar(x, y))$ y D:

Medalla	x	y	z
	ana	*brisca*	2
	hugo	*emboque*	1
	eva	*null*	5

Jugar	x	y
	ana	*brisca*
	eva	*pool*

Desde que x e y aparecen dos veces en ψ, $\mathcal{A}(\psi) = \{Medalla[1], Jugar[1], Medalla[2], Jugar[2]\}$. El valor z no es relevante para comprobar la satisfacción de ψ, debido a que sólo se deben chequear que los primeros dos atributos en *Medalla* aparezcan también en *Jugar*. Comprobar la satisfacción de esta restricción equivale a chequear si la regla

$\forall xy(Medalla^{\mathcal{A}(\psi)}(x,y) \rightarrow Jugar^{\mathcal{A}(\psi)}(x,y))$ es satisfecha por $D^{\mathcal{A}(\psi)}$. La instancia $D^{\mathcal{A}(\psi)}$ es:

$Medalla^{\mathcal{A}(\psi)}$	x	y
	ana	*brisca*
	hugo	*emboque*
	eva	*null*

$Jugar^{\mathcal{A}(\psi)}$	x	y
	ana	*brisca*
	eva	*pool*

$D \models_N \forall xyz(Medalla(x,y,z) \rightarrow Jugar(x,y)).$ (ψ)

$D^{\mathcal{A}(\psi)} \models \forall xy(Medalla^{\mathcal{A}(\psi)}(x,y) \rightarrow (x = null \vee y = null \vee Jugar^{\mathcal{A}(\psi)}(x,y)))$. (a)

Para $x = hugo$ e $y = emboque$, dado que ninguno de ellos es *null*, se necesita comprobar si $D^{\mathcal{A}(\psi)} \models Jugar^{\mathcal{A}(\psi)}(hugo, emboque)$ y resulta falso, la restricción no se satisface. Para $x = ana$ e $y = brisca$, dado que ninguno es *null*, es necesario comprobar si $D^{\mathcal{A}(\psi)} \models Jugar^{\mathcal{A}(\psi)}(ana, brisca)$; como esto es verdadero, la restricción se satisface para $x = ana$ e $y = brisca$. Para $x = eva$ e $y = null$, como $y = null$ no es necesario comprobar que $D^{\mathcal{A}(\psi)} \models Medalla^{\mathcal{A}(\psi)}(eva, null)$. La restricción se satisface para $x = eva$ e $y = null$. En consecuencia, como no todas las tuplas satisfacen la regla, la restricción no es satisfecha por esta instancia. □

Cuando surge la inconsistencia en una instancia D, la consistencia puede ser restaurada ya sea eliminando o insertando tuplas. En este sentido, una *reparación* es una nueva instancia de la base de datos, con el mismo esquema, que satisface todas las restricciones de integridad y que dista mínimamente de la instancia original. Si se borra o se elimina más de lo necesario, entonces no es una reparación. La *distancia* [1] entre la instancia original D y su reparación D', está dada en términos de la diferencia simétrica denotada por $\triangle$; donde $\triangle(D, D') = (D - D') \cup (D' - D)$.

La noción de *proveer mayor o menor información* es introducida por [18]. (a) Una constante c provee menor o igual información que una constante d, denotado por $c \sqsubseteq d$ si y sólo si $c = d$ o c es nulo. (b) Una tupla $\overline{t}_1 = (c_1, c_2, \ldots, c_n)$ provee menor o igual información que otra tupla $\overline{t}_2 = (d_1, d_2, \ldots, d_n)$, denotado por $\overline{t}_1 \sqsubseteq \overline{t}_2$ si para cada $i = 1, 2, \ldots, n$ se cumple que $c_i \sqsubseteq d_i$. Finalmente, $\overline{t}_1 \sqsubset \overline{t}_2$ significa que $\overline{t}_1 \sqsubseteq \overline{t}_2$ y $\overline{t}_1 \neq \overline{t}_2$. (d) Las dependencias funcionales llamadas de inclusión, tanto completas

como no completas, descritas anteriormente, son representadas como UICs y RICs, respectivamente, y son satisfechas si para cada tupla $\overline{t}_1 \in P$ existe otra tupla $\overline{t}_2 \in Q$, tales que $\overline{t}_1 \sqsubseteq \overline{t}_2$.

Sean D, D' y D'' instancias de una base de datos con un esquema y un dominio $\mathcal{U}$ comunes. Se tiene que $D' \leq_D D''$ si y sólo si para cada átomo de la base de datos $P(\bar{a}) \in \triangle(D, D')$ existe $P(\bar{a}')$ tal que: (a)$P(\bar{a}') \in \triangle(D, D'')$; (b) $P(\bar{a}) \sqsubseteq P(\bar{a}')$; (c) si $P(\bar{a}) \sqsubset P(\bar{a}')$, entonces $P(\bar{a}') \notin \triangle(D, D')$. Finalmente, $D'' <_D D'$ significa que $D'' \leq_D D'$ pero no que $D' \leq_D D''$ [6].

Dada una instancia D y un conjunto de restricciones de integridad IC de la forma (2.2) y restricciones del tipo NNC, una *reparación* [6] de D con respecto a IC es una instancia de la base de datos D con el mismo esquema, tal que $D' \models_N IC$, y que la diferencia $D' \leq_D D$ es minimal entre todas las instancias de base de datos que satisfacen IC con respecto a $\models_N$ y que comparten el mismo esquema.[3] Es decir, que no exista otra instancia D'' tal que $D'' \leq_D D'$, donde $D'' <_D D'$ significa que $D'' \leq_D D'$ pero no que $D' \leq_D D''$. El conjunto de reparaciones de D con respecto a sus IC se denota por $Rep(D, IC)$.

Ejemplo 8 (Ejemplo 7 cont.) Esta instancia es inconsistente con respecto a ψ, dado que se tiene las tupla $Medalla(hugo, emboque, 1)$ pero falta la tupla correspondiente $Jugar(hugo, emboque)$.[4] Intuititvamente, se generan dos reparaciones:

$$\begin{aligned} Rep_1 &= \{M(ana, brisca, 2), M(hugo, emboque, 1), M(eva, null, 5), J(ana, brisca), \\ &\quad J(eva, pool), J(hugo, emboque)\} \\ Rep_2 &= \{M(ana, brisca, 2), M(eva, null, 5), J(ana, brisca), J(eva, pool)\} \end{aligned}$$

En Rep_1 se mantuvo la tupla $M(hugo, emboque, 1)$ y se insertó la tupla $J(hugo, emboque)$. En Rep_2 se eliminó la tupla $M(hugo, emboque, 1)$. El conjunto $\{M(eva, null, 5), J(ana, brisca)\}$ no es una reparación, ya que se eliminaron más tuplas de las necesarias. □

[3] La definición formal de *reparación*, dada por [1] no incorpora los valores nulos como forma de reparar restricciones, sin embargo en [6] se define la reparación incorporándolos.

[4] Los predicados han sido renombrados por sus iniciales.

Se ha demostrado que las reparaciones de una base de datos inconsistente pueden ser especificadas como los modelos estables de programas de reparación en lógica disyuntiva, tanto para obtener respuestas consistentes a nivel local [11], como para sistemas de intercambio de datos P2P [6]. Estos programas se exponen más adelante. Para implementar estos programas y obtener modelos estables, se han desarrollado sistemas como *DLV* [16], que utiliza esta semántica en programas DATALOG, y que será utilizado en la etapa de implementación de este proyecto, descrita en el capítulo 4.

2.4. Problemática del Intercambio de Datos

Definición 3 Un *sistema de intercambio de datos punto a punto* [4, 6, 7] se define por $\beta =< \mathcal{P}$, IC , Σ, $confianza >$, donde: (a) $\mathcal{P}$ es un conjunto finito de n nodos, o bases de datos relacionales $\mathtt{P}_1, \mathtt{P}_2, \ldots \mathtt{P}_n$. Para cada nodo $\mathtt{P}$ tenemos: (i) Un esquema de datos $\mathcal{R}(\mathtt{P})$, que incluye un dominio $\mathcal{U}(\mathtt{P})$, y relaciones $R_i(\mathtt{P})$.

(ii) Una instancia del esquema de la base de datos $\mathcal{R}(\mathtt{P})$ denotada por $D(\mathtt{P})$. (iii) Un conjunto de sentencias escritas en $\mathcal{L}(\mathtt{P})$, denominadas $IC(\mathtt{P})$ que son las restricciones de integridad para $\mathcal{R}(\mathtt{P})$. (b) $IC = \bigcup_{\mathtt{P}\in\mathcal{P}} IC(\mathtt{P})$ representa todas las restricciones de integridad de β. (c) Una colección $\Sigma(\mathtt{P})$ de conjuntos de restricciones de intercambio de datos DECs, $\Sigma(\mathtt{P}, \mathtt{Q})$ para cada nodo $\mathtt{P}$, consistente en sentencias escritas en el lenguaje de primer orden de $\mathcal{R}(\mathtt{P}) \cup \mathcal{R}(\mathtt{Q})$, donde los $\mathtt{Q}$ son algunos de los otros nodos en $\mathcal{P}$. $\Sigma = \bigcup_{\mathtt{P}\in\mathcal{P}} \Sigma(\mathtt{P})$ representa todas las restricciones de intercambio de datos en β. (d) Una relación de $confianza \subseteq \mathcal{P} \times \{menos, igual\} \times \mathcal{P}$, con la intención de significar que cuando $(\mathtt{A}, menos, \mathtt{B}) \in confianza$, el nodo $\mathtt{A}$ confía en sí mismo menos que en $\mathtt{B}$; mientras que $(\mathtt{A}, igual, \mathtt{B}) \in confianza$ indica que $\mathtt{A}$ confía en sí mismo de igual manera que en $\mathtt{B}$. Por defecto, un nodo confía en sus propios datos más que en los datos de los otros nodos. Existe una correspondencia de uno a uno entre los elementos de $confianza$ y cada DEC no vacía. (e) Dado un conjunto de nodos $\mathcal{P}$, denotamos con $\overline{\mathcal{R}}(\mathcal{P})$ al esquema compuesto por la unión de todos los esquemas $\mathcal{R}(\mathtt{P})$, para todos los nodos $\mathtt{P} \in \mathcal{P}$. (f) Usamos $\overline{D}(\mathcal{P})$ para denotar la instancia del esquema $\overline{\mathcal{R}}(\mathcal{P})$. (g) Si D es una instancia de algún esquema

$\mathcal{S}$, y $\mathcal{S}'$ es un subesquema de $\mathcal{S}$, entonces $D|\mathcal{S}'$ denota que D está restringido por $\mathcal{S}'$. En particular, si $\mathcal{R}(\mathtt{P}) \subseteq \mathcal{S}$, entonces, $D|\mathtt{P}$ denota que D está condicionado por $\mathcal{R}(\mathtt{P})$. (h) Denotamos por $\mathcal{R}(\mathtt{P})^{menos}$ a la unión de todos los esquemas $\mathcal{R}(\mathtt{Q})$, con $(\mathtt{P}, menos, \mathtt{Q}) \in confianza$, es decir, la unión de todos los esquemas de los nodos en los que P confía más. Análogamente, se define $\mathcal{R}(\mathtt{P})^{igual}$. □

Una restricción de intercambio de datos o DEC es una sentencia de primer orden que determina la semántica entre dos nodos $\mathtt{P}_i$, $\mathtt{P}_j$, en un sistema P2P. Una DEC se denota por $\Sigma(\mathtt{P}_i, \mathtt{P}_j)$, con $i \neq j$. Se debe destacar que por cada DEC existe su correspondiente relación de confianza; además, las DECs $\Sigma(\mathtt{P}_i, \mathtt{P}_j)$ y $\Sigma(\mathtt{P}_j, \mathtt{P}_i)$ pueden ser diferentes.Existen dos tipos de DECs, las universales y las referenciales, denotadas por UDECs y RDECs, respectivamente.

Una *restricción universal de intercambio de datos* [6], denotada por *UDEC* (Universal Data Exchange Constraint), en $\Sigma(\mathtt{P1}, \mathtt{P2})$ es una DEC de la forma:

$$\forall \overline{x}(\bigwedge_{i=1}^{n} R_i(\overline{x}_i) \rightarrow (\bigvee_{j=1}^{m} Q_j(\overline{y}_j) \vee \varphi)). \tag{2.7}$$

donde para $R = \{R_i \mid i \in \{1, \ldots, n\}\}$ y $Q = \{Q_j \mid j \in \{1, \ldots, m\}\}, R \cup Q \subseteq \overline{\mathcal{R}}(\{\mathtt{P1}, \mathtt{P2}\}), (R \cup Q) \cap \mathcal{R}(\mathtt{P1}) \neq \emptyset, (R \cup Q) \cap \mathcal{R}(\mathtt{P2}) \neq \emptyset$ y $\bar{x}_i, \bar{y}_j \subseteq \bar{x}$. φ es una fórmula que contiene sólo átomos built-in.

Una *restricción referencial de intercambio de datos* [6], denotada por *RDEC* (Referential Data Exchange Constraint) en $\Sigma(\mathtt{P1}, \mathtt{P2})$ es una DEC de la forma:

$$\forall \overline{x}(R(\overline{x}) \rightarrow \exists \overline{y}\ (Q(\overline{x}', \overline{y}))). \tag{2.8}$$

donde $R, Q \subseteq \overline{\mathcal{R}}(\{\mathtt{P1}, \mathtt{P2}\}), \{R, Q\} \cap \mathcal{R}(\mathtt{P1}) \neq \emptyset$, y $\{R, Q\} \cap \mathcal{R}(\mathtt{P2}) \neq \emptyset$ y $\bar{x}' \subseteq \bar{x}$.

Ejemplo 9 Sea β el sistema P2P, compuesto por los nodos R, S e I.

P	x	y
	1	j
	2	m
	3	e

D	x
	5
	3
	1

Nodo R

C	x	y
	1	t
	3	e

M	x	z
	3	5
	2	3

L	x
	2
	3

Nodo S Nodo I

$confianza = \{(\mathtt{S}, igual, \mathtt{R}), (\mathtt{I}, menos, \mathtt{R}), (\mathtt{S}, menos, \mathtt{I})\}$
$\Sigma(\mathtt{S}, \mathtt{R}) = \{\forall xyw(C(x, y), P(x, w) \rightarrow y = w)\}$
$\Sigma(\mathtt{S}, \mathtt{I}) = \{\forall xz(M(x, z) \rightarrow L(x))\}$
$\Sigma(\mathtt{I}, \mathtt{R}) = \{\forall x(L(x) \rightarrow \exists y(P(x, y)))\}$ □

Definición 4 El *grafo de accesibilidad* [6], denotado por $\mathcal{G}_A(\beta)$, para un sistema de intercambio de datos P2P $\beta = \langle \mathcal{P}, \Sigma, IC, confianza \rangle$ se define como sigue: cada nodo $\mathtt{P} \in \mathcal{P}$ es un vértice, y hay un arco dirigido desde Pi hasta Pj ssi existe una DEC en $\Sigma(\mathtt{Pi}, \mathtt{Pj})$ y $(\mathtt{Pi}, menos, \mathtt{Pj}) \in confianza$ o $(\mathtt{Pi}, igual, \mathtt{Pj}) \in confianza$. Los arcos dirigidos son etiquetados con "$<$" o "$=$". Para $\mathcal{S} \subseteq \mathcal{P}, \mathcal{G}_A(\beta)[\mathcal{S}]$ es la restricción de $\mathcal{G}_A(\beta)$ a los vértices en $\mathcal{S}$. Si existe un arco dirigido desde el nodo A hasta el nodo B, etiquetado con "$<$", es porque existe una DEC $\Sigma(\mathtt{A}, \mathtt{B})$ y su correspondiente $(\mathtt{A}, menos, \mathtt{B}) \in confianza$. Análogamente se etiquetan los arcos con "$=$". □

Ejemplo 10 (Ejemplo 9 cont.) El grafo de accesibilidad para el ejemplo 9 se muestra en la Figura 2.3. □

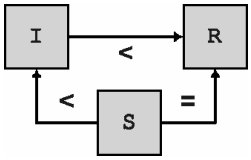

Figura 2.3: Grafo de Accesibilidad Ejemplo 10

Definición 5 [6] Un nodo P′ es *accesible* desde el nodo P si existe un camino en el grafo dirigido $\mathcal{G}_A(\beta)$ que vaya desde P hasta P′ o si $\mathtt{P}' = \mathtt{P}$. Sea $\mathcal{AC}(\mathtt{P})$ el conjunto de nodos que son accesibles desde P. Un nodo P′ es un nodo *vecino* de P si existe un arco desde P hasta P′ o si $\mathtt{P}' = \mathtt{P}$ en el grafo dirigido $\mathcal{G}_A(\beta)$. Sea $\mathcal{N}(\mathtt{P})$ el conjunto de nodos que son vecinos de P. Dado un nodo $\mathtt{P} \in \mathcal{P}$, denotamos por $\mathcal{G}_A(\mathtt{P})$ al grafo $\mathcal{G}_A(\beta)$ restringido a los nodos en $\mathcal{AC}(\mathtt{P})$. □

Ejemplo 11 (Ejemplo 10 cont.) El nodo R es vecino de sí mismo y no tiene nodos accesibles. Los vecinos de I son $\mathcal{N}(\mathtt{I}) = \{\mathtt{I}, \mathtt{R}\}$ y los nodos accesibles de I son $\mathcal{AC}(\mathtt{I}) =$

$\{\mathtt{I}, \mathtt{R}\}$. Los vecinos de $\mathtt{S}$ son $\mathcal{N}(\mathtt{S}) = \{\mathtt{S}, \mathtt{I}, \mathtt{R}\}$ y los nodos accesibles de $\mathtt{S}$ son $\mathcal{AC}(\mathtt{S}) = \{\mathtt{S}, \mathtt{I}, \mathtt{R}\}$. □

Ejemplo 12 (Ejemplo 9 cont.) Las siguientes son las restricciones de integridad para cada nodo del sistema P2P del ejemplo 10.

$IC(\mathtt{R}) = \{\forall x(D(x) \rightarrow \exists y(P(x, y)))\}$

$IC(\mathtt{S}) = \{\forall xy(M(x, y) \rightarrow \exists z(C(x, z))), \forall xy_1y_2(C(x, y_1) \wedge C(x, y_2) \rightarrow y_1 = y_2)\}$

$IC(\mathtt{I}) = \emptyset$

Se puede señalar que sólo la instancia $D(\mathtt{I})$ es consistente a nivel local, pues su conjunto de ICs es vacío. En tanto las instancias $D(\mathtt{R})$ y $D(\mathtt{S})$ no son consistentes, dado que, en el primer caso, la tupla $D(5)$ no tiene su correspondencia en la relación P, mientras que en el segundo caso, falta la correspondiente tupla en C para $M(2, 3)$. Sólo la DEC $\Sigma(\mathtt{S}, \mathtt{R})$ no se satisface, puesto que para C y P no coinciden los valores en el atributo y para el valor 1 en x. Por el contrario, $\Sigma(\mathtt{S}, \mathtt{I})$ y $\Sigma(\mathtt{I}, \mathtt{R})$ se satisfacen, ya que existe una correspondencia entre las tuplas en M y L, así como entre las tuplas en L y P. □

Localmente, cada nodo es responsable de mantener su instancia consistente con respecto a sus restricciones de integridad, independiente de los otros nodos, lo que en términos de sistemas P2P, se denomina consistencia a nivel local. Estos aspectos fueron descritos en la Sección 2.3. Sin embargo, cuando una consulta es hecha a un nodo $\mathtt{P}$, que participa de un sistema P2P, no basta con comprobar la consistencia a nivel, para entregar respuestas consistentes. $\mathtt{P}$ puede necesitar considerar tanto sus propios datos como los datos almacenados en los otros nodos, que están relacionados con $\mathtt{P}$ mediante DECs y relaciones de confianza, además puede significar no usar parte de los propios datos. Bajo el esquema de los sistemas P2P, los nodos deben mantener la consistencia a nivel de intercambio de datos, mediante la satisfacción de las DECs conforme con las relaciones de confianza, además de mantenerse consistentes localmente.

Capítulo 3

Consistencia en Sistemas P2P

En esta sección se describen aspectos lógicos específicos de los sistemas de intercambio de datos P2P. Se analizan los elementos que afectan la consistencia de los nodos y toda la semántica desarrollada en [6, 7] para obtener respuestas consistentes a consultas en sistemas P2P. Se describen los posibles cambios virtuales que deben hacerse en los nodos para restaurar la consistencia. Se finaliza presentando el programa de reparación en lógica disyuntiva, para obtener respuestas consistentes de nodos.

Si un nodo P confía más en los datos de Q que en los propios, P acomodará sus propios datos a los de Q, para mantener las restricciones de intercambio $\Sigma(\mathtt{P}, \mathtt{Q})$ satisfechas. El problema que origina esta situación es que por satisfacer a $\Sigma(\mathtt{P}, \mathtt{Q})$, las $IC(\mathtt{P})$ o las $IC(\mathtt{Q})$ podrían ser transgredidas, provocando que P o Q queden en un estado de inconsistencia.

Idealmente, las respuestas a la consulta obtenidas por P deben ser consistentes con respecto a las DEC's $\Sigma(\mathtt{P})$ y con las ICs $IC(\mathtt{P})$. En principio, P, que no está habilitado para cambiar los datos de otros nodos, podría tratar de *reparar* su base de datos con tal de satisfacer $\Sigma(\mathtt{P}) \cup IC(\mathtt{P})$, pero esto no se lleva a cabo. P envía subconsultas a otros nodos involucrados en sus DECs, resolviendo sus conflictos semánticos o incompletitud de datos, en tiempo de consulta, llevando a cabo *reparaciones virtuales*, de manera que las respuestas que se intercambian sean PCAs.

En resumen, para la obtención de PCAs [6], es necesario incurrir en cambios virtuales

a los propios datos realizándolos en tiempo de consulta. Para acomodarse a los datos de los otros nodos, las IC locales pueden ser violadas virtualmente. Es posible mantener las IC satisfechas incluso en tiempo de consulta, por medio del uso de metodologías desarolladas para respuestas consistentes a consultas [2].

La semántica de consistencia para sistemas P2P desarrollada por [7], define un conjunto de instancias globales virtuales, llamada instancias de solución (o simplemente soluciones) para un nodo. En este sentido, los datos consistentes para un nodo `P` son los que están en sus instancias de solución, y ellos dependen por sobre de la instancia de `P`, de las instancias locales de los nodos relacionados, de la satisfacción de las DECs y de la satisfacción de sus ICs locales. Las respuestas consistentes de nodos para `P` a una consulta $\mathcal{Q}$ son definidas como aquéllas respuestas que pueden ser obtenidas de cada solución posible para `P`. Lo que el nodo `P` entrega al usuario son todas las respuestas verdaderas en cada una de las instancias de solución. Las instancias de solución de un nodo están determinadas tanto por los nodos involucrados en DECs con el nodo `P` como por los nodos involucrados con los nodos involucrados en DECs con el nodo `P`, y así, sucesivamente, hasta que no queden DECs por verificar.

3.1. Semántica de PCAs

Bajo esta semántica, definida en [7], los datos presentes en los nodos deben ser filtrados adecuadamente para poder producir PCAs, y para ello,se utilizó la siguiente idea: los datos que un nodo `P` utiliza, de un nodo vecino `Q`, para construir sus propias soluciones, son los presentes en la intersección de las soluciones de `Q`. Luego de que `P` recibe estos datos, solamente se consideran las DECs e ICs del propio `P`. Esto se conecta con la idea de las reparaciones y las respuestas consistentes a nivel local, donde las respuestas consistentes son las presentes en la intersección de todas las reparaciones. Notar que si se aplica la misma idea al nodo `Q`, la noción de solución se transforma en recursiva. Por este motivo, la semántica para la obtención de PCAs es influenciada por la presencia de ciclos en el grafo de accesibilidad del sistema P2P $\mathcal{G}_A(\beta)$, asi que se

asume que $\mathcal{G}_A(\beta)$ es acíclico.

En un sistema P2P se pueden tener otras fuentes para ciclos. Hay un ciclo a través de restricciones de integridad si el grafo de dependencias $\mathcal{G}(\mathtt{P})$ es cíclico RIC (ver definición 1). Aún teniendo grafos de accesibilidad y grafos de dependencias acíclicos, se pueden tener ciclos en un conjunto de restricciones, DECs e ICs, dada su generalidad, pudiendo tener relaciones de cualquiera de los dos nodos implicados en el antecedente así como en el consecuente de la sentencia, denominados ciclos REF. Es así como para un nodo $\mathtt{P}$, el grafo de dependencias construído a partir de $\Sigma(\mathtt{P}) \cup IC(\mathtt{P})$ se dice que es *cíclico REF* [7], cuando existen ciclos a través de RDECs o RICs.

Para que el problema de la obtención de PCAs sea decidible, se requiere que para cada nodo $\mathtt{P}$, el grafo de dependencias $\mathcal{G}(\mathtt{P})$ sea acíclico RIC. Además, se requiere que el grafo de dependencias construído a partir de $\Sigma(\mathtt{P}) \cup IC(\mathtt{P})$ sea acíclico REF [7].

Ejemplo 13 Considere un sistema P2P con dos nodos, $\mathtt{P}_1$ y $\mathtt{P}_2$, que poseen los esquemas $\mathcal{R}(\mathtt{P}_1) = \{R^1(\cdot,\cdot), S^1(\cdot,\cdot)\}$ y $\mathcal{R}(\mathtt{P}_2) = \{R^2(\cdot,\cdot)\}$. $IC(\mathtt{P}_1) = \{\forall xy(S^1(x,y) \rightarrow \exists z(R^1(x,z)))\}$. $IC(\mathtt{P}_2) = \emptyset$. La confianza es $(\mathtt{P}_1, menos, \mathtt{P}_2)$ y las DECs son $\Sigma(\mathtt{P}_1, \mathtt{P}_2) = \{\forall xy(R^1(x,y) \rightarrow R^2(x,y)), \forall xy(R^2(x,y) \rightarrow \exists z(S^1(x,z)))\}$ Si se observa la Figura 3.1, el grafo de accesibilidad (1) es acíclico, pero en cambio el grafo de dependencias (2), construído en base a $\Sigma(\mathtt{P}_1, \mathtt{P}_2)) \cup IC(\mathtt{P}_1)$, no lo es. Notar que si la RDEC fuera UDEC o la RIC fuera UIC no existirían tales ciclos REF. □

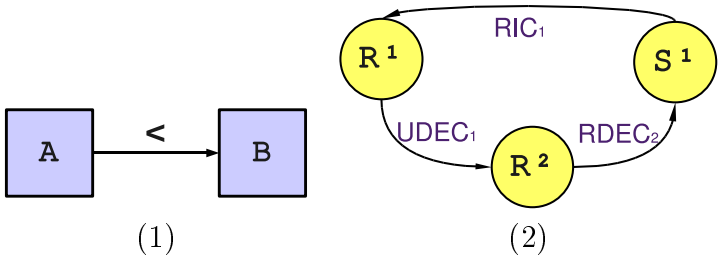

Figura 3.1: Grafos de Accesibilidad y de Dependencia Ejemplo 13

Con respecto a la comprobación de la consistencia, las DECs son satisfechas de forma similar en que lo son las ICs, vistas en la sección 2.3, conforme a los atributos

relevantes para una restricción (ya sea DEC o IC) y reparando con nulos en el caso de las RDEC. El elemento nuevo es que ahora se toman en cuenta las relaciones de confianza para realizar reparaciones en los predicados de los nodos que confían más en otros que en sus propios datos.

Definición 6 Dado un nodo P en un sistema P2P β, e instancias D y D' sobre el esquema compuesto por la unión de los esquemas de todos los nodos vecinos de P, $\mathcal{R}(\mathcal{N}(\mathtt{P})) = \bigcup_{\mathtt{Q}\in\mathcal{N}(\mathtt{P})}\mathcal{R}(\mathtt{Q})$, $\mathcal{R}(\mathtt{P})$, D' es una *solución de vecinos* [7], para el nodo P si:
(a) $D' \models \Sigma(\mathtt{P}) \cup IC(\mathtt{P}) = D' \models \bigcup_{\mathtt{Q}\in\mathcal{N}(\mathtt{P})}\Sigma(\mathtt{P},\mathtt{Q}) \cup IC(\mathtt{P})$.
(b) $D'|R = D|R$, para cada predicado $R \in \mathcal{R}^{menos}(\mathtt{P})$, según la definición 3.
(c) No existe una instancia D'' que satisfaga (1) y (2) tal que $D'' <_D D'$. □

No se requiere que se satisfagan $IC(\mathtt{Q})$, porque Q trasladará sus datos al sitio del nodo P, donde las inconsistencias serán resueltas localmente, conforme con la definición 7, donde $\mathcal{S}(\mathtt{P})$ denota el conjunto de soluciones para el nodo P.

Definición 7 Sea un nodo P y una instancia D del esquema $\mathcal{R}(\mathtt{P})$. D es una *instancia solución* [7] para P si:
(a) $D = D(\mathtt{P})$ y $(\Sigma(\mathtt{P}) \cup IC(\mathtt{P})) = \emptyset$ o bien
(b) $(\Sigma(\mathtt{P})\cup IC(\mathtt{P})) \neq \emptyset$, $D = \overline{D}|\mathtt{P}$, donde $\overline{D}$ es la solución de vecinos para P y la instancia de la base de datos $D(\mathtt{P}) \cup \bigcup_{\mathtt{Q}\in(\mathcal{N}(\mathtt{P})\setminus\{\mathtt{P}\})}\bigcap_{\mathcal{I}\in\mathcal{S}(\mathtt{Q})}\mathcal{I}$, sobre el esquema $\bigcup_{\mathtt{Q}\in\mathcal{N}(\mathtt{P})}\mathcal{R}(\mathtt{Q})$. □

Esta definición sugiere que la obtención de PCAs tiene una solución recursiva, donde cada vecino del nodo P tiene como instancia local la intersección de todas sus soluciones. Esto produce una base de datos combinada [7]. Las soluciones para P son obtenidas restringiendo a P las soluciones de vecinos de la instancia combinada. La solución de vecinos captura las actualizaciones que sean necesarias para satisfacer DECs e ICs locales. Así como pueden haber varias soluciones de vecinos, varias instancias de solución para un nodo son posibles.

Ejemplo 14 (Ejemplo 12 cont.) Considere las siguientes instancias para el sistema P2P β y la consulta $\mathcal{Q} : C(x, y)$.

P	x	y
	1	j
	2	m
	3	e

D	x
	5
	3
	1

Nodo R

C	x	y
	1	t
	3	e

M	x	z
	3	5
	2	3

L	x
	2
	3

Nodo S Nodo I

Para responder la consulta el nodo S requiere de los datos de los nodos R e I. Como el nodo R tiene inconsistencias, para satisfacer $IC(\mathtt{R})$ se generan dos reparaciones, una donde se inserta $P(5, null)$ y otra donde se elimina $D(5)$. La solución de vecinos de R es $\{P(1,j),\ P(2,m),\ P(3,e),\ D(1),\ D(3)\}$ que es su instancia solución. El nodo I también obtiene de R la solución anterior, y ahora debe reparar $\{P(1,j),\ P(2,m),\ P(3,e),\ D(1),\ D(3),\ L(2),\ L(3)\}$ cra $\Sigma(\mathtt{I},\mathtt{R})$. La DEC se satisface, por lo que la instancia solución de I es $\{L(2), L(3)\}$. Con la unión de las dos instancias solución, el nodo S debe reparar cra a $IC(\mathtt{S})$, a $\Sigma(\mathtt{S},\mathtt{R})$ y a $\Sigma(\mathtt{S},\mathtt{I})$ a partir de los datos que obtuvo de sus vecinos y sus propios datos: $\{P(1,j),\ P(2,m),\ P(3,e),\ D(1),\ D(3),\ L(2),\ L(3),\ C(1,t),\ C(3,e),\ M(3,5),\ M(2,3)\}$. Existen cuatro soluciones de vecinos para S, ya que la confianza con R es *igual*, las tuplas $P(1,j)$ y $C(1,t)$ no satisfacen $\Sigma(\mathtt{S},\mathtt{R})$ y las tupla $M(2,3)$ no satisface la IC local de S.

$D'_1 = \{L(2), L(3), P(1,j), P(2,m), P(3,e), C(3,e), M(3,5), M(2,3), C(2,null)\}$

$D'_2 = \{L(2), L(3), P(1,j), P(2,m), P(3,e), C(3,e), M(3,5)\}$

$D'_3 = \{L(2), L(3), P(2,m), P(3,e), C(1,t), C(3,e), M(3,5)\}$

$D'_4 = \{L(2), L(3), P(2,m), P(3,e), C(1,t), C(3,e), M(3,5), M(2,3), C(2,null)\}$

Existen cuatro instancias solución para S:

$\mathcal{S}_1 = \{C(3,e), M(3,5), M(2,3), C(2,null)\}$

$\mathcal{S}_2 = \{C(3,e), M(3,5)\}$

$\mathcal{S}_3 = \{C(1,t), C(3,e), M(3,5)\}$

$\mathcal{S}_4 = \{C(1,t), C(3,e), M(3,5), M(2,3), C(2,null)\}$ □

Una solución para un nodo captura la idea que sólo algunas bases de datos de nodos son relevantes para P. Se asume que los datos y DECs que son relevantes son

sólo aquellos correspondientes a los nodos accesibles de P. En este sentido, esta es una noción recursiva, ya que para obtener las soluciones de los nodos accesibles lo que se hace es obtener la intersección de las soluciones de los vecinos de los vecinos, y así sucesivamente. Es decir, se toman en cuenta las dependencias transitivas.

Ahora, si pensamos en lo que sucede en cada nodo, al recibir una consulta es necesario definir el concepto de respuesta a una consulta.

3.2. Respuestas Consistentes

Definición 8 Dada una instancia D, una tupla de constantes $\bar{t}$ es una *respuesta a una consulta* [1] $\mathcal{Q}$ en D ssi $D \models \mathcal{Q}(\bar{t})$, es decir, $\mathcal{Q}(\bar{x})$ se hace verdadero en D cuando las variables son reemplazadas por las correspondientes constantes en $\bar{t}$. En una base de datos posiblemente inconsistente, una *respuesta consistente a una consulta* [1] es aquella respuesta a la consulta $\mathcal{Q}(\bar{x})$ en cada reparación D' de D. El conjunto de respuestas consistentes a una consulta $\mathcal{Q}(\bar{x})$ en D, se denota por $ConsA(\mathcal{Q})$.

Dada una consulta de primer orden $\mathcal{Q}(x) \in \mathcal{L}(\mathtt{P})$ formulada a un nodo P, una tupla instanciada $\bar{t}$ es una *respuesta consistente de nodo* [6] para $\mathcal{Q}$, denotada por PCA (Peer Consistent Answer) de P ssi $D' \models \mathcal{Q}(\bar{t})$ en cada solución local D' de P. □

Ejemplo 15 (Ejemplo 14 cont.) Para la consulta $\mathcal{Q} : C(x, y)$, formulada al nodo S, son PCAs las tuplas presentes en cada solución. Por ende, la única PCA para esta consulta es $C(3, e)$. □

Se ha mostrado en [6], que a partir de programas de reparación, se pueden obtener las reparaciones para una base de datos completa, es decir, toda una instancia. Se pueden especificar soluciones para un nodo, como los modelos estables de programas en lógica disyuntiva [6]. Estos programas, similares a los expuestos en [11], se adaptan ahora para garantizar la satisfacción de DECs, respetando las relaciones de confianza. Se incorpora la utilización de nulos para reparar restricciones referenciales, tema expuesto en la sección 2.3.

Los programas de reparación computan todas las reparaciones de la base de datos, lo que implica que la producción de instancias reparadas completas, que contienen todas las tuplas consistentes de la base de datos. Sin embargo, para responder a una consulta no es necesario recuperar todas las tuplas, sino solamente las involucradas en la consulta. A menos de que se desee consultar una instancia completa, para computar las *respuestas consistentes a una consulta*, no basta con obtener todas las reparaciones, esto es el primer paso.

Es así como es introducida el concepto de *predicado relevante* para una consulta, desarrollado por [11].

Definición 9 Un predicado P es relevante [11] para obtener respuestas consistentes a una consulta $\mathcal{Q}$, con respecto al conjunto de restricciones de integridad IC, si P es un componente conectado a c en el grafo $\mathcal{G}(IC)$, y c aparece en la consulta $\mathcal{Q}$. $Rel(\mathcal{Q}, IC)$ denota el conjunto de predicados relevantes para $\mathcal{Q}$ e IC. □

En virtud de las condiciones que deben darse para que la semántica de PCAs sea aplicable, en términos de la aciclicidad de los grafos de dependencias y los grafos de accesibilidad, es posible aplicar la misma idea de predicado relevante, ahora en los sistemas P2P, considerando que se ha formulado una consulta $\mathcal{Q}$ a un nodo $\mathtt{P}$, escrita en $\mathcal{L}(\mathtt{P})$, y el conjunto de restricciones $\Re(\mathtt{P}) = \Sigma(\mathtt{P}) \cup IC(\mathtt{P})$.

Definición 10 Un predicado R es relevante para obtener respuestas consistentes de nodos, a una consulta $\mathcal{Q}$, formulada a un nodo $\mathtt{P}$, con respecto al conjunto de restricciones $\Re(\mathtt{P}) = \Sigma(\mathtt{P}) \cup IC(\mathtt{P})$, si R es un componente conectado a c en el respectivo grafo de dependencias $\mathcal{G}(\Re(\mathtt{P}))$, y c aparece en la consulta $\mathcal{Q}$. $Rel(\mathcal{Q}, \Re(\mathtt{P}))$ denota el conjunto de predicados relevantes para $\mathcal{Q}$ y $\Re(\mathtt{P})$. □

Ejemplo 16 Considere un sistema P2P con tres nodos, $\mathtt{A}$,$\mathtt{B}$ y $\mathtt{C}$. Los esquemas de los nodos son $\mathcal{R}(\mathtt{A}) = \{A^1(\cdot), A^2(\cdot), A^3(\cdot), A^4(\cdot)\}$, $\mathcal{R}(\mathtt{B}) = \{B^1(\cdot), B^2(\cdot,\cdot)\}$ y $\mathcal{R}(\mathtt{C}) = \{C^1(\cdot), C^2(\cdot)\}$. $IC(\mathtt{A}) = \{\forall x(A^1(x) \rightarrow A^2(x))\}$. La relación de confianza es $\{(\mathtt{A}, \mathit{menos}, \mathtt{B}), (\mathtt{A}, \mathit{igual}, \mathtt{C})\}$. Las DECs son $\Sigma(\mathtt{A}, \mathtt{B}) = \{\forall x(A^1(x) \rightarrow B^1(x))\}$, $\Sigma(\mathtt{A}, \mathtt{C}) = \{\forall x(A^4(x)$

$\rightarrow C^1(x))\}$. Las ICs son $IC(\mathtt{A}) = \{\forall x(A^1(x) \rightarrow A^2(x)), \forall x(A^3(x) \rightarrow A^4(x))\}$, $IC(\mathtt{B}) = \{\forall x(B^1(x) \rightarrow \exists y(B^2(x,y)))\}$, $IC(\mathtt{C}) = \{\forall x(C^1(x) \rightarrow C^2(x))\}$. La consulta es $\mathcal{Q} : A^1(x)$.

La Figura 3.2 muestra el el grafo de dependencias $\mathcal{G}(\Re(\mathtt{P}))$, que es acíclico REF, cuyos componentes conectados son por un lado, $\{A^1, A^2, B^1, B^2\}$ y por otro, $\{A^3, A^4, C^1, C^2\}$. Para la consulta $\mathcal{Q} : A^1(x)$, los predicados relevantes son $Rel(\mathcal{Q}, \Re(\mathtt{P})) = \{A^1, A^2, B^1, B^2\}$. □

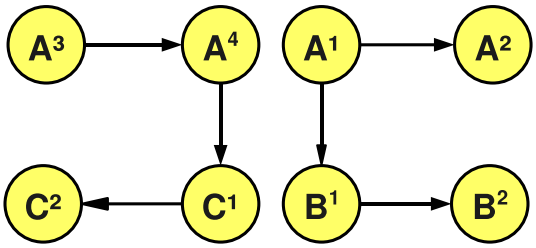

Figura 3.2: Grafo de Dependencias $\mathcal{G}(\Re(\mathtt{P}))$ Ejemplo 16.

3.3. Programa Solución

Los programas de reparación se computan para obtener reparaciones y así reconstituir la consistencia de una base de datos. Las reparaciones son especificadas como modelos estables de DLPs de reparación. De acuerdo a la definición de DLPs (sección 2.1), cualquier base de datos puede ser representada como un programa DLP Π. El conjunto de restricciones de integridad de la base de datos, IC, es transformado en las reglas del programa, mientras que instancia de la base de datos D, representa la base de datos extensional. Esto es, construir un programa $\Pi(D, IC)$.

Se debe hacer una especial distinción en que los DLP expuestos están escritos en lógica de predicados de primer orden, y que para poder computarlos, en la práctica, deben ser reescritos en el lenguaje DATALOG.

El programa de reparación incorpora *anotaciones* a las reglas del programa. Estas anotaciones se componen de constantes añadidas a los predicados, que indican para cada átomo del programa, las alternativas que existen para lograr restaurar la consistencia.

Suponiendo como átomo del programa a $P(\bar{a})$, las anotaciones son las siguientes:

- $P\_(\bar{a}, \mathbf{t}_a) : P(\bar{a})$ es aconsejable que se haga verdadero.
- $P\_(\bar{a}, \mathbf{f}_a) : P(\bar{a})$ es aconsejable que se haga falso.
- $P\_(\bar{a}, \mathbf{t}^*) : P(\bar{a})$ es verdadero o se hace verdadero.
- $P\_(\bar{a}, \mathbf{t}^{**}) : P(\bar{a})$ es verdadero en la reparación.

La siguiente es una definición del programa de reparación para la obtención de PCAS (Peer Consistent Answers), denominado *Programa Solución*, desarrollado por [7], que ha sido modificado conforme a las optimizaciones presentadas por [10], para programas de reparación que obtienen CQAs (Consistent Query Answers).

El programa requiere como entradas las instancias de los nodos, las relaciones de confianza, el conjunto de restricciones de integridad, el conjunto de restricciones de intercambio y la consulta (o subconsulta) que desea responderse. Además, como precondición, se requiere que el grafo de dependencias contraído, construído a partir de las ICs dadas, sea acíclico RIC.

El programa solución, que resulta de estas modificaciones, adopta ahora un enfoque en la consulta. Mientras que el original obedecía a la obtención de instancias completas, esta versión se centra en obtener respuestas a las consultas (o subconsultas) que se han formulado. El programa solución tiene un carácter recursivo, dada la naturaleza del problema, por lo que tanto la consulta como las instancias de entrada al programa irán variando, conforme vayan avanzando las llamadas recursivas. La diferencia existente entre entregar CQAs y PCAs radica en que las CQAs son las que se entregan al usuario en última instancia, en respuesta a su consulta, una vez que la recursión ha concluído; mientras que las PCAs son las tuplas que se entregan en respuesta a las subconsultas surgidas con el intercambio entre nodo y nodo, en cada llamada recursiva. En este sentido, se ha incorporado al programa solución una regla adicional, que computa el programa de consulta $\Pi(\mathcal{Q})$, necesario para responder a la consulta según sea el caso, con PCAs o con CQAs.

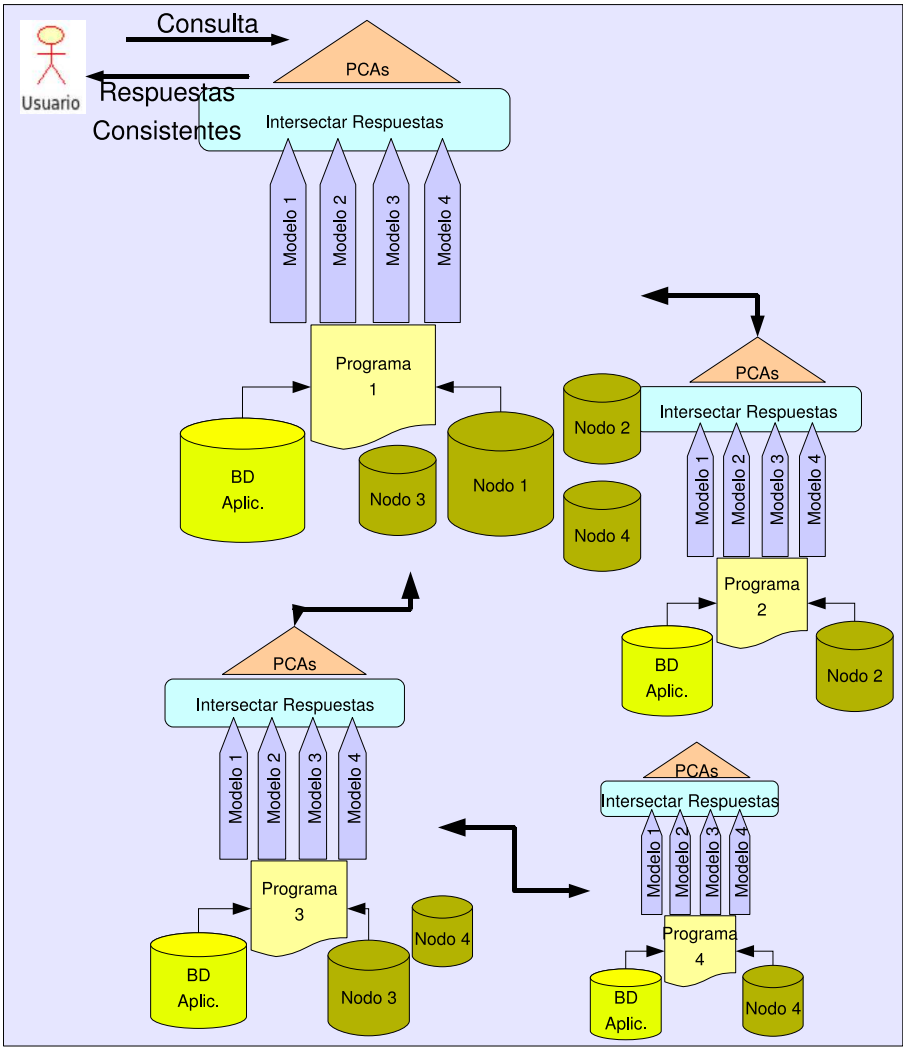

Figura 3.3: Estructura Arbol del Programa de Reparación

El programa de reparación puede ser visto como una estructura de árbol $n - ario$, donde cada hoja es un nodo `P` que participa del sistema P2P y que cuyas n ramas son la cantidad de vecinos de `P`. La Figura 3.3 muestra la idea de cómo los programas de reparación se deben computar, en cada hoja, para obtener las respuestas consistentes a una consulta. Es el usuario quien evalúa una consulta en un nodo 1, momento en que se genera el programa 1. El programa 1 debe satisfacer DECs que tiene el nodo 1 con todos sus vecinos además de sus propias ICs. El nodo 1 formula subconsultas para sus vecinos 2, 3. Para responder a la subconsulta, el nodo 2, lo hace de manera que sus

respuestas sean consistentes con sus ICs locales. Mientras que el nodo 3 debe hacer a su vez una subconsulta al nodo 4, quien le responde con sus respuestas consistentes, igual que en el caso del nodo 2, localmente. En el nodo 3 se computa el programa 3 reparando con respecto a las DECs e ICs del nodo 3 e incorporando las respuestas del nodo 4 como hechos del programa 3. Así, en el nodo 1 se ejecuta el programa 1, donde se recuperan las respuestas a las subconsultas formuladas a los nodos 2 y 3, las cuales se incorporan como hechos del programa 1. Así, el programa 1 es finalmente computado para obtener las respuestas consistentes a la consulta original que formuló el usuario.

Definición 11 Considérese un sistema P2P de intercambio de datos $\beta = \langle \mathcal{P}, \Sigma, IC, confianza \rangle$, un nodo $\mathtt{P} \in \mathcal{P}$, con $\mathcal{N}(\mathtt{P}) = \{\mathtt{P}, \mathtt{P_1}, \ldots, \mathtt{P_n}\}$ y el conjunto $\mathcal{I} = \{I_1, \ldots, I_n\}$, donde I_j es una instancia de base de datos sobre el esquema de $\mathtt{P_j}$. Considere además una consulta de primer orden $\mathcal{Q}$, formulada en $\mathcal{L}(\mathtt{P})$. Sea

$$\mathcal{Q}(\bar{x}) = \bigwedge_{i=1}^{n} R_i(\bar{x}_i), \bigwedge_{j=1}^{m} \mathit{not}\ R_j(\bar{x}_j), \varphi; \tag{3.1}$$

con $R_i, R_j \in \mathtt{P}$. El *programa solución* para la obtención de PCAs, denotado por $\Pi(\beta, \mathtt{P}, \mathcal{I}, \mathcal{Q})$ es el siguiente:

1. $R(\bar{a})$ para cada átomo $R(\bar{a}) \in D(\mathtt{P})$, tales que $R \in Rel(\mathcal{Q}, \Re(\mathtt{P}))$.

   $R(\bar{a})$, para cada $R(\bar{a}) \in I$ con $I \in \mathcal{I}$, tales que $R \in Rel(\mathcal{Q}, \Re(\mathtt{P}))$.

2. Para cada UDEC $\psi \in \Sigma(\mathtt{P}, \mathtt{P_j})$, de la forma (2.7), tal que $\mathtt{P_j} \in \mathcal{N}(\mathtt{P})$ y exista una relación de confianza del tipo $(\mathtt{P}, \{\mathit{menor\ o\ igual}\}, \mathtt{P_j})$, las reglas:

$$\bigvee_{R \in R_{\mathtt{P}}} R\_(\bar{x}_i, \mathbf{f_a}) \vee \bigvee_{Q \in Q_{\mathtt{P}}} Q\_(\bar{y}_j, \mathbf{t_a}) \ \leftarrow \ \bigwedge_{i=1}^{n} R_i\_(\bar{x}_i, \mathbf{t}^*), \bigwedge_{Q\_j \in Q'} Q_j\_(\bar{y}_j, \mathbf{f_a}), \bigwedge_{Q_k \in Q''} \mathit{not}\ Q_k(\bar{y}_k), \bigwedge_{x_l \in \mathcal{A}(\psi) \cap \bar{x}} x_l \neq \mathit{null},\ \bar{\varphi}.$$

   para cada par de conjuntos Q' y Q'' de átomos que aparecen en la fórmula 2.7, tales que $Q' \cup Q'' = \bigcup_{j=1}^{m} Q_j(\bar{y}_j)$ y que $Q' \cap Q'' = \emptyset$, donde $\bar{x} = \bigcup_{i=1}^{n} \bar{x}_i$. Aquí, $\mathcal{A}(\psi)$ es el conjunto de atributos relevantes de ψ, y $\bar{\varphi}$ es una conjunción de built-ins que

es equivalente a la negación de φ. $\mathcal{R} = \{R_i \mid i \in \{1, \ldots, n\}$, donde todos los R_i son predicados que aparecen en la fórmula de UDEC, tales que $R_i \in Rel(\mathcal{Q}, \Re(\mathtt{P}))\}$. $R_{\mathtt{P}} = \mathcal{R} \cap \mathcal{R}(\mathtt{P})$, si $(\mathtt{P}, menos, \mathtt{P_j})$; mientras que $R_{\mathtt{P}} = \mathcal{R}$ si $(\mathtt{P}, igual, \mathtt{P_j})$. $Q_{\mathtt{P}}$ es definido en forma análoga, en términos de los predicados Q_j que aparecen en la fórmula (2.7, tales que $Q_j \in Rel(\mathcal{Q}, \Re(\mathtt{P}))$).

3. Para cada RDEC $\psi \in \Sigma(\mathtt{P}, \mathtt{P_j})$, de la forma (2.8), tal que $\mathtt{P_j} \in \mathcal{N}(\mathtt{P})$ y exista $(\mathtt{P}, \{menos\ o\ igual\}, \mathtt{P_j}) \in confianza$, tales que los predicados R y Q pertenezcan a $Rel(\mathcal{Q}, \Re(\mathtt{P}))$.:

   - Si $(\mathtt{P}, igual, \mathtt{P_j}) \in confianza$, la regla:

     $R\_(\bar{x}, \mathbf{f_a}) \vee Q\_(\bar{x}', \overline{null}, \mathbf{t_a}) \leftarrow R\_(\bar{x},\ \mathbf{t}^*),\ \ not\ aux_\psi(\bar{x}'),\ \bar{x}' \neq null.$

   - $Si(\mathtt{P}, menos, \mathtt{P_j}) \in confianza$ y $R \in \mathcal{R}(\mathtt{P})$, la regla:

     $R\_(\bar{x}, \mathbf{f_a}) \leftarrow R\_(\bar{x}, \mathbf{t}^*),\ not\ aux_\psi(\bar{x}'),\ \bar{x}' \neq null.$

   - Si $(\mathtt{P}, menos, \mathtt{P_j}) \in confianza$ y $Q \in \mathcal{R}(\mathtt{P})$, la regla:

     $Q\_(\bar{x}', \overline{null}, \mathbf{t_a}) \leftarrow R\_(\bar{x}, \mathbf{t}^*),\ not\ aux_\psi(\bar{x}'),\ \bar{x}' \neq null.$

     más las reglas auxiliares:

     $aux_\psi(\bar{x}') \leftarrow Q(\bar{x}', \overline{null}),\ not\ Q\_(\bar{x}', \overline{null}, \mathbf{f_a}),\ \bar{x}' \neq null.$

     Para cada $y_i \in \bar{y}$ la regla:

     $aux_\psi(\bar{x}') \leftarrow Q\_(\bar{x}', \bar{y}, \mathbf{t}^*),\ not\ Q\_(\bar{x}', \bar{y}, \mathbf{f_a}),\ \bar{x}' \neq null,\ y_i \neq null.$

4. Para cada UIC $\psi \in IC(\mathtt{P})$, de la forma (2.7), tales que los predicados P_i y Q_j pertenezcan a $Rel(\mathcal{Q}, \Re(\mathtt{P}))$, las reglas:

$$\bigvee_{i=1}^{n} P_i\_(\bar{x}_i, \mathbf{f_a}) \vee \bigvee_{j=1}^{m} Q_{j\_}(\bar{y}_j, \mathbf{t_a}) \ \leftarrow\ \bigwedge_{i=1}^{n} P_i\_(\bar{x}_i, \mathbf{t}^*), \bigwedge_{Q\_j \in Q'} Q_{j\_}(\bar{y}_j, \mathbf{f_a}), \bigwedge_{Q_k \in Q''} not\ Q_k(\bar{y}_k), \bigwedge_{x_l \in \mathcal{A}(\psi) \cap \bar{x}} x_l \neq null,\ \bar{\varphi}.$$

   para cada par de conjuntos Q' y Q'' de átomos que aparecen en la fórmula (2.7), tales que $Q' \cup Q'' = \bigcup_{j=1}^{m} Q_j(\bar{y}_j)$ y que $Q' \cap Q'' = \emptyset$, donde $\bar{x} = \bigcup_{i=1}^{n} \bar{x}_i$, $\mathcal{A}(\psi)$ es

el conjunto de atributos relevantes de ψ y $\bar{\varphi}$ es una conjunción de átomos built-in equivalentes a la negación de φ.

5. Para cada RIC $\psi \in IC(\mathtt{P})$, de la forma (2.8), tales que los predicados P y Q pertenezcan a $Rel(\mathcal{Q}, \Re(\mathtt{P}))$, la regla:

   $P_{\_}(\bar{x}, \mathbf{f_a}) \vee Q_{\_}(\bar{y}, \overline{null}, \mathbf{t_a}) \leftarrow P_{\_}(\bar{x}, \mathbf{t}^*),\ not\ aux_{\psi}(\bar{y}),\ \bar{y} \neq null.$

   $aux_{\psi}(\bar{y}) \leftarrow Q(\bar{y}, \overline{null}),\ not\ Q_{\_}(\bar{y}, \overline{null}, \mathbf{f_a}),\ \bar{y} \neq null.$

   Para cada $z_i \in \bar{z}$:

   $aux_{\psi}(\bar{y}) \leftarrow Q_{\_}(\bar{y}, \bar{z}, \mathbf{t}^*),\ not\ Q_{\_}(\bar{y}, \bar{z}, \mathbf{f_a}),\ \bar{y} \neq null,\ z_i \neq null$

6. Para cada predicado $R \in Rel(\mathcal{Q}, \Re(\mathtt{P}))$, las reglas de anotación:

   $R_{\_}(\bar{x}, \mathbf{t}^*) \leftarrow R(\bar{x}).$

   $R_{\_}(\bar{x}, \mathbf{t}^*) \leftarrow R_{\_}(\bar{x}, \mathbf{t_a}).$

7. Para cada predicado $R \in Rel(\mathcal{Q}, \Re(\mathtt{P}))$, la regla de interpretación:

   $R_{\_}(\bar{x}, \mathbf{t}^{**}) \leftarrow R_{\_}(\bar{x}, \mathbf{t}^*),\ not\ R_{\_}(\bar{x}, \mathbf{f_a}).$

8. Para cada restricción NNC de la forma (2.7), tales que $P \in Rel(\mathcal{Q}, \Re(\mathtt{P}))$, las cláusulas:

   $P_{\_}(\bar{x}, \mathbf{f}_a) \leftarrow P_{\_}(\bar{x}, \mathbf{t}^*), x_i = null.$

9. Para cada predicado que sea vértice intermedio en el grafo de dependencias construído a partir de todos los predicados $R \in \{\Sigma(\mathtt{P}) \cup IC(\mathtt{P})\}$, la restricción de negación del programa: $\leftarrow R_{\_}(\bar{x}, \mathbf{t_a}), R_{\_}(\bar{x}, \mathbf{f_a}).$

10. Para la consulta $\mathcal{Q}(\bar{x})$, incorporar en el cuerpo de la regla: Cada literal $R_i(\bar{x}_i) \in \mathcal{Q}$ como $R_{i\_}(\bar{x}_i, \mathbf{t}^{**})$. Cada literal $not\ R_j(\bar{x}_j) \in \mathcal{Q}$ como $not\ R_{j\_}(\bar{x}_j, \mathbf{t}^{**})$. La conjunción de builts-in φ: $ans(\bar{x}) \leftarrow \bigwedge_{i=1}^{n} R_{i\_}(\bar{x}_i, \mathbf{t}^{**}), \bigwedge_{j=1}^{m} not\ R_{j\_}(\bar{x}_j, \mathbf{t}^{**}), \varphi.$

□

Los hechos del programa corresponden a los hechos de la instancia del nodo P y a los hechos en las instancias $I(\mathtt{P}_i)$ de los vecinos $\mathtt{P}_i$. Las instancias $I(\mathtt{P}_i)$ usadas en el programa pueden no coincidir con las instancias físicas $D(\mathtt{P}_i)$. De hecho, cada $I(\mathtt{P}_i)$ es la intersección de las soluciones para $\mathtt{P}_i$, y por ende, existe una correspondencia de uno a uno entre los modelos estables del programa y las soluciones para el nodo P [7].

Dado que se ejecutan actualizaciones virtuales en las instancias de los nodos, sus ICs locales deben mantenerse satisfechas. Para ello se incorporan al programa las reglas 4 y 5. De la regla 1, se eliminó el predicado *dom*. Tanto las reglas en 2 como en 4 fueron modificadas. Se han eliminado del programa las anotaciones $\mathbf{f}^*$ que el programa original utilizaba, cambiándo los átomos que la contenían $R\_(\bar{x}, \mathbf{f}^*)$, por el par de átomos del tipo $R\_(\bar{x}, \mathbf{f_a})$ y *not* $R(\bar{x})$. La regla 8 del programa también es modificada, añadiendo la condición del grafo de dependencias. Se adopta la semántica de los modelos estables para este programa solución, es decir, los modelos resultantes son sus modelos estables. En cada modelo estable, los átomos de la forma $R\_(\bar{x}, \mathbf{t}^{**})$ sólo contienen predicados pertenecientes a P, por lo tanto, estos átomos son los que definen la instancia solución para P. La regla 9 se incorpora para capturar las respuestas consistentes utilizando el predicado artificial *ans*. Es así como las ocurrencias de *ans* que coincidan en cada uno de los modelos estables del programa solución corresponden a las PCAs.

Definición 12 [7] La instancia de base de datos para P asociada al modelo estable $\mathcal{M}$ del programa $\Pi(\beta, \mathtt{P}, \mathcal{I})$ es $D_{\mathcal{M}} = \{R(\bar{a}) | R\_(\bar{a}, \mathbf{t}^{**}) \in \mathcal{M}\}$. □

Ejemplo 17 (Ejemplo 15 cont.) El programa solución para responder a la consulta $\mathcal{Q} : C(x, y)$ formulada al nodo S requiere hacer las subconsultas $\mathcal{Q}_1 : P(x, y)$ al nodo R y $\mathcal{Q}_2 : L(x)$ al nodo I. A su vez, el nodo I también deberá formular la subconsulta $\mathcal{Q}_3 : P(x, y)$ al nodo R. Es posible computar este programa solución, dado que el grafo de accesibilidad es acíclico, según se muestra en la Figura 2.3. Adicionalmente, se tiene que el grafo de dependencias de S es acíclico RIC (1). Se cumple también que los grafos de dependencias de las DECs e ICs tanto del nodo I como del nodo S son acíclicos REF, como se muestra en los grafos (2) y (3), respecivamente, en la Figura 3.4.

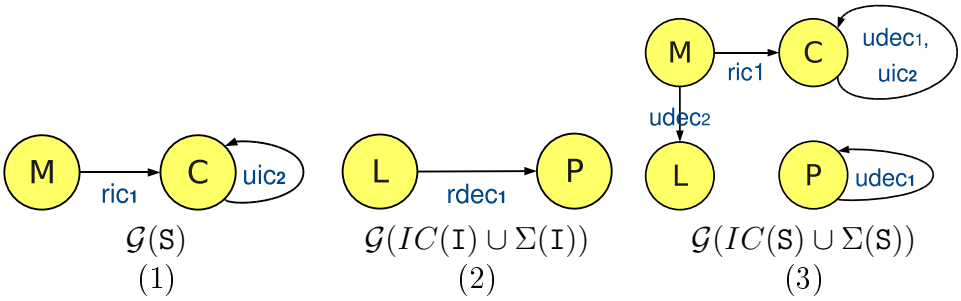

Figura 3.4: Grafos de Dependencia Ejemplo 17

Se deben computar tres programas, pero para el cómputo de los primeros se requiere el cómputo de los últimos, ya que se van incorporando las PCAs como hechos del programa en tiempo de ejecución. Se tienen el programa $\Pi_3(\beta, \mathtt{R}, I(\mathtt{R}), \mathcal{Q}_3)$, el programa $\Pi_2(\beta, \mathtt{I}, I(\mathtt{I}), \mathcal{Q}_2)$ y el programa final $\Pi_1(\beta, \mathtt{S}, I(\mathtt{R}) \cup I(\mathtt{I}), \mathcal{Q})$

Programa 1 El programa solución $\Pi_3(\beta, \mathtt{R}, I(\mathtt{R}), \mathcal{Q}_3)$ es el siguiente:

1. Instancia $D(\mathtt{R})$:

   $D(1).\ D(3).\ D(5).\ P(1, j).\ P(2, m).\ P(3, e).$

2. Reglas para $IC(\mathtt{R}) = \{\forall x(D(x) \rightarrow \exists y(P(x, y)))\}$

   $D\_(x, \mathbf{f_a}) \vee P\_(x, null, \mathbf{t_a}) \leftarrow D\_(x, \mathbf{t}^*), not\ aux_3(x), x \neq null.$

   $aux_3(x) \leftarrow P(x, null), not\ P\_(x, null, \mathbf{f_a}), x \neq null.$

   $aux_3(x) \leftarrow P\_(x, y, \mathbf{t}^*), not\ P\_(x, y, \mathbf{f_a}), x \neq null, y \neq null.$

3. Reglas de Anotación:

   $D\_(x_1, \mathbf{t}^*) \leftarrow D(x_1).\ D\_(x_1, \mathbf{t}^*) \leftarrow D\_(x_1, \mathbf{t_a}).$

   $P\_(x_1, x_2, \mathbf{t}^*) \leftarrow P(x_1, x_2).\ P\_(x_1, x_2, \mathbf{t}^*) \leftarrow P\_(x_1, x_2, \mathbf{t_a}).$

4. Reglas de Interpretación:

   $P\_(x_1, x_2, \mathbf{t}^{**}) \leftarrow P\_(x_1, x_2, \mathbf{t}^*), notP\_(x_1, x_2, \mathbf{f_a}).$

   $D\_(x_1, \mathbf{t}^{**}) \leftarrow D\_(x_1, \mathbf{t}^*), notD\_(x_1, \mathbf{f_a}).$

5. Programa de consulta $\mathcal{Q}_3 : P(x, y)$ $ANS(x, y) \leftarrow D\_(x, \mathbf{t}^{**}), P\_(x, y, \mathbf{t}^{**})$.

Los modelos estables para este programa son:

$\mathcal{M}_1 = \{P(1, j), P(2, m), P(3, e), D(1), D(3), D(5), P\_(5, null, \mathbf{t_a}), D\_(1, \mathbf{t}^{**}), D\_(3, \mathbf{t}^{**}),$
$D\_(5, \mathbf{t}^{**}), P\_(1, j, \mathbf{t}^{**}), P\_(2, m, \mathbf{t}^{**}), P\_(3, e, \mathbf{t}^{**}), P\_(5, null, \mathbf{t}^{**}),$
$ANS(1, j), ANS(3, e), ANS(5, null)\}$

$\mathcal{M}_2 = \{P(1, j), P(2, m), P(3, e), D(1), D(3), D(5), D\_(5, \mathbf{f_a}), D\_(1, \mathbf{t}^{**}), D\_(3, \mathbf{t}^{**}),$
$P\_(1, j, \mathbf{t}^{**}), P\_(2, m, \mathbf{t}^{**}), P\_(3, e, \mathbf{t}^{**}), ANS(1, j), ANS(3, e)\}$

Las respuestas consistentes para este programa son: $P(1, j)$. $P(2, m)$. $P(3, e)$.

Programa 2 El programa solución $\Pi_2(\beta, \mathtt{I}, I(\mathtt{R}), \mathcal{Q}_2)$ es el siguiente:

1. Instancias $D(\mathtt{I})$ e $I(\mathtt{R})$:

   $L(2)$. $L(3)$. $P(1, j)$. $P(2, m)$. $P(3, e)$.

2. Reglas para $\Sigma(\mathtt{I}, \mathtt{R}) = \{\forall x(L(x) \rightarrow \exists y(P(x, y)))\}$

   $L\_(x, \mathbf{f_a}) \leftarrow L\_(x, \mathbf{t}^*),\ not\ aux2(x), x \neq null.$

   $aux2(x) \leftarrow P(x, null),\ not\ P\_(x, null, \mathbf{f_a}), x \neq null.$

   $aux2(x) \leftarrow P\_(x, y, \mathbf{t}^*),\ not\ P\_(x, y, \mathbf{f_a}), x \neq null, y \neq null.$

3. Reglas de Anotación:

   $L\_(x, \mathbf{t}^*) \leftarrow L(x)$. $L\_(x, \mathbf{t}^*) \leftarrow L\_(x, \mathbf{t_a})$.

   $P\_(x, y, \mathbf{t}^*) \leftarrow P(x, y)$. $P\_(x, y, \mathbf{t}^*) \leftarrow P\_(x, y, \mathbf{t_a})$.

4. Reglas de Interpretación:

   $L\_(x, \mathbf{t}^{**}) \leftarrow L\_(x, \mathbf{t}^*),\ not\ L\_(x, \mathbf{f_a}).$

5. Programa de consulta $\mathcal{Q}_2 : L(x)$

   $ans(x) \leftarrow L\_(x, \mathbf{t}^{**})$.

Notar que en este programa no se han incorporado las restricciones de negación, debido a que el conjunto $\Sigma(\mathtt{I}) \cup IC(\mathtt{I})$ no presenta vértices intermedios. Si las incorporásemos los modelos resultantes serían los mismos. Sólo existe una solución de vecinos para el nodo $\mathtt{I}$, es decir, se tiene un único modelo estable para programa anterior:

$$\mathcal{M}_1 = \{P(1,j), P(2,m), P(3,e), L(2), L(3), L\_(2,\mathbf{t}^*), L\_(3,\mathbf{t}^*), aux(1), aux(2), aux(3),$$
$$P\_(1,j,\mathbf{t}^*), P\_(2,m,\mathbf{t}^*), P\_(3,e,\mathbf{t}^*), L\_(2,\mathbf{t}^{**}), L\_(3,\mathbf{t}^{**}), ans(2), ans(3)\}$$

En consecuencia, sólo existe una instancia solución: $I(\mathtt{I}) = D_{\mathcal{M}_1}(\mathtt{I}) = \{L(2), L(3)\}$.

Programa 3 El programa $\Pi_1(\beta, \mathtt{S}, I(\mathtt{R}) \cup \bigcap \mathcal{S}(\mathtt{I})) = \Pi_1(\beta, \mathtt{R}, I(\mathtt{R}) \cup \mathcal{S}(\mathtt{I}))$ es el siguiente:

(1) Instancias $D(\mathtt{S})$, $I(\mathtt{R})$ y $\mathcal{S}_1(\mathtt{I})$:

$C(1,t).\ C(3,e).\ M(3,5).\ M(2,3).\ P(1,j).\ P(2,m).\ P(3,e).\ L(2).\ L(3).$

(2) Reglas para las UDECs: $\Sigma(\mathtt{S},\mathtt{R}) = \{\forall xyz(C(x,y), P(x,z) \rightarrow y = z)\}$ y

$\Sigma(\mathtt{S},\mathtt{I}) = \{\forall xy(M(x,y) \rightarrow L(x))\}$, respectivamente:

$$C\_(x,y,\mathbf{f_a}) \vee P\_(x,z,\mathbf{f_a}) \leftarrow C\_(x,y,\mathbf{t}^*), P\_(x,z,\mathbf{t}^*), x \neq null, y \neq null,$$
$$z \neq null, y \neq z.$$

$$M\_(x,y,\mathbf{f_a}) \leftarrow M\_(x,y,\mathbf{t}^*), L\_(x,\mathbf{f_a}), x \neq null.$$

$$M\_(x,y,\mathbf{f_a}) \leftarrow M\_(x,y,\mathbf{t}^*),\ not\ L(x), x \neq null.$$

(3) Reglas para las ICs de $\mathtt{S}$:

$$C\_(x,y,\mathbf{f_a}) \vee C\_(x,z,\mathbf{f_a}) \leftarrow C\_(x,y,\mathbf{t}^*), C\_(x,z,\mathbf{t}^*), x \neq null, y \neq null,$$
$$z \neq null, y \neq z.$$

$$M\_(x,y,\mathbf{f_a}) \vee C\_(x,null,\mathbf{t_a}) \leftarrow M\_(x,y,\mathbf{t}^*),\ not\ aux(x), x \neq null.$$

$$aux(x) \leftarrow C(x,null),\ not\ C\_(x,null,\mathbf{f_a}), x \neq null.$$

$$aux(x) \leftarrow C\_(x,y,\mathbf{t}^*),\ not\ C\_(x,y,\mathbf{f_a}), x \neq null, y \neq null.$$

(4) Reglas de anotación:

$$P\_(x,y,\mathbf{t}^*) \leftarrow P(x,y).\ P\_(x,y,\mathbf{t}^*) \leftarrow P\_(x,y,\mathbf{t_a}).$$

$$C\_(x,y,\mathbf{t}^*) \leftarrow C(x,y).\ C\_(x,y,\mathbf{t}^*) \leftarrow C\_(x,y,\mathbf{t_a}).$$

$$M\_(x,y,\mathbf{t}^*) \leftarrow M(x,y).\ M\_(x,y,\mathbf{t}^*) \leftarrow M\_(x,y,\mathbf{t_a}).$$

$$L\_(x,\mathbf{t}^*) \leftarrow L(x).\ L\_(x,\mathbf{t}^*) \leftarrow L\_(x,\mathbf{t_a}).$$

(5) Reglas de interpretación:

$C\_(x, y, \mathbf{t}^{**}) \leftarrow C\_(x, y, \mathbf{t}^{*}),\ not\ C\_(x, y, \mathbf{f_a})$.

$M\_(x, y, \mathbf{t}^{**}) \leftarrow M\_(x, y, \mathbf{t}^{*}),\ not\ M\_(x, y, \mathbf{f_a})$.

(6) Programa de Consulta:

$ans(x, y) \leftarrow C\_(x, y, \mathbf{t}^{**})$.

Notar que en este programa tampoco se han incorporado las restricciones de negación. Existen cuatro soluciones de vecinos para el nodo S, osea, cuatro modelos estables para programa anterior:

$\mathcal{M}_1 = \{L\_(3, \mathbf{t}^{*}), aux(3), C\_(2, null, ta), M\_(2, 3, \mathbf{t}^{**}), M\_(3, 5, \mathbf{t}^{**}), C\_(2, null, \mathbf{t}^{*}),$
$C\_(1, t, \mathbf{t}^{**}), , C\_(3, e, \mathbf{t}^{**}), P\_(1, j, \mathbf{f_a}), C\_(2, null, \mathbf{t}^{**}), ans(1, t), ans(2, null),$
$ans(3, e)\}$

$\mathcal{M}_2 = \{C\_(2, null, \mathbf{t_a}), M\_(2, 3, \mathbf{t}^{**}), M\_(3, 5, \mathbf{t}^{**}), C\_(1, t, \mathbf{f_a}), C\_(3, e, \mathbf{t}^{**}), C\_(2, null, \mathbf{t}^{**}),$
$ans(2, null), ans(3, e)\}$

$\mathcal{M}_3 = \{M\_(2, 3, \mathbf{f_a}), M\_(3, 5, \mathbf{t}^{**}), C\_(1, t, \mathbf{t}^{**}), C\_(3, e, \mathbf{t}^{**}), P\_(1, j, \mathbf{f_a}), ans(1, t), ans(3, e)\}$

$\mathcal{M}_4 = \{M\_(2, 3, \mathbf{f_a}), M\_(3, 5, \mathbf{t}^{**}), C\_(1, t, \mathbf{f_a}), C\_(3, e, \mathbf{t}^{**}), ans(3, e)\}$

En consecuencia, existen cuatro instancias solución:

$D_{\mathcal{M}_1}(\mathtt{S}) = \{C(1, t),\ C(2, null),\ C(3, e), M(2, 3), M(3, 5)\}$,

$D_{\mathcal{M}_2}(\mathtt{S}) = \{C(2, null),\ C(3, e),\ M(2, 3),\ M(3, 5)\}$,

$D_{\mathcal{M}_3}(\mathtt{S}) = \{C(1, t),\ C(3, e),\ M(3, 5)\}$ y

$D_{\mathcal{M}_4}(\mathtt{S}) = \{C(3, e),\ M(3, 5)\}$.

Se busca la intersección de los cuatro para obtener las respuestas consistentes a $\mathcal{Q}$: $C(x, y)$. Se obtiene la solución $I(\mathtt{S}) = \{C(3, e)\}$. □

Está demostrado en [7] que para un sistema P2P β, $\mathtt{P} \in \mathcal{P}$, $\mathcal{N}(\mathtt{P}) = \{\mathtt{P}, \mathtt{P}_1, \ldots, \mathtt{P}_n\}$, $n \geq 0$, $D(\mathtt{P})$ una instancia para P, e $\mathcal{I}^* = \{I_1, \ldots, I_n\}$ instancias para $\mathtt{P}_1, \ldots, \mathtt{P}_n$, respectivamente. Si $\Sigma \cup IC$ es acíclico REF y cada una de las I_i es la intersección de las soluciones para el nodo $\mathtt{P}_i$, entonces las instancias de la forma $D_{\mathcal{M}}$, donde $\mathcal{M}$ es un modelo estable de programa $\Pi(\beta, \mathtt{P}, \mathcal{I}^*)$, son todas y las únicas soluciones para P.

Capítulo 4

Diseño e Implementación

En este capítulo se explican detalladamente los modelos utilizados en el proceso de desarrollo de la aplicación. Se muestra en qué consiste la aplicación, su funcionalidad, información que maneja, funciones, procesamientos, algoritmos, conexiones, etc. Se muestran los detalles de la implementación y los principales resultados obtenidos.

La solución propuesta es la implementación de una aplicación que genera los programas de reparación necesarios para obtener PCAs, los evalúa para finalmente deplegar respuestas consistentes a una consulta hecha por el usuario, expresada en DATALOG. La aplicación recibe el nombre de *"Peer Consistent Answers System"* y su acrónimo es *"PeerCA System"*. En adelante habrán de utilizarse estos nombres, indistintamente.

Desde el punto de vista del usuario, es posible identificar dos perfiles de usuario: administrador y usuario consultor. El primero es el encargado de especificar el sistema P2P. El segundo, solamente formula consultas y recibe las respuestas consistentes, programas y modelos. La Figura 4.1 proporciona una visión externa y global de PeerCA System, identificándose los principales flujos de datos a nivel de usuario.

Si bien, para este trabajo un nodo constituye una base de datos concreta, se da por hecho que la base de datos ya existe como tal, con un esquema, instancia y usuarios determinados de antemano, y con sus respectivos datos de conexión, como son nombre de usuario, clave de usuario, dirección IP, y nombre de la base de datos. En este sentido, la

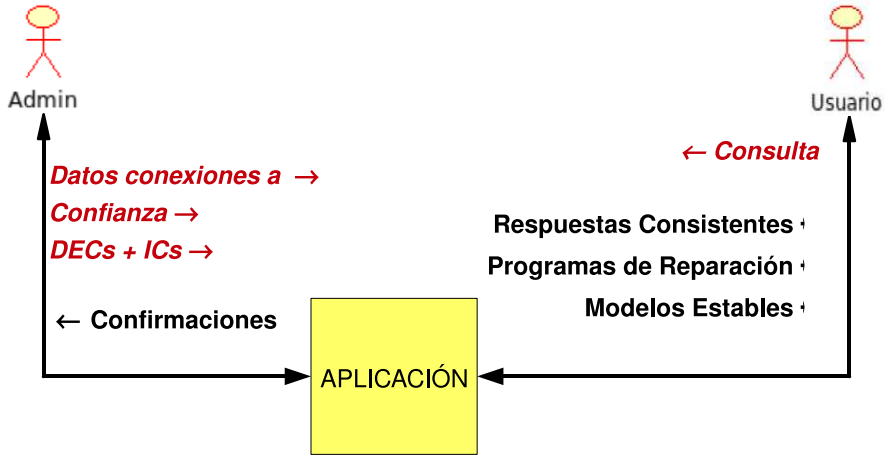

Figura 4.1: Escenario Preliminar PeerCA

aplicación web está pensada como una herramienta académica y didáctica, que permite la comunicación con cada uno de estos nodos, formulando consultas (o subconsultas) y recuperando instancias o parte de ellas, pero que en ningún caso busca funcionar como un SGBD.

Los elementos de información que la aplicación maneja son los datos de conexión a los nodos, las sentencias escritas en DATALOG para representar las consultas, las DECs e ICs de cada nodo, así como también gestiona las conexiones con los nodos (mediante drivers) y la comunicación directa con DLV System (mediante la línea de comandos), que es el motor de inferencias desarrollado por [16] para la semántica de modelos estables. La aplicación se encarga de ir realizando las comprobaciones que sean pertinentes para conseguir el objetivo general, que es el cómputo de los programas solución y la obtención de respuestas consistentes a consultas a partir de ellos.

El entorno de operación de la aplicación contempla, en general, la especificación del sistema P2P y la consulta como principal entrada, la generación del programa de reparación DATALOG, la comunicación con las bases de datos, la ejecución del programa en DLV System y la entrega de respuestas consistentes al usuario. La Figura 4.2 muestra este panorama.

El usuario consultor es quien actualiza las consultas en la aplicación, mientras que ésta retorna las respuestas consistentes, los programas de reparación y sus modelos esta-

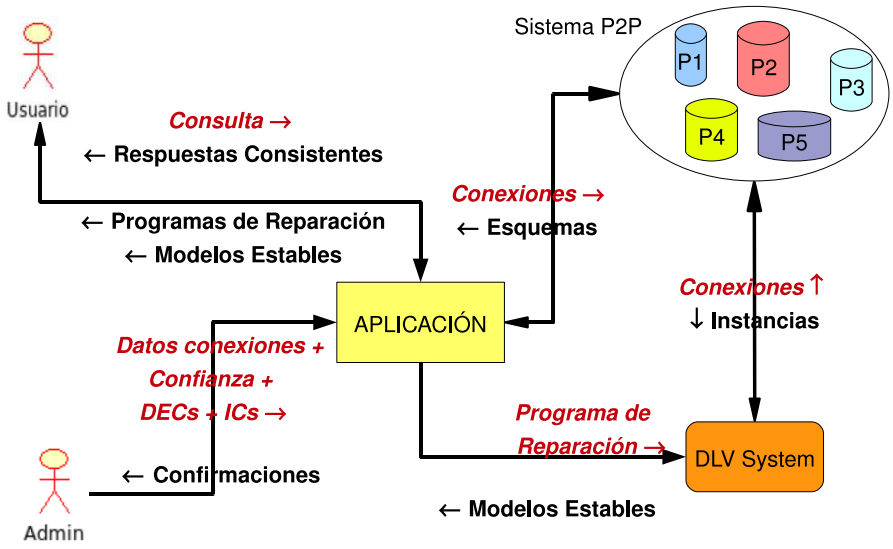

Figura 4.2: Entorno de Operación

bles. Por otro lado, el administrador es el encargado de suministrar los datos asociados a las conexiones de los nodos, las DECs, las relaciones de confianza y las ICs y recibe de la aplicación las confirmaciones de que los datos son válidos. Existe comunicación directa entre la aplicación y los nodos del sistema P2P, obteniendo la información de sus esquemas y metadatos (tablas, usuarios, permisos, etc.) por medio de conexiones. La aplicación se comunica con DLV System, proporcionando los programas solución escritos en DATALOG, mientras que DLV System retorna los modelos estables de los programas. Se puede apreciar además que DLV System mantiene comunicación directa con los nodos del sistema P2P, por medio de conexiones logra recuperar las instancias, que son incorporadas como hechos del programa.

4.1. Arquitectura de la Aplicación

La aplicación puede establecer conexiones con todas las bases de datos participantes del sistema P2P (1), aunque el usuario proporcione sólo sus propios datos de conexión. Se permite al administrador la actualización de los datos de conexión de un nodo dado (5), éstos son el nombre, usuario, clave y dirección IP. También es posible la actuali-

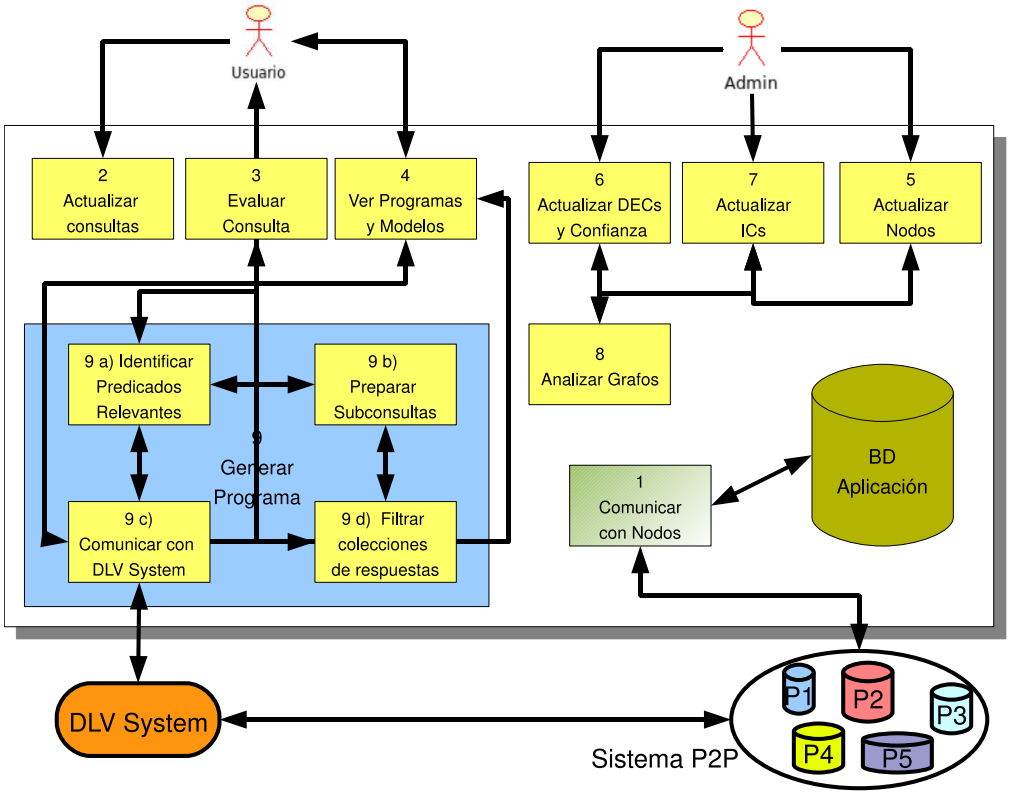

Figura 4.3: Arquitectura de PeerCA System

zación de las restricciones, tanto DECs (6) como ICs (7). La aplicación se encarga de validar que éstas cumplan con la sintaxis de DATALOG, de que las sentencias se encuentren expresadas en términos del esquema de los nodos involucrados y de comprobar que las restricciones de integridad validadas se redacten de acuerdo con las formas de una RIC (2.8), de una UIC (2.7) y de una NNC (2.5) descritas anteriormente. Se impide la presencia de ciclos los grafos de dependencias y de accesibilidad (8), al momento en que las restricciones son especificadas. Todos los módulos están relacionados con el módulo de Comunicar con Nodos, directa o indirectamente, pero se han omitido las flechas para efectos de mejor visualización.

La aplicación permite al usuario la actualización de consultas DATALOG para un nodo dado (2). Se comprueba que la regla cumpla con la forma del programa de consulta y comprobar que las consultas ingresadas se encuentran en función del esquema de un nodo determinado. Permite al usuario evaluar una de sus consultas (6) y desplegar

las respuestas consistentes a una consulta (de haberlas), computando los programas de reparación que sean necesarios. Si el usuario lo desea, puede desplegar por pantalla el(los) programa(s) de reparación generados y sus respectivos modelos estables (4). Este módulo ha sido subdividido en subprogramas, que en conjunto constituyen el programa de reparación.

1. **Identificar predicados relevantes:** La aplicación recibe la consulta formulada a un nodo. Conforme sus relaciones de confianza y sus DECs, la aplicación identifica cuáles son las reglas que se deben incorporar al programa y cuáles son los nodos que están implicados.

2. **Preparar subconsultas:** La consulta formulada a un nodo conlleva la formulación subconsultas a sus nodos vecinos. Este módulo se encarga de la identificación de los nodos accesibles y relevantes para la consulta en cuestión, analizando las DECs involucradas. Obtiene los predicados relevantes para poder formular las subconsultas derivadas.

3. **Comunicar con DLV System:** Encargado de transmitir las instrucciones para que DLV System las ejecute. Se debe procesar las salidas que DLV entrega una vez ejecutadas las instrucciones. DLV System permite establecer las conexiones con los nodos. Este módulo actúa de intermediario entre la aplicación y DLV System.

   DLV System: Aunque no es un módulo de la aplicación, este software es parte del entorno de operación. Es un motor de inferencias desarrollado por [16], para computar programas en lógica disyuntiva, implementando la semántica de modelos estables. Entre sus funcionalidades a explotar se encuentran:

   - Soporte para DATALOG.
   - Computación de programas en lógica disyuntiva.
   - Obtención de modelos estables.
   - Conexión a distintos motores de bases de datos.

La principal salida es el despliegue de respuestas consistentes a consultas en sistemas P2P posiblemente inconsistentes. Además el sistema deberá emitir las siguientes salidas: (1) Listado de nodos ordenados por nombre. (2) Listado de DECs. (3) Listado de ICs para un nodo determinado. (4) Listado de consultas para un usuario determinado. (5) Programas computados dada una consulta. (6) Respuestas consistentes para una consulta. (7) Listado de modelos estables para un programa determinado.

4.2. Especificando el Sistema P2P

A continuación se detallará cómo PeerCA System es poblado con la información necesaria de los nodos, DECs, ICs y consultas.

En este sistema los nodos constituyen una fuente para realizar consultas, pero en ningún caso para modificar sus datos. Todos los datos relativos a PeerCA System han sido concebidos en una base de datos propia. Los datos que esta base de datos debe almacenar son los siguientes: (a) Datos de conexión para cada nodo. Para cada nodo se debe almacenar un código identificador, el nombre de la base de datos, el nombre del usuario administrador, su clave de acceso, la dirección ip y el tipo de SGBD (DB2, SqlServer, Oracle, PostgreSql, MySql). (b) Para cada IC se debe almacenar la sentencia misma, el tipo de IC (referencial, universal o de no nulidad), y el nodo al que pertenece. (c) Para cada DEC se debe almacenar la sentencia misma, el tipo de DEC (referencial o universal), el nodo origen, el nodo destino y la relación de confianza. (d) Para cada consulta se debe almacenar la regla misma, el nodo al que se formuló y el usuario que lo hizo.

La base de datos de la aplicación se diseñó para administrar la información relativa al sistema P2P completo. Esta base de datos almacena los datos de conexión de todos los nodos que participan del sistema P2P. De esta forma, con estos datos efectua conexiones ocultas, simulando ser una llave maestra que permite acceder a la totalidad de esquemas de los nodos que se deban consultar, aún cuando el consultor no pueda acceder, por ser necesario a la hora de proveer datos al programa de reparación. Además, almacena

información de las DECs, las ICs, sus reglas asociadas, los datos de los usuarios que se conectan y sus consultas.

El manejo de conexiones se realiza en dos niveles: las líneas continuas reflejan las conexiones que dependen exclusivamente del usuario. En el caso del administrador, establece una conexión directa con la base de datos de la aplicación. En el caso del usuario consultor, establece una conexión con el nodo del cual es usuario. Para que pueda interactuar, el nodo debe pertenecer al sistema P2P especificado por el administrador. Las líneas punteadas reflejan las conexiones que establece la aplicación, sin que dependa de la intervención del usuario. La Figura 4.4 refleja la situación.

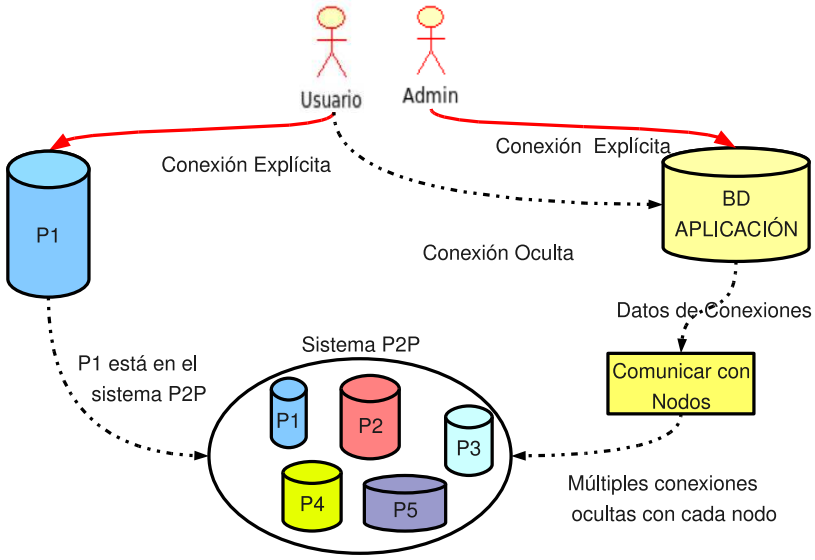

Figura 4.4: Conectar con Nodos

Dentro de la información necesaria para que la aplicación funcione está la actualización de los nodos del Sistema P2P. La Figura 4.5 detalla cómo se realiza esta actualización. Se busca el nodo en el Sistema P2P. Si es un nodo registrado, se ejecuta la actualización. Si se trata de un nuevo nodo, los datos de conexión para el nuevo Nodo P son comprobados. Se establece una conexión directamente con el nodo. Si resulta exitosa (3a), el nuevo Nodo P es agregado al sistema P2P y sus datos son almacenados en la base de datos de la aplicación. Los nodos también pueden ser eliminados y editados.

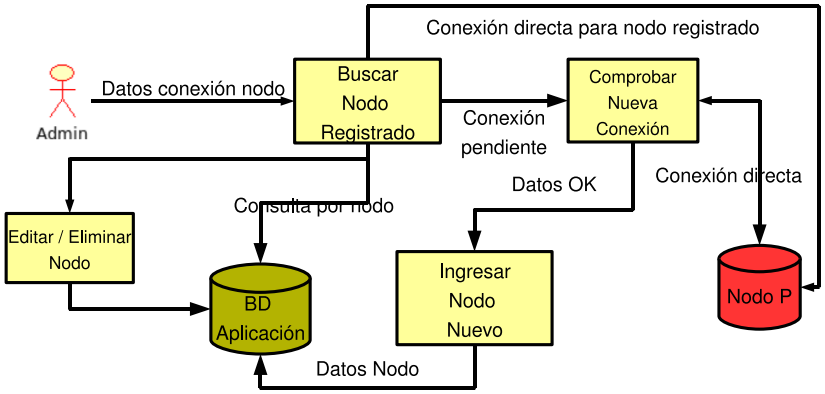

Figura 4.5: Actualizar Nodos

Otra de las tareas a detallar es la actualización de consultas, de ICs y de DECs. Cada una de estas clases obedece a una sintaxis común, ya que todas son expresadas en DATALOG. La aplicación debe realizar la comprobación tanto de que la sintaxis DATALOG sea correcta, como de que las sentencias estén en conformidad con el o los esquemas a los que pertenecen. Además, para generar el programa de reparación 11 es necesario definir reglas de mapeo que transformen una restricción en un conjunto de reglas. Para facilitar estas labores, se han diseñado las estructuras de datos pensando en la operatoria del programa solución y los mapeos de transformación. El razonamiento se basó en encontrar los elementos generales para cualquier tipo de sentencia DATALOG, que son los mostrados en la Figura 4.6.

La nomenclatura usada fue H para indicar la cabeza de la regla. Si la cabeza tiene disyunción, se denota por HD y si sólo tiene un átomo se dice que es cabeza simple HS. Los átomos de la cabeza son $A(H)$. B indica el cuerpo. Los átomos positivos del cuerpo son $A^+(B)$ y los negativos, $A^-(B)$. $VCB(H)$ denota a los built-ins de la cabeza, y los del cuerpo son $VCB(B)$, ya que su estructura obedece a Variable, Built-in y Comparando. Será utilizada en los algoritmos posteriores y en las reglas de mapeo para generar el programa de reparación. Las reglas que reparan los distintos tipos de restricciones se denotan con las iniciales de la restricción que reparan, anteponiendo la letra R. Por ejemplo, la sentencia $RRIC$ es la regla que repara una RIC.

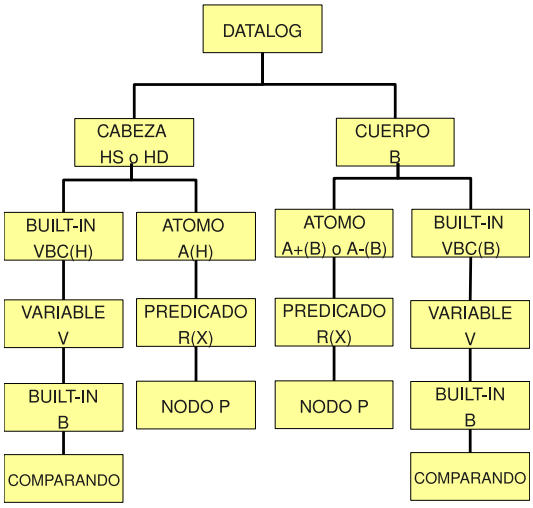

Figura 4.6: Estructura de una Sentencia DATALOG

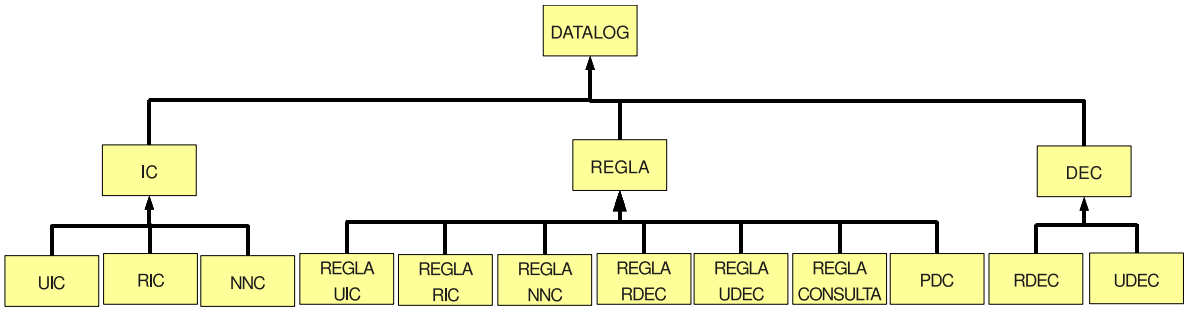

Figura 4.7: Clasificación de Sentencias DATALOG

Se ha indicado en la Figura 4.6 que cada *Atomo* se origina a partir de *Predicado*, el que a su vez proviene de un *Nodo*. Detallamos ahora las diferencias que dan origen a clasificaciones en el cuadro 4.1. Lo que diferencia a un tipo de sentencia de otra es la cantidad de ocurrencias que tienen sus componentes. Por ejemplo, una restricción de negación del programa PDC no tiene cabeza, mientras que la regla que repara una UIC tiene al menos dos átomos en la cabeza.

Es así como al ir encontrando las diferencias entre los tipos de sentencia DATALOG, pueden definirse clasificaciones, que se resumen en la Figura 4.7.

El usuario Consultor puede agregar y eliminar sus consultas. Al agregar una nueva consulta, se realiza una conexión directa con el nodo del cual es usuario, desplegando su esquema. La regla de la nueva consulta debe ser expresada en DATALOG y corresponder

Tipo	HD o HS	$A(H)$	$VBC(H)$	$A^{+}(B)$	$A^{-}(B)$	$VBC(B)$
UIC	HD	0,N	0,N	1,N	NO	NO
RIC	HS	1	NO	1	NO	NO
NNC	NO	NO	NO	1	NO	1
UDEC	HD	0,N	0,N	1,N	NO	NO
RDEC	HS	1	NO	1	NO	NO
RUIC	HD	1,N	NO	1,N	0,N	0,N
RRIC	HD	1,2	NO	1,N	0,N	NO
RNNC	HS	1	NO	1	NO	1
RUDEC	HD	1,N	NO	1,N	0,N	0,N
RRDEC	HD	1,2	NO	1,N	0,N	NO
CONSULTA	HS	1	NO	1,N	NO	0,N
PDC	NO	NO	NO	1	1	NO

Cuadro 4.1: Resumen de Características Sintácticas DATALOG.

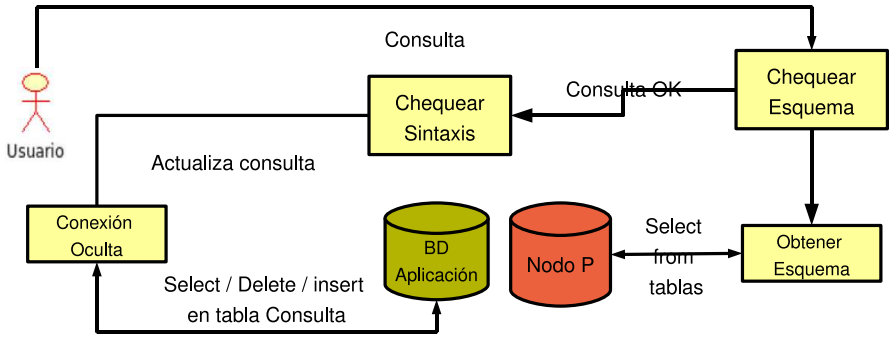

Figura 4.8: Actualizar Consultas

con el esquema que la aplicación despliega. El detalle del algoritmo que realiza estas comprobaciones se muestra en 1 y se basa en la fórmula 3.1. Por debajo, la aplicación almacena los datos actualizados en su base de datos local. Esta dinámica es mostrada en la Figura 4.8.

Algoritmo 1 Comprobar Consulta

Precondiciones: Predicados, Aridades, Cabeza, Cuerpo.

Postcondiciones: Consulta comprobada.

1: Vector P ← DividirEnPalabras(Cuerpo)

2: Cantidad ← ObtenerLargo(P)

3: **for** ($i = 0$; $i <$Cantidad; $i++$) **do**

```
4:  if (P[i] ∈ Predicados) then
5:    Posicion ← BuscarPosicion(Aridades, Predicados, P)
6:    Aridad ← Predicados[Posicion]
7:    for (j = 0; j <Aridad; j++) do
8:      Variables ← P[i]
9:      P[i] ← P[i+1]
10:     CompletarConsulta(P[i])
11:   end for
12: else if (P[i] == NOT) then
13:   P[i] ← P[i+1]
14:   if (P[i] ∈ Tablas) then
15:     Posicion ← BuscarPosicion(Predicados, Tablas, P)
16:     Aridad ← Predicados[Posicion]
17:     for (j = 0; j <Aridad; j++) do
18:       Variables ← P[i]
19:       CompletarConsulta(P[i])
20:     end for
21:   else
22:     return False
23:   end if
24: else if (EsVariable(P[i])) then
25:   Variables ← P[i]
26:   P[i] ← P[i+1]
27:   if (EsBVC(P[i])) then
28:     P[i] ← P[i+1]
29:     if (EsConstante(P[i])) then
30:       CompletarConsulta(P[i])
31:     else
32:       return False
33:     end if
34:   else
```

```
35:          return False
36:       end if
37:    else
38:       return False
39:    end if
40: end for
41: if (i == Cantidad ) then
42:    Vector P ← DividirEnPalabras(Cabeza)
43:    Cantidad ← ObtenerLargo(P)
44:    if (P[0] == ANS) then
45:       for (j = 1; j <Cantidad; j + +) do
46:          if ( not P[i] ∈ Variables[]) then
47:             return False
48:          end if
49:       end for
50:       if (j == Cantidad ) then
51:          return True
52:       else
53:          return False
54:       end if
55:    else
56:       return False
57:    end if
58: else
59:    return False
60: end if
```

La forma en que funciona la actualización de ICs se muestra en la Figura 4.9. Si el administrador agrega una IC, la aplicación establece conexiones ocultas, una con el nodo en sí y otra con la base de datos de la aplicación. La sentencia es analizada para comprobar su sintaxis, la presencia de ciclos en los grafos de dependencias y su

correspondencia con el esquema del nodo. De ser correctos, se incorpora al conjunto de ICs del sistema P2P, junto con generar y almacenar sus respectivas reglas.

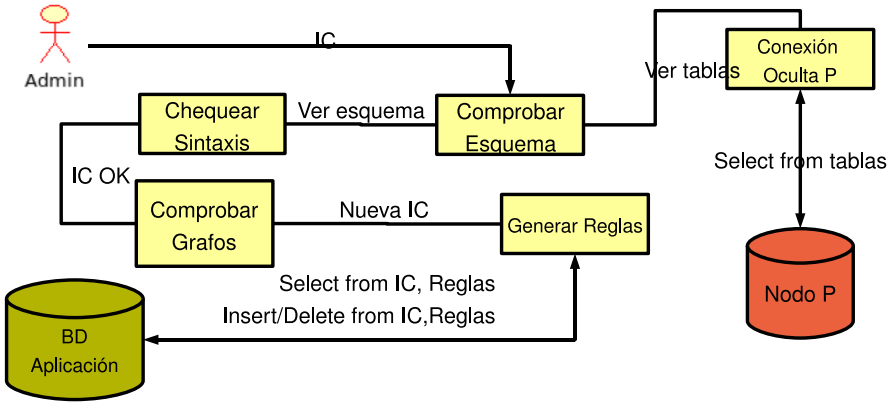

Figura 4.9: Actualizar ICs

El proceso a seguir para comprobar que las ICs son adecuadas para ser ingresadas se detalla en los Algoritmos 2 y 3.

Algoritmo 2 Comprobar UIC

Precondiciones: Predicados, Aridades, Cabeza, Cuerpo.

Postcondiciones: IC comprobada.

```
1:  Vector P ← DividirEnPalabras(Cuerpo)
2:  Cantidad ← ObtenerLargo(P)
3:  for (i = 0; i <Cantidad; i++) do
4:     if (P[i] ∈ Predicados) then
5:        Posicion ← BuscarPosicion(Aridades, Predicados, P)
6:        Aridad ← Predicados[Posicion]
7:        for (j = 0; j <Aridad; j++) do
8:           Variables ← P[i]
9:           P[i] ← P[i+1]
10:          CompletarUIC(P[i])
11:       end for
12:    else
```

```
13:        return False
14:     end if
15: end for
16: if (i == Cantidad ) then
17:     Vector P ← DividirEnPalabras(Cabeza)
18:     Cantidad ← ObtenerLargo(P)
19:     for (i = 0; i <Cantidad; i++) do
20:        if (P[i] ∈ Predicados) then
21:           Posicion ← BuscarPosicion(Aridades, Predicados, P)
22:           Aridad ← Predicados[Posicion]
23:           for (j = 0; j <Aridad; j++) do
24:              Variables ← P[i]
25:              P[i] ← P[i+1]
26:              CompletarUIC(P[i])
27:           end for
28:        else if (EsVariable(P[i])) then
29:           Variables ← P[i]
30:           P[i] ← P[i+1]
31:           if (EsBVC(P[i])) then
32:              P[i] ← P[i+1]
33:              if (EsConstante(P[i])) then
34:                 CompletarUIC(P[i])
35:              else
36:                 return False
37:              end if
38:           else
39:              return False
40:           end if
41:        else if (EsDisyuncion(P[i])) then
42:           P[i] ← P[i+1]
43:        else
```

```
44:          return False
45:       end if
46:    end for
47: end if
```

Algoritmo 3 Comprobar RIC

Precondiciones: Predicados, Aridades, Cabeza, Cuerpo.

Postcondiciones: RIC comprobada.

```
1: Vector P ← DividirEnPalabras(Cuerpo)
2: CantidadX ← ObtenerLargo(P)
3: if (P[0] ∈ Predicados) then
4:    Posicion ← BuscarPosicion(Aridades, Predicados, P)
5:    Aridad ← Predicados[Posicion]
6:    for (j = 1; j <Aridad; j++) do
7:       Variables ← P[i]
8:       P[i] ← P[i+1]
9:       Completar-RIC(P[i])
10:   end for
11:   Vector P ← DividirEnPalabras(Cabeza)
12:   Cantidad ← ObtenerLargo(P)
13:   for (j = 1; j <Cantidad; j++) do
14:      if (P[i] ∈ Variables[]) then
15:         NoExistenciales ← P[i]
16:      else if (not Existenciales ← P[i]) then
17:         return True
18:      end if
19:   end for
20:   CantidadY ← ObtenerLargo(Variables)
21:   CantidadZ ← ObtenerLargo(Existenciales)
22:   if (CantidadZ >0 ∧ CantidadX >0 ) then
23:      return True
```

24: **else**
25: **return** False
26: **end if**
27: **else**
28: **return** False
29: **end if**

Como las restricciones de integridad obedecen a tres tipos específicos de estructura, se han diseñado tres tipos de mapeo diferentes para la generación de las reglas. Para las restricciones de integridad universales, de la forma (2.7), las reglas de mapeo para transformarlas en una regla del programa solución son descritas en el Algoritmo 4.

Algoritmo 4 Reglas de mapeo para UICs

Precondiciones: : Una Restricción de Integridad Universal UIC especificada.

Postcondiciones: : Dos reglas $RUIC$ construídas a partir de UIC.

1: **for** (Para cada $A(H)$ de la forma $P(\bar{x}) \in UIC$) **do**
2: Incorporar a $RUIC_1$ y a $RUIC_2$ en $A(H)$ el átomo $P\_(\bar{x}, \mathbf{f_a})$
3: Incorporar a $RUIC_1$ y a $RUIC_2$ en $A(H)$ el símbolo "$\vee$"
4: Incorporar a $RUIC_1$ y a $RUIC_2$ en $A^+(B)$ el átomo $P\_(\bar{x}, \mathbf{t}^*)$
5: Incorporar a $RUIC_1$ y a $RUIC_2$ en $A^+(B)$ el símbolo ","
6: **end for**
7: **for** (Para cada $VBC(H)$ de la forma $\varphi \in UIC$) **do**
8: Incorporar a $RUIC_1$ y a $RUIC_2$ en $A^+(B)$ el nuevo VBC' con B negado.
9: **if** ($A^+(B)$ no es el último átomo) **then**
10: Incorporar a $RUIC_1$ y a $RUIC_2$ en $A^+(B)$ el símbolo ","
11: **end if**
12: **end for**
13: **for** (Cada $x \in UIC$ que aparezca dos veces) **do**
14: Incorporar a $RUIC_1$ y a $RUIC_2$ en $A^+(B)$ la expresión "$x \neq null$"
15: **if** ($A^+(B)$ no es el último átomo) **then**
16: Incorporar a $RUIC_1$ y a $RUIC_2$ en $A^+(B)$ el símbolo ","

17: **end if**
18: **end for**
19: **for** (Para cada $A^+(B)$ de la forma $Q(\bar{x})$) **do**
20: Incorporar a $RUIC_1$ y a $RUIC_2$ en $A(H)$ el átomo $Q\_(\bar{x}, \mathbf{t_a})$
21: **if** ($A^+(B)$ no es el último átomo) **then**
22: Incorporar a $RUIC_1$ y a $RUIC_2$ en $A(H)$ "$\vee$"
23: Incorporar a $RUIC_1$ y a $RUIC_2$ en $A^+(B)$ el símbolo ","
24: **end if**
25: Incorporar a $RUIC_1$ $A^+(B)$ el átomo $Q\_(\bar{x}, \mathbf{f_a})$
26: Incorporar a $RUIC_2$ $A^-(B)$ el átomo *not* $Q(\bar{x})$
27: **end for**
28: Incorporar a $RUIC_1$ y a $RUIC_2$ al final de $A(B)$ el símbolo"."

Para las restricciones de integridad referenciales, de la forma (2.8), las reglas de mapeo para transformarlas en una regla del programa solución son descritas en el Algoritmo 5.

Algoritmo 5 Reglas de mapeo para RICs

Precondiciones: : Una Restricción de Integridad Referencial $RRIC$ especificada.

Postcondiciones: : N reglas $RRIC$ construídas a partir de RIC.

1: Incorporar a $RRIC1$ y en $A(H)$ el átomo $P\_(\bar{x}, \mathbf{f_a})$
2: Incorporar a $RRIC1$ en $A(H)$ el átomo $Q\_(\bar{y}, n\bar{u}ll, \mathbf{t_a}))$
3: Incorporar a $RRIC1$ y en $A(B)^+$ el átomo $P\_(\bar{x}, \mathbf{t}^*)$
4: Incorporar a $RRIC1$ en $A^-(B)$ el predicado $AUX(\bar{y})$
5: Incorporar a la regla $RRIC2$ en $A(H)$ el predicado $AUX(\bar{y})$.
6: Incorporar a $RRIC2$ en $A(B)^+$ el átomo $Q(\bar{y}, n\bar{u}ll)$
7: Incorporar a $RRIC2$ en $A(B)^-$ el átomo $Q\_(\bar{y}, n\bar{u}ll, \mathbf{f_a})$
8: **for** (Para cada variable $x \in \bar{x}$) **do**
9: Incorporar a $RRIC1$ el $VBC(B)$ la expresión $x \neq null$
10: Incorporar a $RRIC2$ el $VBC(B)$ la expresión $x \neq null$
11: **end for**

12: **for** (Para cada variable $z \in \bar{z}$) **do**

13: Hacer una nueva regla $AUX(\bar{y}) \Leftarrow Q\_(\bar{y}, \bar{z}, \mathbf{t}^*)), not\ Q\_(\bar{y}, \bar{z}, \mathbf{f_a})), \bar{y} \neq null, z \neq null.$

14: **end for**

Dado que la correspondencia entre la relación de confianza y las DECs es de uno a uno, el objeto DEC incorpora estos dos elementos, conteniendo la sentencia lógica, la confianza menor o mayor, y haciendo referencia al Nodo Origen P y al Nodo Destino Q. La Figura 4.10 muestra el diseño de esta parte de la aplicación. Es bastante similar a la lógica seguida en la actualización de ICs, con la diferencia que las conexiones se realizan a los dos nodos, para comprobar los esquemas.

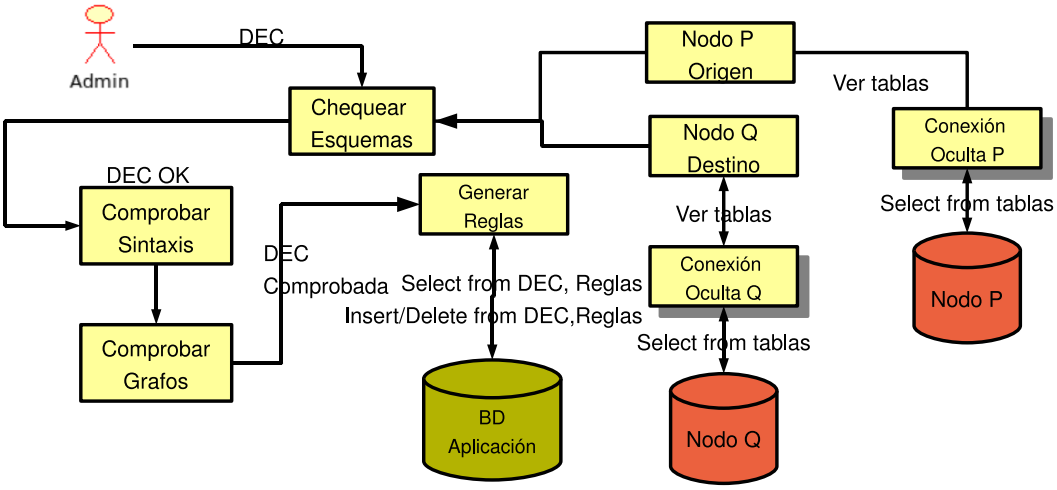

Figura 4.10: Actualizar DECs

Como se muestra en la Figura 4.10, al agregar una DEC a la aplicación, es necesario generar las reglas asociadas al programa de reparación. Los pasos necesarios para transformar cada tipo de UDEC en reglas para el programa, son descritos en los Algoritmos 6, 7 y refalg:rdecs.

Algoritmo 6 Algoritmo 1 Reglas de Mapeo para UDECs

Precondiciones: : Una Restricción de Intercambio de Datos Universal $UDEC(\mathtt{P}_1, \mathtt{P}_2)$ especificada.

Postcondiciones: : Una cantidad U de reglas $RUDEC$ construídas a partir de $UDEC$ $(\mathtt{P_1}, \mathtt{P_2})$. Con U dado por la cantidad d de literales que dependen en la restricción (aquéllos que se encuentran en el consecuente) elevado al cuadrado.

1: **for** (Para cada $A(H)$ de la forma $R(\bar{x}) \in UDEC(\mathtt{P_1}, \mathtt{P_2})$) **do**
2: **if** (Confianza es "menos") **then**
3: **if** $(R(\bar{x}) \in \mathtt{P_1})$ **then**
4: Incorporar a $RUDEC_1$ y a $RUDEC_2$ en $A(H)$ el átomo $R\_(\bar{x}, \mathbf{f_a})$
5: Incorporar a $RUDEC_1$ y a $RUDEC_2$ en $A(H)$ el símbolo "$\vee$"
6: Incorporar a $RUDEC_1$ y a $RUDEC_2$ en $A^+(B)$ el átomo $R\_(\bar{x}, \mathbf{t}^*)$
7: Incorporar a $RUDEC_1$ y a $RUDEC_2$ en $A^+(B)$ el símbolo ","
8: **else if** $(R(\bar{x}) \in \mathtt{P_2})$ **then**
9: Incorporar a $RUDEC_1$ y a $RUDEC_2$ en $A^+(B)$ el átomo $R\_(\bar{x}, \mathbf{t}^*)$
10: Incorporar a $RUDEC_1$ y a $RUDEC_2$ en $A^+(B)$ el símbolo ","
11: **end if**
12: **else if** (Confianza es "igual") **then**
13: Incorporar a $RUDEC_1$ y a $RUDEC_2$ en $A(H)$ el átomo $R\_(\bar{x}, \mathbf{f_a})$
14: Incorporar a $RUDEC_1$ y a $RUDEC_2$ en $A(H)$ el símbolo "$\vee$"
15: Incorporar a $RUDEC_1$ y a $RUDEC_2$ en $A^+(B)$ el átomo $R\_(\bar{x}, \mathbf{t}^*)$
16: Incorporar a $RUDEC_1$ y a $RUDEC_2$ en $A^+(B)$ el símbolo ","
17: **end if**
18: **end for**
19: **for** (Para cada $VBC(H)$ de la forma $\varphi \in UDEC$) **do**
20: Incorporar a $RUDEC_1$ y a $RUDEC_2$ en $A^+(B)$ el nuevo VBC' con B negado.
21: **if** ($A^+(B)$ no es el último átomo) **then**
22: Incorporar a $RUDEC_1$ y a $RUDEC_2$ en $A^+(B)$ el símbolo ","
23: **end if**
24: **end for**
25: **for** (Cada $x \in UDEC$ que aparezca dos veces) **do**
26: Incorporar a $RUDEC_1$ y a $RUDEC_2$ en $A^+(B)$ la expresión "$x \neq null$"
27: **if** ($A^+(B)$ no es el último átomo) **then**

28: Incorporar a $RUDEC_1$ y a $RUDEC_2$ en $A^+(B)$ el símbolo ","
29: **end if**
30: **end for**

Algoritmo 7 Algoritmo 2 Reglas de Mapeo para UDECs

1: **for** (Para cada $A(B)$ de la forma $Q(\bar{x}) \in UDEC(\mathtt{P}_1, \mathtt{P}_2)$) **do**
2: **if** (Confianza es "menos") **then**
3: **if** ($Q(\bar{x}) \in \mathtt{P}_1$) **then**
4: Incorporar a $RUDEC_1$ y a $RUDEC_2$ en $A(H)$ el átomo $Q\_(\bar{x}, \mathbf{t_a})$
5: Incorporar a $RUDEC_1$ y a $RUDEC_2$ en $A(H)$ el símbolo "$\vee$"
6: Incorporar a $RUDEC_1$ $A^+(B)$ el átomo $Q\_(\bar{x}, \mathbf{t}^*)$
7: Incorporar a $RUDEC_2$ $A^-(B)$ el átomo *not* $Q(\bar{x})$
8: Incorporar a $RUDEC_1$ y a $RUDEC_2$ en $A^+(B)$ el símbolo ","
9: **else if** ($Q(\bar{x}) \in \mathtt{P}_2$) **then**
10: Incorporar a $RUDEC_1$ $A^+(B)$ el átomo $Q\_(\bar{x}, \mathbf{t}^*)$
11: Incorporar a $RUDEC_2$ $A^-(B)$ el átomo *not* $Q(\bar{x})$
12: Incorporar a $RUDEC_1$ y a $RUDEC_2$ en $A^+(B)$ el símbolo ","
13: **end if**
14: **else if** (Confianza es "igual") **then**
15: Incorporar a $RUDEC_1$ y a $RUDEC_2$ en $A(H)$ el átomo $Q\_(\bar{x}, \mathbf{t_a})$
16: Incorporar a $RUDEC_1$ y a $RUDEC_2$ en $A(H)$ el símbolo "$\vee$"
17: Incorporar a $RUDEC_1$ y a $RUDEC_2$ en $A^+(B)$ el átomo $Q\_(\bar{x}, \mathbf{t}^*)$
18: Incorporar a $RUDEC_2$ $A^-(B)$ el átomo *not* $Q(\bar{x})$
19: Incorporar a $RUDEC_1$ y a $RUDEC_2$ en $A^+(B)$ el símbolo ","
20: **end if**
21: **end for**
22: Incorporar a $RUDEC_1$ y a $RUDEC_2$ al final de $A(B)$ el símbolo"."

Para las restricciones de intercambio de datos referenciales, de la forma (2.8), las reglas de mapeo para transformarlas en una regla del programa solución son descritas en el Algoritmo 8.

Algoritmo 8 Reglas de Mapeo para RDECs

Precondiciones: : Una Restricción de Intercambio de Datos Referencial $RDEC$ especificada.

Postcondiciones: : N reglas $RRDEC$ construídas a partir de $RDEC(\mathtt{P}_1, \mathtt{P}_2)$

1: **if** (Confianza es "igual") **then**
2: Incorporar a $RRDEC1$ y en $A(H)$ el átomo $R\_(\bar{x}, \mathbf{f_a})$
3: Incorporar a $RRDEC1$ en $A(H)$ el átomo $Q\_(\bar{x}', \bar{null}, \mathbf{t_a}))$
4: Incorporar a $RRDEC1$ y en $A(B)^+$ el átomo $R\_(\bar{x}, \mathbf{t}^*)$
5: Incorporar a $RRDEC1$ en $A^-(B)$ el predicado $AUX(\bar{x}')$
6: **else if** (Confianza es "menos") **then**
7: **if** $(R \in \mathtt{P}_1)$ **then**
8: Incorporar a $RRDEC1$ y en $A(H)$ el átomo $R\_(\bar{x}, \mathbf{f_a})$
9: Incorporar a $RRDEC1$ y en $A(B)^+$ el átomo $R\_(\bar{x}, \mathbf{t}^*)$
10: Incorporar a $RRDEC1$ en $A^-(B)$ el predicado $AUX(\bar{x}')$
11: **else if** $(Q \in \mathtt{P}_1)$ **then**
12: Incorporar a $RRDEC1$ en $A(H)$ el átomo $Q\_(\bar{x}', \bar{null}, \mathbf{t_a}))$
13: Incorporar a $RRDEC1$ y en $A(B)^+$ el átomo $R\_(\bar{x}, \mathbf{t}^*)$
14: Incorporar a $RRDEC1$ en $A^-(B)$ el predicado $AUX(\bar{x}')$
15: **end if**
16: **end if**
17: Incorporar a la regla $RRDEC2$ en $A(H)$ el predicado $AUX(\bar{x}')$.
18: Incorporar a $RRDEC2$ en $A(B)^+$ el átomo $Q(\bar{x}', \bar{null})$
19: Incorporar a $RRDEC2$ en $A(B)^-$ el átomo $Q\_(\bar{x}', \bar{null}, \mathbf{f_a})$
20: **for** (Para cada variable $x \in \bar{x}$) **do**
21: Incorporar a $RRDEC1$ el $VBC(B)$ la expresión $x \neq null$
22: Incorporar a $RRDEC2$ el $VBC(B)$ la expresión $x \neq null$
23: **end for**
24: **for** (Para cada variable $y \in \bar{y}$) **do**
25: Hacer una nueva regla $AUX(\bar{x}') \Leftarrow Q\_(\bar{x}', \bar{y}, \mathbf{t}^*)), not\ Q\_(\bar{x}', \bar{y}, \mathbf{f_a})), \bar{x}' \neq null, y \neq null.$
26: **end for**

Esta función es crucial para el desarrollo del programa solución. Se analizarán los tres tipos de grafos, cada uno con algoritmos y programas DATALOG propios.

Se hace uso del siguiente programa DATALOG para comprobar la existencia de ciclos en el grafo de accesibilidad construído a partir de Σ. Para ello, de deben irse incorporando las relaciones de confianza, una por una. Para representar el grafo, se empleará el predicado artificial `VECINO(X,Y)`, que representa un vértice donde `X` es el nodo origen e `Y` es el nodo destino. No es necesario conocer el valor para la relación de confianza (si es menos o igual). La información se obtiene a partir de los datos que ingresa el administrador a la hora de especificar el sistema P2P.

Sin embargo, para efectos de ejecutar el programa DATALOG los nombres de los nodos pierden relevancia, mientras se utilicen variables con iniciales en mayúsculas y constantes con iniciales en minúsculas (o valores numéricos).

El administrador es quien va ingresando los datos de las DECs y la aplicación se encarga de incorporarlas en el programa internamente con el predicado `Vecino` y de ir comprobando la presencia de ciclos a medida que se ingresan una a una. De no representar un ciclo, la nueva DEC es almacenada en la base de datos de la aplicación. La primera regla del programa ha sido ingresada a modo de ejemplo. El detalle de este programa está en el Algoritmo

```
Vecino(a,b).
Accesible(X,Y) :- Vecino(X,Y).
Accesible(X,Y) :- Vecino(X,Z), Accesible(Z,Y).
Ans(X) :- Accesible(a,X).
```

Precondiciones: Especificar DECs una por una.

Postcondiciones: DECs ingresadas al sistema P2P. Σ acíclico.

1: **for** (Cada nueva DEC especificada en Σ(`a,b`)) **do**
2: Agregar el hecho `Vecino(a,b).` al programa DATALOG.
3: Modificar la consulta `Ans(X)` : $-$`Accesible(a,X)`.
4: Esto es reemplazar `a` por el valor que tome la primera variable en la nueva DEC.

Hecho	Consulta	Respuestas	¿Ingresar?
Vecino(a, b)	Ans(X) : −Accesible(a, X).	{b}	SÍ
Vecino(b, c)	Ans(X) : −Accesible(b, X).	{c}	SÍ
Vecino(c, d)	Ans(X) : −Accesible(c, X).	{d}	SÍ
Vecino(d, a)	Ans(X) : −Accesible(d, X).	{a, b, c, d}	NO

Cuadro 4.2: Resultados de Casos de Prueba del Algoritmo

5: Ejecutar el programa DATALOG.
6: **if** (a ∈ {Ans(X)}) **then**
7: **print** "Existen ciclos". Esta DEC no será ingresada al sistema.
8: **else**
9: CONECTAR-NODO(*bdaplicacion*)
10: INGRESAR-DEC(DEC ∪ Σ(a, b))
11: **end if**
12: **end for**

Se han evaluado los resultados con diferentes casos de prueba, descritos en el cuadro 4.2.

Con el siguiente programa se puede identificar la presencia de ciclos mediante restricciones referenciales, ya sean RICs o RDECs, o su conjunto. La idea es que a medida que se vayan ingresando las restricciones de integridad no se vayan generando respuestas para Ans(X). Se han empleado los predicados RC(X, Y) para indicar restricciones referenciales del tipo RDEC o RIC, mientras que el predicado UC(X, Y) representa a las restricciones universales del tipo UDEC o UIC en un grafo de dependencias. Como en el ejemplo anterior, X representa el nombre del predicado en el antecedente (vértice origen) e Y es el nombre del predicado en el consecuente (vértice destino). Si se generan respuestas, quiere decir que se encontraron ciclos mediante restricciones referenciales. Al ir incorporando restricciones, como hechos al programa, una por una, se debe ir cambiando la consulta, en la última regla del programa. Por ejemplo, si se tiene un nuevo hecho UC(a, b), la consulta quedaría Ans(X) : −Depende(a, X), RC(X, Y), Depende(Y, b). Lo mismo sucede si se ingresa el hecho RC(a, b). Este programa DATALOG no genera respuestas si se producen ciclos mediante restricciones universales. Sólo detecta ciclos

Hecho	Consulta	Respuestas	¿Ingresar?
RC(n, d).	Ans(X) : −Depende(n, X), RC(X, Y), Depende(Y, d).	∅	SÍ
UC(s, d).	Ans(X) : −Depende(s, X), RC(X, Y), Depende(Y, d).	∅	SÍ
UC(d, p).	Ans(X) : −Depende(d, X), RC(X, Y), Depende(Y, p).	∅	SÍ
UC(p, s).	Ans(X) : −Depende(p, X), RC(X, Y), Depende(Y, s).	∅	SÍ
UC(a, n).	Ans(X) : −Depende(a, X), RC(X, Y), Depende(Y, n).	∅	SÍ
UC(p, a)	Ans(X) : −Depende(p, X), RC(X, Y), Depende(Y, a).	{n}	NO
RC(d, a)	Ans(X) : −Depende(d, X), RC(X, Y), Depende(Y, a).	{n, d}	NO

Cuadro 4.3: Datos de Prueba para el Programa de Detección de Ciclos Referenciales en Dependencias.

RIC o ciclos REF.

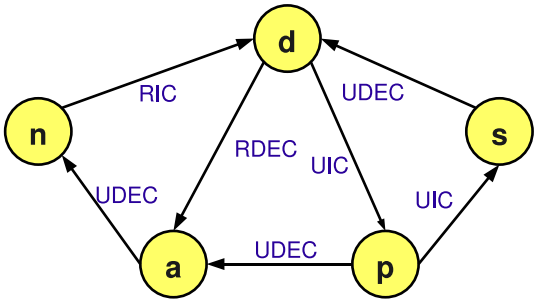

Figura 4.11: Grafo de Dependencias

Se han evaluado los resultados con diferentes casos de prueba. Los hechos corresponden al grafo mostrado en la Figura 4.11. Para cada restricción, ya sea RDEC(X, Y) o RIC(X, Y), se incorpora el hecho RC(X, Y). Para cada restricción UDEC(X, Y) o UIC(X, Y), se incorpora el hecho UC. Los resultados son descritos en el cuadro 4.3.

4.3. Generando el Programa de Reparación

PeerCA System genera el o los programas de reparación necesarios en el momento en que el usuario consultor dispone una consulta para ser evaluada. Este proceso de detalla en la Figura 4.12.

Se ha mencionado anteriormente la importancia de DLV System para la aplicación. Constituye el motor de inferencias que entrega los modelos estables de los programas de reparación que PeerCA System genera. Por un lado, la aplicación accede a los nodos

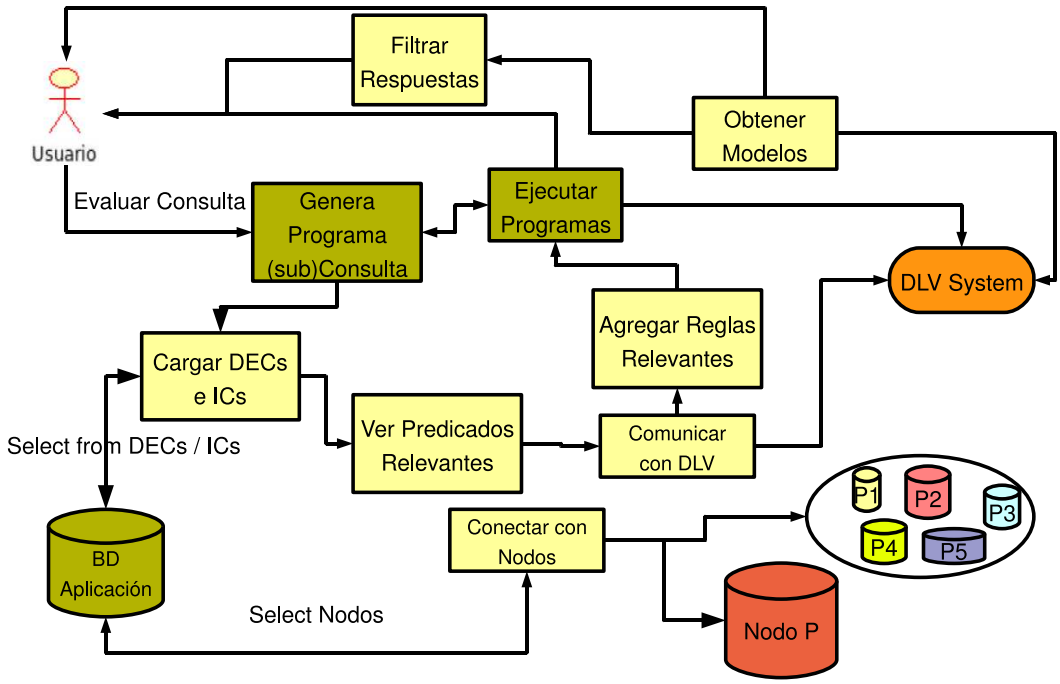

Figura 4.12: Evaluar Consulta

del sistema P2P por medio de conexiones, pero DLV System también necesita acceder a los nodos. Esto lo hace mediante una sentencia especial llamada *import*. Se incorpora como una regla más al programa de reparación de tal manera que DLV System incopora *internamente* las tuplas como hechos del programa de reparación. Los hechos nunca son cargados en memoria durante la ejecución del programa de reparación, sino hasta cuando son procesados como respuestas consistentes a partir de los modelos estables que DLV entrega..

```
#import (nodo, "usuario","clave","select * from Tabla", Predicado).
```

El módulo encargado de comunicarse con PeerCA System lo hace a través de la línea de comandos, llamando a DLV directamente:

```
$dlv nombreArchivoReparacion.dlv
```

Internamente, el módulo de comunicación con DLV se encarga de procesar la salida de la línea de comandos, para utilizar los modelos estables que DLV entrega y por otro

lado, se encarga de invocar la ejecución de los programas llamandolo por medio de la línea de comandos.

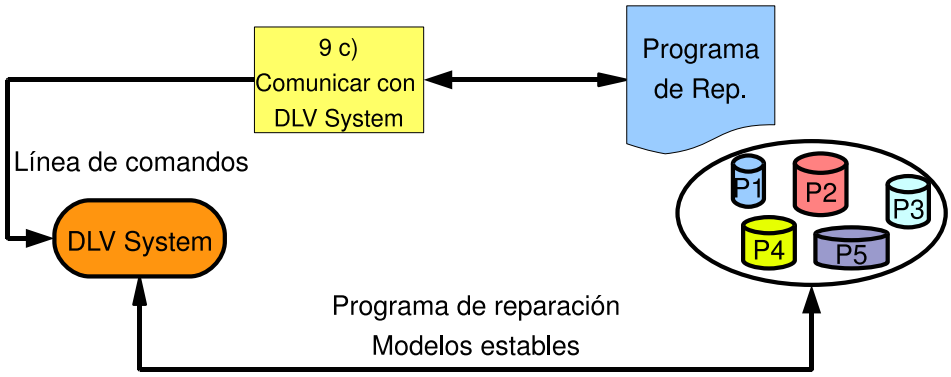

Figura 4.13: PeerCA System y DLV

Se tiene la conexión que el usuario proporciona al iniciar sesión, y otra conexión con la base de datos de la aplicación. Esta última permite acceder a los datos de las DECs, ICs para establecer si es posible aplicar el programa solución. De serlo, se identifican los predicados relevantes para la consulta. Se incorporarán sólo las reglas generadas previamente que estén relacionadas con los predicados relevantes para la consulta. Se debe acceder a cada nodo involucrado, gracias a los datos de conexión almacenados, y así obtener la información de los esquemas. SUBCONS es el módulo que obtiene subconsultas. PROG es el programa de reparación. FILTRARESP es el módulo que obtiene las pcas a partir de modelos estables, intersectando aquellos átomos que aparezcan en todos los modelos. Con todos estos elementos se construye el programa solución, que se detalla en el Algoritmo 9. Notar que todos las funciones tienen un caracter recursivo. Las llamadas se comunican entre sí intercambiando subconsultas y respuestas PCAs.

Algoritmo 9 Proceso de Evaluación de una Consulta

Precondiciones: Consultas actualizadas.

Postcondiciones: Respuestas consistentes a una consulta formulada a un nodo P.

1: *usuario* ⇐INICIAR-SESION (*nombre*, *clave*, P, *direccionip*)

2: **if** $(usuario = consultor)$ **then**
3: CONECTAR-NODO $(bdaplicacion)$
4: EVALUAR-CONSULTA $(\mathtt{P}, \mathcal{Q})$
5: $cicloref \Leftarrow (\Sigma(\mathtt{P}), IC(\mathtt{P}))$
6: **for** (Cada $\mathtt{P_i} \in \beta$) **do**
7: CONECTAR-NODO $(\mathtt{P_i})$
8: **end for**
9: COMUNICAR-CON-DLV()
10: **if** $(\mathcal{N}(\mathtt{P}) = \{\mathtt{P}\})$ **then**
11: $pcas \Leftarrow \mathtt{P}|R_i \Leftarrow$PROG $(\mathtt{P}, \beta, \emptyset, \emptyset, \mathcal{Q})$, con $R_i \in \mathcal{Q}$
12: **else**
13: $pcas \Leftarrow$FILTRARESP(PROG($\mathtt{P}, \beta, \bigcup_{i=1}^{n}$FILTRARESP(PROG($\mathcal{N}(\mathtt{P})$, SUBCONS($\mathcal{Q}$))), $\mathcal{Q}$))
14: **return** $pcas$
15: **else**
16: PRINT("No es posible obtener respuestas consistentes")
17: **end if**
18: CERRARSESION()
19: **else**
20: **print** "Error: imposible realizar la conexión."
21: **end if**

Se identifican dos casos en el que se aplica el programa solución: el caso base es cuando no quedan vecinos por recorrer, el caso recursivo recorre todos los vecinos de un nodo, a partir de P, y va filtrando las respuestas de los modelos y a la vez generando nuevas subconsultas, para seguir aplicando más programas solución.

Para una consulta, hemos visto en el capítulo 2 que no todos los predicados de un nodo se utilizan para generar programas de reparación. La noción de predicado relevante dice relación con los componentes conectados en un grafo de dependencias, construído a partir de restricciones donde los predicados de la consulta participen.

Para caracterizar el grafo de dependencias, se utiliza el predicado artificial para indicar las restricciones (de cualquier tipo) $\mathtt{REST}(\mathtt{X}, \mathtt{Y})$, donde X es el nombre del predicado

que es vértice origen, mientras que `Y` es el nombre del predicado que es vértice destino. También se utilizan las estructuras de datos presentadas en 4.1. En este caso, el sistema P2P se encuentra especificado previamente. Esto es explicado en el Algoritmo 10.

Algoritmo 10 Generación del Grafo de Dependencias

Precondiciones: Sistema P2P especificado.

Postcondiciones: Grafo de dependencias

1: **for** (Cada $\mathtt{P}_i \in \beta$) **do**
2: $ics[*] \Leftarrow$ OBTENER-ICS($\mathtt{P}_i$)
3: $decs[*] \Leftarrow$OBTENER-DECS($\mathtt{P}_i$)
4: $sentencias[*] \Leftarrow ic[*] \cup decs[*]$
5: Cargar estructuras de datos.
6: **for** (Cada $sentencias[i]$) **do**
7: **for** (Cada $A_i(B)$) **do**
8: **if** ($(sentencias[i]\ es\ HS)$ **then**
9: **if** ($VBC(B)$) **then**
10: Crear el hecho $\mathtt{REST}(\mathtt{A_i}(\mathtt{B}), \mathtt{A_i}(\mathtt{B}))$
11: **else**
12: Crear el hecho $\mathtt{REST}(\mathtt{A_i}(\mathtt{B}), \mathtt{A_1}(\mathtt{H}))$
13: **end if**
14: **else**
15: **for** (Cada $A_j(H)$) **do**
16: Crear el hecho $\mathtt{REST}(\mathtt{A_i}(\mathtt{B}), \mathtt{A_j}(\mathtt{H}))$
17: **end for**
18: **for** (Cada $VBC(B)$) **do**
19: Crear el hecho $\mathtt{REST}(\mathtt{A_i}(\mathtt{B}), \mathtt{A_i}(\mathtt{B}))$
20: **end for**
21: **end if**
22: **end for**
23: **end for**
24: **end for**

La aplicación invoca el Algoritmo 11 cuando un usuario evalúa una consulta y se debe generar el programa solución e identificar las subconsultas. La aplicación identifica los predicados presentes en la consulta y a partir de ellos, genera las consultas para el algoritmo.

Algoritmo 11 Predicados Relevantes para una Consulta

Precondiciones: Grafo de dependencias generado.

Postcondiciones: Predicados relevantes para $\mathcal{Q}$

1: Usuario evalúa consulta $\mathcal{Q}$ en nodo `P`
2: **for** (Cada predicado $R_i \in \mathcal{Q}$) **do**
3: En el programa DATALOG, reemplazar `t` por R_i.
4: Ejecutar el programa DATALOG.
5: $relevantes[*] \Leftarrow relevantes[*] \cup \{\texttt{Ans(X)}\}$
6: **end for**

El siguiente programa entrega todos los predicados que son componentes conectados para un predicado `t` dado (ver última regla).

```
Conectado(X,Y) :- REST(X,Y).
Conectado(X,Y) :- REST(Y,X).
Conectado(X,Y) :- REST(X,Z), Conectado(Z,Y).
Ans(X) :- Conectado(t,X).
```

Los siguientes son los hechos que deben incorporarse al programa anterior:

```
REST(a,cuo). REST(cuo,pre). REST(pre,a).
REST(ju,ga). REST(ju,ju).
REST(ju,em). REST(ju,per). REST(tram,per).
```

Se han evaluado los resultados con diferentes casos de prueba, descritos en el cuadro 4.4.

Ejemplo 18 El siguiente es el código de uno de los programas de reparación que la aplicación genera, para el ejemplo 15. □

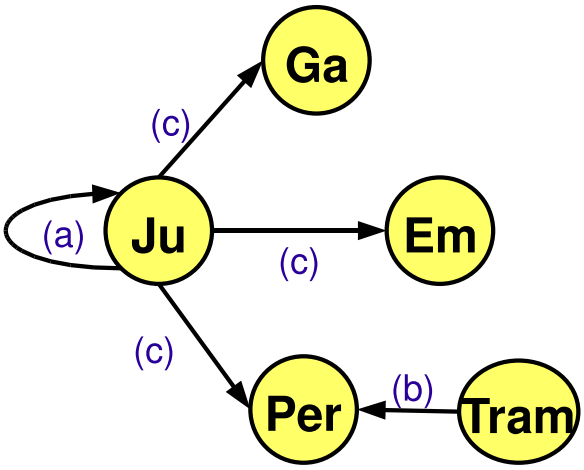

Figura 4.14: Grafo de Dependencias Ejemplo 6)

Consulta	Respuestas
$\texttt{Ans(X)} : -\texttt{Conectado(h, X)}$	$\emptyset$
$\texttt{Ans(X)} : -\texttt{Conectado(cuo, X)}$	$\{\texttt{a}, \texttt{cuo}, \texttt{pre}\}$
$\texttt{Ans(X)} : -\texttt{Conectado(a, X)}$	$\{\texttt{a}, \texttt{cuo}, \texttt{pre}\}$
$\texttt{Ans(X)} : -\texttt{Conectado(ga, X)}$	$\{\texttt{ju}, \texttt{em}, \texttt{per}, \texttt{tram}\}$
$\texttt{Ans(X)} : -\texttt{Conectado(per, X)}$	$\{\texttt{ju}, \texttt{ga}, \texttt{em}, \texttt{tram}\}$
$\texttt{Ans(X)} : -\texttt{Conectado(tram, X)}$	$\{\texttt{ju}, \texttt{ga}, \texttt{em}, \texttt{per}\}$

Cuadro 4.4: Casos de Prueba para el Algoritmo

```
#import (nodoi, "db2inst1","db2inst1","select * from L", L).
L_(X,fa) :- L_(X,ts),  not aux2(X), X != "null".
aux2(X) :- P(X,"null"), not P_(X,"null",fa), X != "null".
aux2(X) :- P_(X,Y,ts), not P_(X,Y,fa), X != "null", Y != "null".
P_(X1,X2,ts) :- P(X1,X2).
P_(X1,X2,ts) :- P_(X1,X2,ta).
:- P_(X1,X2,ta), P_(X1,X2,fa).
P_(X1,X2,tss) :- P_(X1,X2,ts), not P_(X1,X2,fa).
L_(X1,ts) :- L(X1).
L_(X1,ts) :- L_(X1,ta).
:- L_(X1,ta), L_(X1,fa).
L_(X1,tss) :- L_(X1,ts), not L_(X1,fa).
ANSO(X1) :- L_(X1,tss).
P(1,"j").
P(2,"m").
```

```
P(3,"e").
```

4.4. Interfaz

En esta Sección se exponen las principales interfaces. Se finaliza el capítulo mostrando los resultados obtenidos al aplicar el programa de reparación que despliega PCAs. Para mostrar el funcionamiento de la aplicación, en el caso de la interfaz del usuario consultor, se ha generado el ejemplo 17.

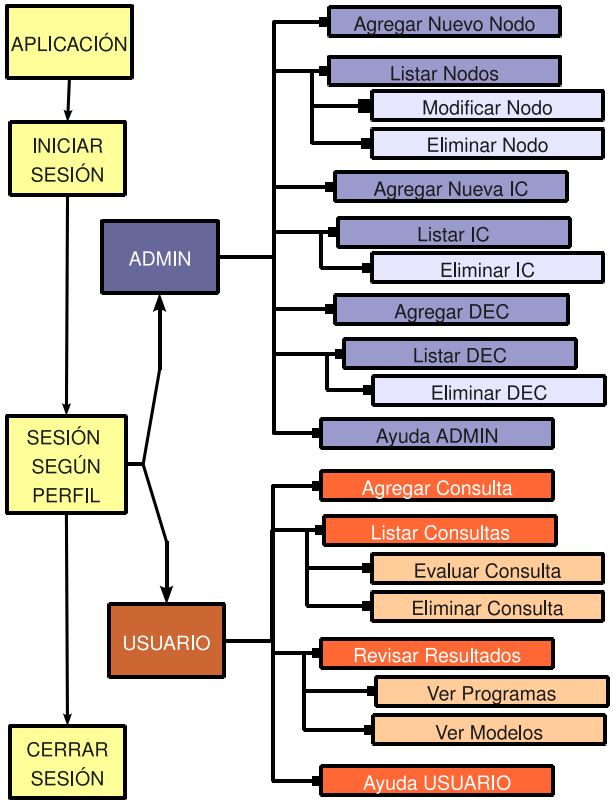

Figura 4.15: Menú de Navegación

PeerCA es una aplicación web, diseñada para dos tipos de sesiones: una para el administrador del sistema P2P y otra para los usuarios que formulan consultas. La estructura del menú navegacional se muestra en la Figura 4.15.

Con respecto al acceso de los usuarios, el administrador de PeerCA System accede como todos los usuarios, proporcionando los datos de conexión para acceder a un nodo. La diferencia es que al ingresar, accede a la base de datos de la aplicación, que es la que almacena toda la especificación del sistema P2P. El usuario accede al sistema

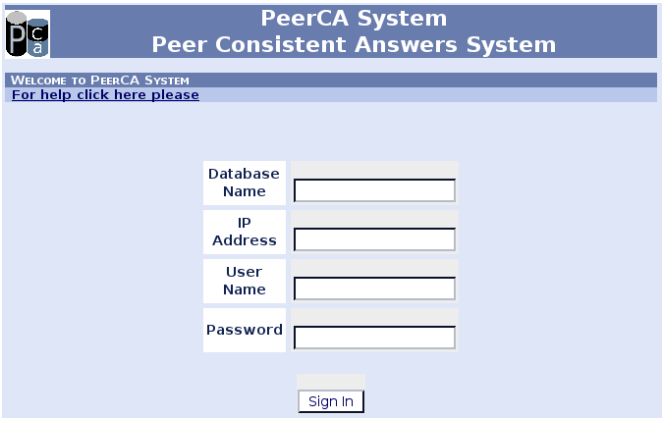

Figura 4.16: Iniciando Sesión

ingresando los datos de conexión a un nodo. Esto se muestra en la Figura 4.16.

Dependiendo de cuál haya sido el nodo al que se conectó, la aplicación muestra ya sea la interfaz para el administrador (ver Figura 4.17) o bien la interfaz para el consultor (ver Figura 4.22).

El Administrador

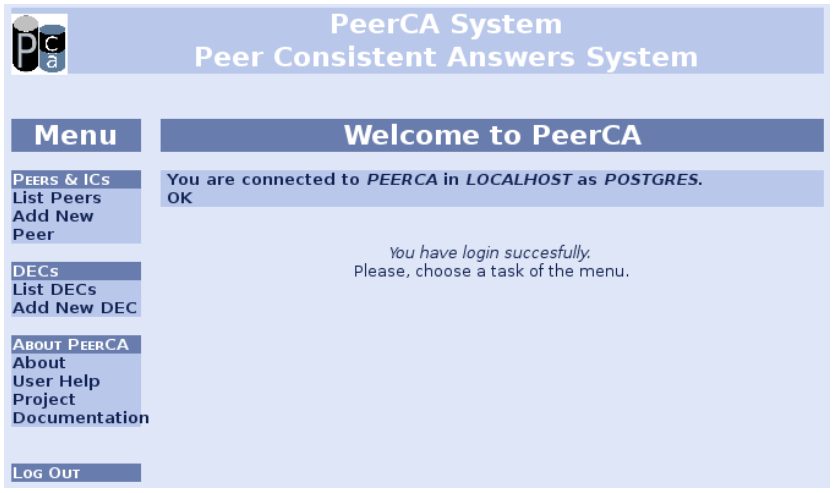

Figura 4.17: Interfaz Administrador

Algunas de las opciones que la aplicación otorga el administrador son mostradas en las siguientes Figuras. Actualización de nodos (Figura 4.19). Actualización de DECs

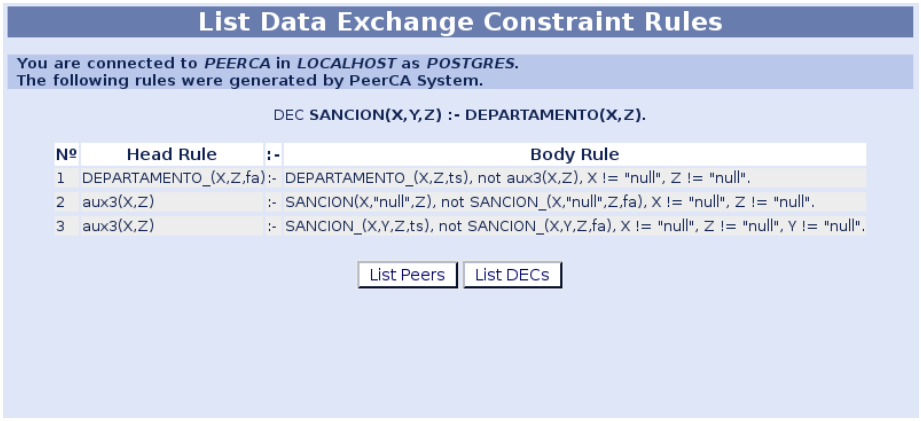

Figura 4.18: IListar Reglas de una Restricción

(Figura 4.20). Listar todas las DECs (Figura 4.21)

Al modificar los datos de conexión de un nodo, se establece una conexión de prueba para comprobar si se tendrá el acceso real al nodo.

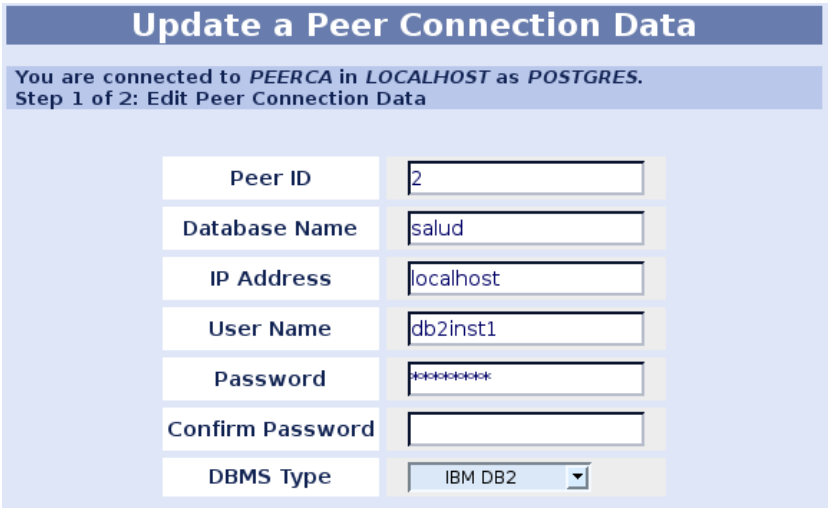

Figura 4.19: Modificar Nodo

El Usuario

Las siguientes son las funcionalidades que el usuario consultor puede realizar en PeerCA System.

El usuario consultor tiene las opciones para listar consultar, ingresar una consulta,

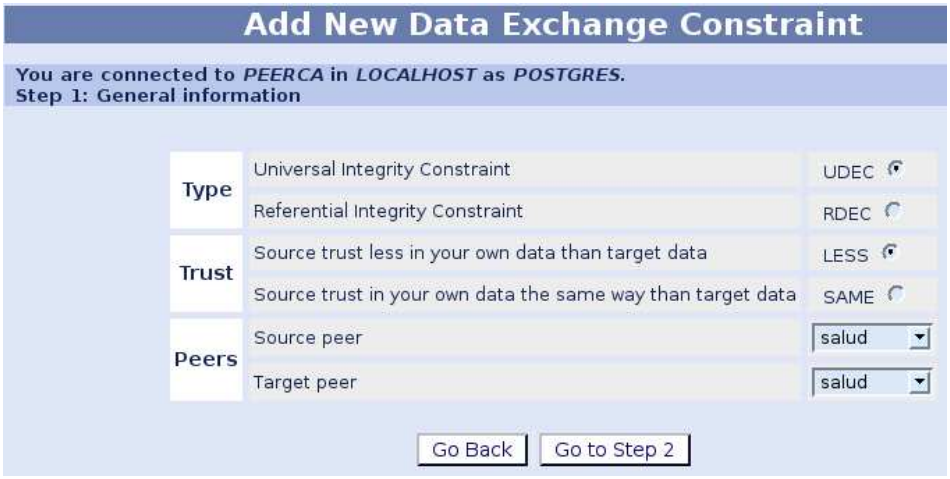

Figura 4.20: Ingresar una DEC

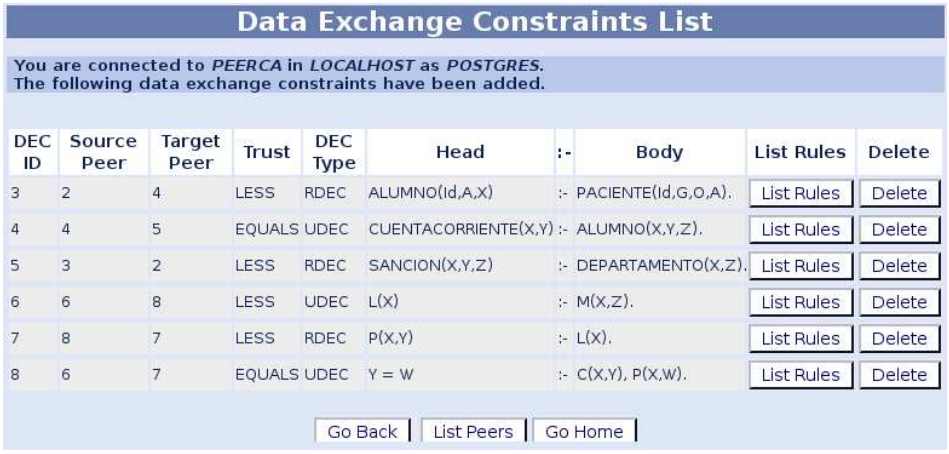

Figura 4.21: Listar DECs

evaluar una consulta, listar programas de reparación, ver modelos estables y obtener PCAs. Notar que las salidas corresponden a la ejecución del caso de prueba descrito en el capítulo 17. (Ver Ejemplo 15.

El usuario ingresa una consulta siempre respetando el esquema al cual se conectó. Además, la consulta debe responder a la estructura de la fórmula (3.1).

El usuario almacena las consultas en la aplicación, pudiendo listarlas, eliminarlas o evaluarlas.

Al evaluar una consulta, la aplicación muestra las respuestas consistentes, de haberlas, o un mensaje de error.

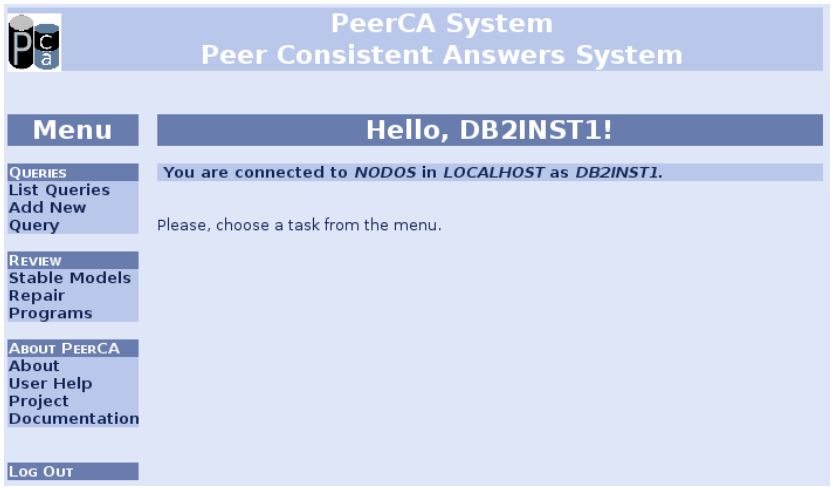

Figura 4.22: Interfaz Consultor

Al evaluar una consulta, se genera una lista de programas de reparación.

De la lista de programas generados, es posible revisar cada uno de ellos.

Al ejecutar un programa de reparaci ón se obtienen sus modelos estables, de haberlos.

A partir de la generación de los modelos estables, es posible intersectarlos para obtener las respuestas consistentes a la consulta.

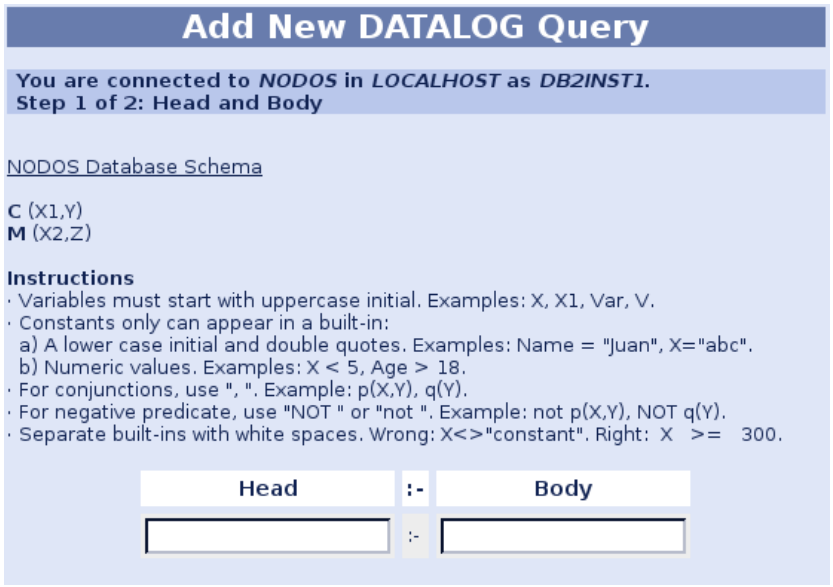

Figura 4.23: Ingresar una Consulta

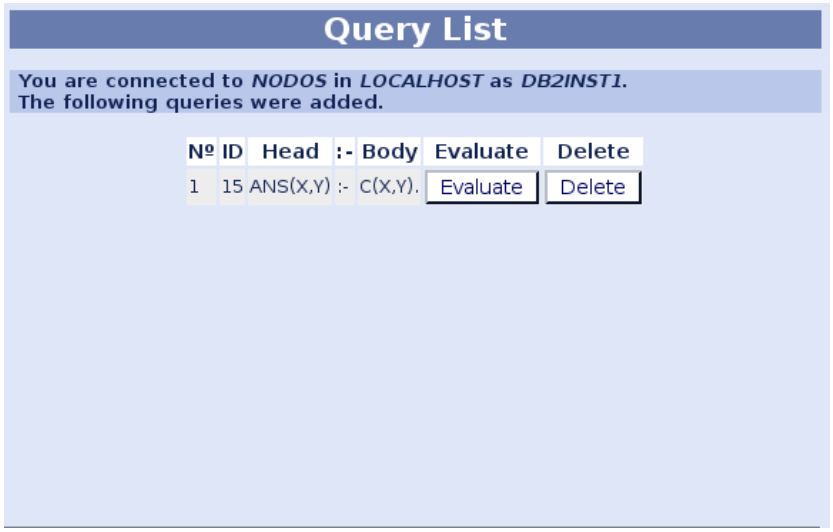

Figura 4.24: Listar Consultas

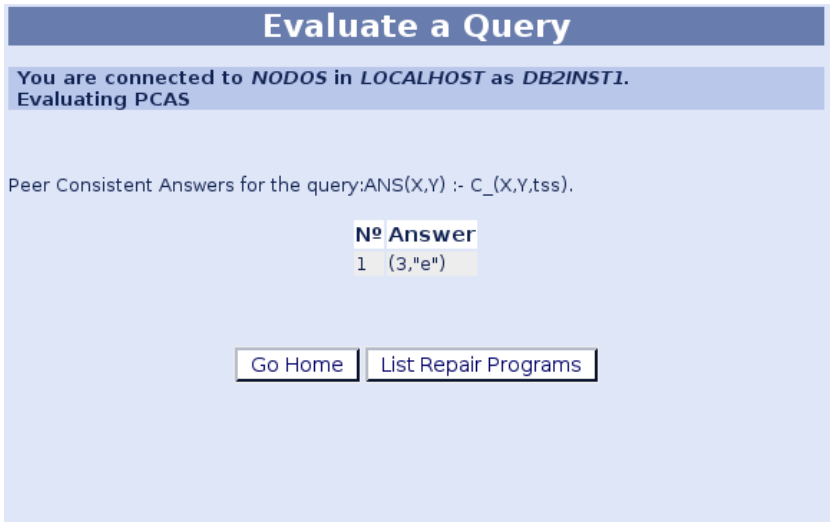

Figura 4.25: Evaluar una consulta

Figura 4.26: Listar Programas

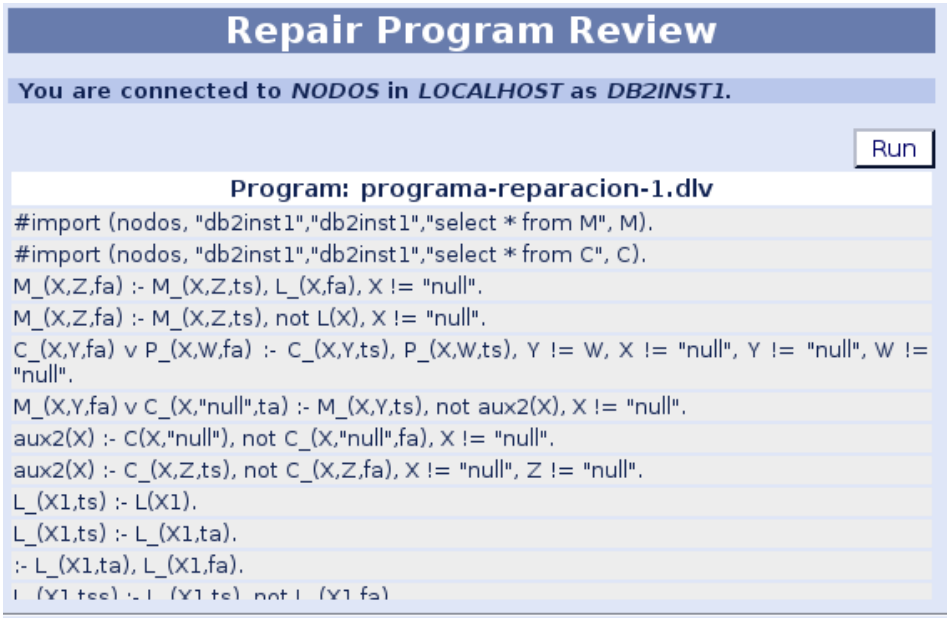

Figura 4.27: Revisar Programa

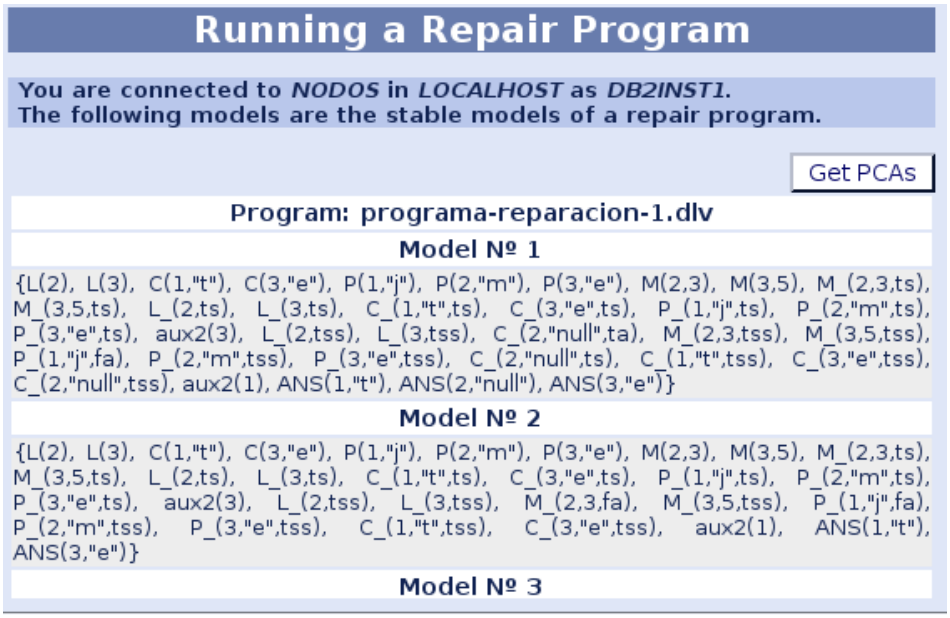

Figura 4.28: Ejecutar un Programa de Reparación

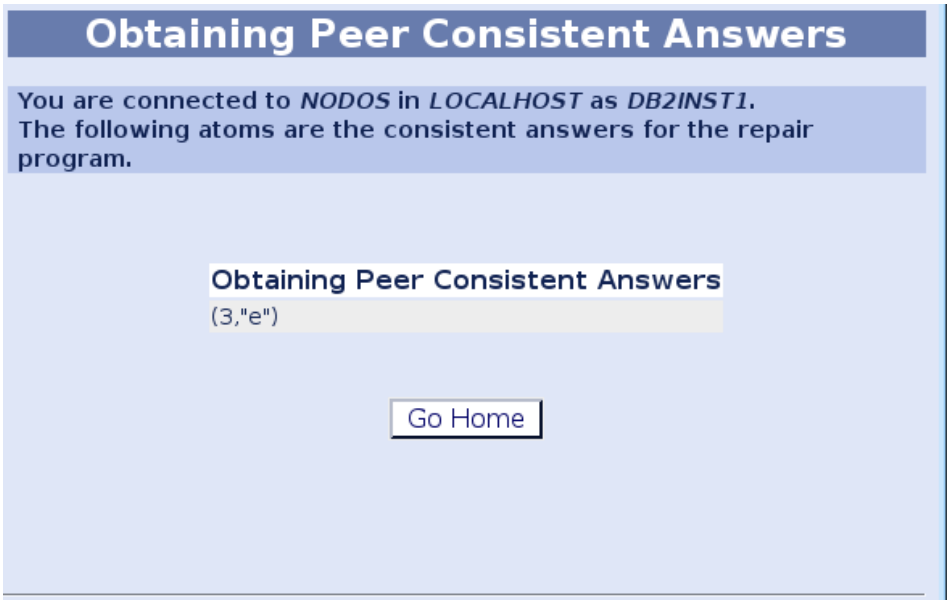

Figura 4.29: Obtener Respuestas Consistentes

Capítulo 5

Conclusiones

La principal idea para finalizar, es que a partir de la semántica de modelos estables, es posible obtener reparaciones de nodos inconsistentes con respecto a sus restricciones de intercambio de datos y de integridad. Es posible además cómputar DLPs de reparación no tan sólo para obtener instancias solución que sean consistentes, sino también para obtener respuestas consistentes a consultas, utilizando la misma metodología. Para determinar que una respuesta sea consistente se utiliza el razonamiento cauteloso. Este razonamiento indica que una respuesta a una consulta es consistente si pertenece a todos los modelos estables de un programa Π.

De acuerdo con lo mostrado hasta ahora, la obtención de respuestas consistentes a consultas en sistemas P2P conlleva a señalar que no es posible computar un DLP de reparación si el grafo de dependencias del nodo a reparar es cíclico RIC. La consistencia de un nodo está determinada por la satisfacción de sus propias ICs y de las DECs que tiene con sus vecinos, y de éstos con sus vecinos, y así, sucesivamente. Ésta es una noción recursiva. Encontrar las respuestas consistentes a una consulta hecha a un nodo `P` implica la formulación de subconsultas y la obtención de PCAs en otros nodos accesibles a `P`. Obtener las PCAs desde un nodo equivale a encontrar la instancia solución para el nodo, la que se construye a partir de las PCAs de cada nodo vecino. Es posible obtener una solución $I(\mathtt{P})$ para un nodo `P` por medio del cómputo de un DLP de reparación,

llamado programa solución, construído en función de $D(\mathtt{P})$, $\Sigma(\mathtt{P})$ e $IC(\mathtt{P})$, las relaciones de confianza de P con sus vecinos, y la unión de las intersecciones de las soluciones de cada nodo vecino.

No es necesario computar el programa solución para obtener PCAs de un nodo P con $\Sigma(\mathtt{P}) \cup IC(\mathtt{P}) = \emptyset$. Simplemente, en este caso se evalúa la subconsulta directamente. No es posible computar el programa solución si el grafo de accesibilidad del sistema P2P es cíclico, así como tampoco lo es cuando el grafo de dependencias construído a partir de un nodo P y de $\Sigma(\mathtt{P}) \cup IC(\mathtt{P})$ es cíclico REF.

Existen tres tipos de ICs, las UICs, las RICs y las NNCs, y para cada una de ellas existen tanto un tipo de reglas que incorporar al programa de reparación como una estructura sintáctica. Asimismo existen dos tipos de DECs, las UDECs y las RDECs, y en ambas intervienen las relaciones de confianza al momento de reparar.

Si se piensa en el sistema P2P como un árbol cuya raíz es el nodo que recibe la consulta del usuario, y cuyas ramas están determinadas por las relaciones de confianza y las DECs, intuitivamente se puede esquematizar el problema en tres casos: (a) No se computa un programa solución, dado que el nodo es un nodo hoja (no existe más intercambio con otros nodos), por lo tanto la entrega de PCAs se traduce a evaluar la subconsulta en la instancia misma. (b) El programa solución es de un nodo intermedio, que actúa como emisor y receptor de PCAs. El nodo colecta las PCAs que recibe de los otros y las incorpora como hechos. Lo que el nodo debe entregar es la intersección de todas sus soluciones. Depende de las PCAs que entreguen otros programas y debe retornar PCAs a los programas que lo invocaron. (c) El nodo es el nodo raiz, que sólo recibe PCAs, las incorpora como hechos y debe responder a la consulta original hecha por el usuario con CQAs, entregando reparaciones locales.

El problema de la obtención de respuestas consistentes a nodos en sistemas P2P, que son inconsistentes con respecto a sus DECs e ICs, se puede solucionar por medio de programas de reparación con la semántica de modelos estables.

Es posible aplicar al programa de reparación para la obtención de PCAs, desarrollada por [7], la misma optimización que la aplicada en los programas de [10] para la

obtención de CQAs, en el sentido de incorporar al programa sólo los predicados relevantes para la consulta y de centrar el programa en la consulta, y no en las instancias completas de los nodos.

Es posible desarrollar una aplicación web que implemente el programa de reparación presentado en [7] para la obtención de respuestas consistentes a consultas de primer orden a nodos en sistemas P2P inconsistentes con respecto a sus ICs y DECs, aplicando la noción de predicado relevante, de manera que se ha optimizado la ejecución de los programas de reparación presentadas en [10], en cuanto a hacer las consultas y subconsultas extrictamente necesarias y en cuanto a requerir datos de los nodos involucrados en una consulta y no a todo el sistema P2P como la teoría propone.

Bibliografía

[1] Marcelo Arenas, Leopoldo Bertossi y Jan Chomicki. "Consistent Query Answering in Inconsistent Databases". In *Proceeding 18 ACM Symposium on Principle of Database Systems (PODS)*. ACM Press, 1999, pp 68-79.

[2] Marcelo Arenas, Leopoldo Bertossi y Jan Chomicki. "Logics for Emerging Applications of Databases". Chapter *Query Answering in Inconsistent Databases*. Springer, 2003, pp 43-83.

[3] Leopoldo Bertossi y Loreto Bravo. "Consistent Query Answering Under Inclusion Dependencies". In *14º Annual IBM Centers for Advanced Studies Conference (CASCON 2004)*. 2004, pp 202-216.

[4] Leopoldo Bertossi y Loreto Bravo. "Query Answering in Peer-to-Peer Data Exchange Systems". In *Proceedings of the International Workshop on Peer-to-Peer Computing and DataBases*. In *Current Trends in Database Technology*. Springer LNCS 3268, 2004, pp 478-485.

[5] Leopoldo Bertossi y Loreto Bravo. "Semantically Correct Query Answers in the Presence of Null Values". In *Proceedins of the EDBT WS on Inconsistency and Incompleteness in Databases (IIDB 06)*. Springer LNCS 4254, 2006, pp 336-357.

[6] Loreto Bravo. "Handling Inconsistency in Databases and Data Integration Systems". Ph.D. Thesis, Carleton University, 2007.

[7] Leopoldo Bertossi y Loreto Bravo. "The Semantics of Consistency and Trust in Peer Data Exchange Systems". In *Proceedins of the 14th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR'07).* Springer LNCS 4790, 2007, pp 107-122.

[8] John W. Lloyd. "Foundations of Logic Programming". Springer Verlag, 1987, ISBN 3-540-18199-7.

[9] Maarten H. Van Emden y Robert A. Kowalski. "The Semantics of Predicate Logic as a Programming Language". ACM, 23, 1976, pp 733-742.

[10] Leopoldo Bertossi y Mónica Caniupán. "Optimizing Repair Programs for Consistent Query Answering". In *Proceeding 25 International Conference of Chilean Computer Science Society (SCCC 2005).* IEEE Computer Science Society Press, 2005, pp 3-12.

[11] Mónica Caniupán. "Optimizing and Implementing Repair Programs for Consistent Query Answering in Databases". Ph.D. Thesis, Carleton University, 2007.

[12] Thomas Eiter, T., Gottlob, G. and Mannila, H. "Disjunctive Datalog". ACM Transactions on Database Systems, 1997, 22(3), pp 364–418.

[13] Michael Gelfond y Vladimir Lifschitz. "The Stable Model Semantics for Logic Programming". In *Proceedings of the International Conference on Logic Programming.* MIT Press, 1988, pp 1070-1080.

[14] Michael Gelfond y Vladimir Lifschitz. "Classical Negation in Logic Programs and Disjunctive Databases". New Generation Computing, 1991, 9(3/4), pp 365-386.

[15] John Grant y Jack Minker. "The impact of logic programming". Communications of the ACM, 1992, 35(3), pp 66-81.

[16] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, Francesco Scarcello. "The DLV System for Knowledge Representation

and Reasoning". Communications of the ACM, ACM Press, New York, 2006, 7(3), pp 499-562. Disponible en *http://www.dbai.tuwien.ac.at/proj/dlv*.

[17] Mónica Caniupán y Leopoldo Bertossi. "The Consistency Extractor System: Querying Inconsistent Databases Using Answer Set Programs". In *Proceedings of the Scalable Uncertainty Management Conference (SUM 2007)*. Springer LNCS 4772, 2007, pp 74-88.

[18] Mark Levene y George Loizou. "Null Inclusion Dependencies in Relational Databases". Information and Computation, 1997, 136(2), pp 67-108.

[19] Teodor C. Przymusinski. "Stable Semantics for Disjunctive Programs". New Generation Computing, 1991, 9(3/4), pp 401-424.

[20] Ray Reiter. "On Closed World Databases". Logic and Databases. H. Gallaire and J. Minker, Eds. Plenum, 1978, pp 56-76.

[21] Jan Chomicki, Jerzy Marcinkowski, y Slawomir Staworko. "Hippo: a System for Computing Consistent Answers to a Class of SQL Queries". In *Proceedings Advances in Database Technology - EDBT 2004, 9th International Conference on Extending Database Technology*. Lecture Notes in Computer Science 2992 Springer, 2004, ISBN 3-540-21200-0, pp 841-844.

[22] Leopoldo Bertossi y Alexander Celle. "Querying Inconsistent Databases: Algorithms and Implementation". In *6th International Conference on Rules and Objects in Databases (DOOD'2000)*. Lecture Notes in Artificial Intelligence 1861, Springer 2000, pp 942-956.

[23] Ariel Fuxman, Diego Fuxman y Renée Miller "ConQuer: a System for Efficient Querying Over Inconsistent Databases". In *Proceedings of the 31st International Conference on Very Large Data Bases*. ACM 2005, 2005, pp 1354-1357.