Diseño de la metadata semántica para un orquestador de operaciones encapsuladas

Una Ontología para inferir cambios a nivel de Servicios

Informe de Proyecto de Grado 2004

Santiago Dalchiele, Ana Díaz, Daniel Gerchicov, Alvaro Rettich
Proyecto de Grado – Grupo PGMETSEM – 2004

Tutores: Regina Motz, Fernando Rodriguez.

Usuario Responsable: Jorge Abin

Carrera de Ingeniería en Computación

Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay

pgmetsem@fing.edu.uy

Resumen—

Este proyecto se enmarca en el contexto de la integración de sistemas. Donde las funcionalidades de los sistemas legados se encapsulan como servicios. La integración hará uso exhaustivo de esos servicios, utilizándolos para componer procesos de negocios, los cuales se expresan mediante una coreografía de software.

Debido a que estos servicios pueden ser desarrollados por agentes externos a la organización que los utiliza, la evolución de los mismos presenta un grave problema para el mantenimiento del sistema en sí. En este punto, se hace necesaria la existencia de un componente capaz de detectar los cambios en los servicios, y de ser posible, modificar cada coreografía afectada en forma automática.

Esto se logra dándole mayor inteligencia al sistema. Por un lado se debe diseñar la metadata que describa los servicios desde un punto de vista conceptual, y no solamente funcional. Otro punto a considerar, es la semántica que nos proporciona el grafo de ejecución inscripto en las coreografías.

La solución implementada y presentada en este trabajo se basa en estándares abiertos. Esto es vital debido a que este proyecto se enmarca en otro de mayor alcance y favorece a que el mismo evolucione, en el contexto tan dinámico que impone esta materia.

Palabras claves—Thalia, Ontología, OWL-S, SOA.

Índice General

I INTRODUCCIÓN

I - 1 CONTEXTO

El mundo globalizado trae aparejado la necesidad de una mayor integración de las operaciones entre las empresas. Como consecuencia directa se presenta requerimientos de integración de sistemas, que se aplican tanto para empresas individuales como entre ellas.

Las tecnologías que soporten esta integración deben poder interactuar con sistemas altamente heterogéneos e integrarlos a través de la Web. La actual respuesta tecnológica a estas necesidades de integración se basa fundamentalmente en los Web Services.

Es así que las funcionalidades de los sistemas legados se encapsulan como servicios, los cuales se exponen como Web Services. La integración hará uso exhaustivo de esos servicios, utilizándolos para componer procesos de negocios, de aquí en más macro operaciones, los cuales se expresan mediante una coreografía de software.

Este trabajo se enmarca en la línea de investigación que se inicia con la tesis de maestría del Ing. Jorge Abin [2], y se continúa con dos proyectos de grado de la carrera de Ingeniería en Computación de la Universidad de la República, con el proyecto SICO [1] y con los trabajos realizados en el Laboratorio de Integración de Sistemas (LINS) del Instituto de Computación de la Facultad de Ingeniería (InCo) (UdelaR), bajo el nombre de MODELO ORQUESTADO, DINÁMICO Y TRANSACCIONAL DE INTEGRACIÓN DE SISTEMAS (M.O.D.T.I.S.) [3]. En el primer proyecto de grado se resolvió la integración de sistemas legados (identificado en la figura I-1 como etapa 1), bajo el título INTEGRACIÓN DE SISTEMAS LEGADOS USANDO WEB SERVICES [4] (año 2002), el cual logra una implementación del modelo de Orquestador-Adaptador sobre el entorno J2EE, según la arquitectura de orquestador y el modelo de adaptadores de software propuesto en [2]. En el siguiente proyecto de grado (identificado en la figura I-1 como etapa 2) se estudiaron aspectos de la transaccionalidad de las macro operaciones, bajo el título TRANSACCIONES DISTRIBUIDAS EN LA WEB [5] (año 2003). En el mismo se propuso un modelo que resuelve la problemática planteada imponiendo algunas condiciones a los servicios, diseñando e

implementando un componente llamado Gestor Transaccional, para el modelo Orquestador-Adaptador y una nueva versión del Orquestador sobre el entorno Microsoft .NET.

La siguiente figura muestra la arquitectura de un COOPERATIVE INFORMATION SISTEM (CIS), según lo descrito en [2] y de qué forma se integran las distintas etapas. En dicha figura aparecen los componentes ya implementados: ORQUESTADOR, GESTOR TRANSACCIONAL y ADAPTADORES y los componentes que se implementaron en el presente proyecto, el REPOSITORIO DE METADATA SEMÁNTICA y el RECONFIGURADOR.

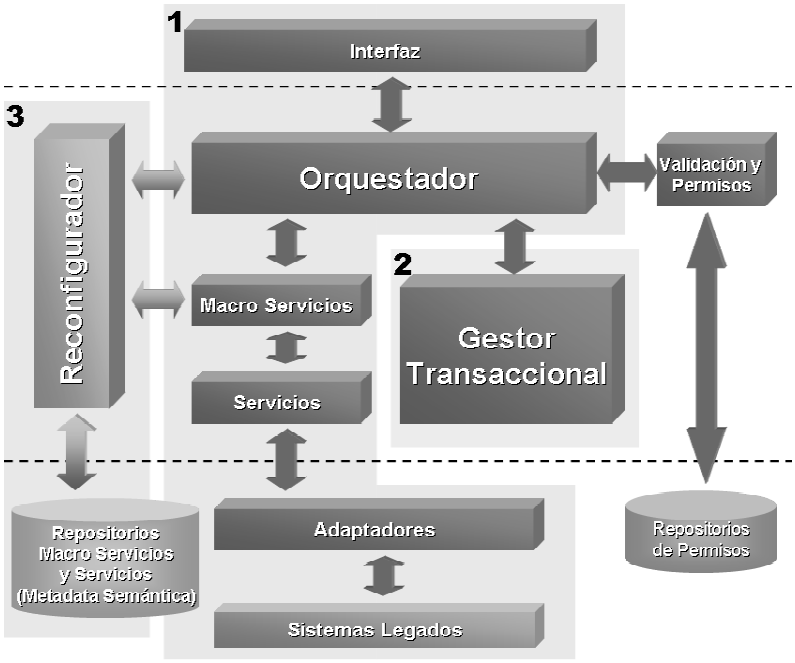

Figura I-1: Arquitectura del CIS, 1-Integración, 2-Transaccionalidad, 3-Metadata Semántica

Básicamente el enfoque tomado por esta serie de proyectos, es el de tener un componente principal llamado ORQUESTADOR quién será el encargado de coordinar (orquestar) las actividades necesarias para llevar a cabo cierta funcionalidad. Lo que se quiere es, dado un conjunto de servicios existentes, implementar uno nuevo combinando los anteriores, produciendo una nueva funcionalidad de mayor nivel de abstracción.

Aquí surge el concepto de Macro Operación (ver II - 1 Conceptos generales) u operación compuesta por varias operaciones (servicios). El lenguaje utilizado para describir una macro operación se llama THALIA y fue definido y concebido en la tesis [2], donde se sientan las bases teóricas y arquitectónicas para los proyectos mencionados en esta introducción.

Por otro lado, el ORQUESTADOR y sus componentes requieren de información sobre las propiedades de las macro operaciones y sobre los servicios que las componen. La información existente en los repositorios UNIVERSAL DESCRIPTION DISCOVERY AND INTEGRATION (UDDI) [7], por ser definida para procesos individuales no contempla los requerimientos necesarios para sustentar coreografías de software. En la tesis [2] se desarrollan los atributos de información requeridos para los servicios y para las macro operaciones. En las etapas anteriores se utilizó parcialmente esas definiciones ya que se ajustaron a lo necesario para su cometido.

En esta realidad, es de esperar que los servicios no estén bajo el estricto control de quien los consume. Por lo tanto, cualquier cambio en un servicio haría que la macro operación involucrada dejara de funcionar, y además sin conocer el motivo. Esto dio lugar a la tercer etapa, el proyecto de grado DISEÑO DE LA METADATA SEMÁNTICA PARA UN ORQUESTADOR DE OPERACIONES ENCAPSULADAS - UNA ONTOLOGÍA PARA INFERIR CAMBIOS A NIVEL DE SERVICIOS [6] (año 2004).

El desafío fue metadatizar la información de servicios y macro operaciones, para lo cual resultó útil la especificación WEB ONTOLOGY LANGUAGE - SERVICES (OWL-S) [21]. Así como también proveer al modelo de un mecanismo que reconozca los cambios en los servicios utilizados por el ORQUESTADOR, y dinámicamente ajuste la información en la metadata, logrando la reconfiguración de las macro operaciones en forma automática, si esto es posible.

Para aclarar un poco más qué es lo que se pretende resolver, se presenta el siguiente ejemplo: se desea poder armar un paquete de viaje, para el cual se utilizarán varios servicios distribuidos en la Web.

Se cuenta con dos servicios: reserva_hotel y reserva_auto brindados por empresas diferentes, sin embargo los conceptos involucrados pertenecen a una misma realidad, a la cual se puede referir como turismo. Al servicio reserva_hotel, se le debe pasar como parámetros la ciudad, la cantidad_de_personas, el nombre_del_cliente y el numero_de_tarjeta para poder debitar el costo de la habitación. Al servicio reserva_auto, se le debe pasar como parámetros el tipo_de_vehiculo deseado y el numero_de_tarjeta del cliente para poder debitar el costo del alquiler. Como se puede observar en ambos servicios se comparten conceptos, el número de la tarjeta de crédito. Una vez definidos los servicios, resta definir la macro operación armar_paquete_de_viaje, la misma es el resultado de ejecutar en forma secuencial reserva_hotel y luego reserva_auto.

Supongamos que el servicio reserva_auto cambia, y el cambio consiste en que se le agrega el parámetro de entrada nombre_de_cliente. En la situación actual del sistema, la macro operación abortaría debido a un error en la ejecución del mismo. Sin embargo, desde el punto de vista conceptual, puede verse que el concepto nombre_de_cliente ya es utilizado por otro servicio dentro de la macro operación. Por lo cual, una vez detectado el cambio y analizado el mismo, se obtiene que asignando el valor del parámetro nombre_de_cliente, obtenido de la ejecución del servicio reserva_hotel, al nuevo parámetro nombre_de_cliente del servicio reserva_auto, la macro operación puede seguir funcionando correctamente.

I - 2 OBJETIVOS

El objetivo principal es implementar un nuevo componente del modelo Orquestador-Adaptador, que utilice la metadata de cada servicio, para detectar los cambios que realiza el desarrollador de los servicios utilizados por el Orquestador, y analizar el impacto producido en las macro operaciones que los utilizan.

Podemos sintetizar los objetivos en los siguientes puntos:

- *Descripción semántica de los servicios:* definir y diseñar la metadata necesaria para describir semánticamente un servicio.

- *Descripción semántica de la macro operación:* para poder inferir posibles correcciones en los cambios, se necesita el grafo de ejecución de las operaciones inscriptas dentro de una macro operación.

- *Monitoreo de cambios en los servicios:* se debe implementar un mecanismo que permita la detección automática de cambios en los servicios.

- *Componente Reconfigurador:* la función de este nuevo componente consiste en, dado uno o más cambios, utilizar la metadata para analizar e inferir posibles soluciones y luego actualizar dicha metadata para lograr la reconfiguración deseada.

Por más detalle ver Anexo I – Objetivos [37].

I - 3 CONCLUSIONES

Se logró una solución extensible, y los componentes implementados gozan de bajo acoplamiento, con los implementados en los proyectos predecesores a éste. La misma se basa en el uso de estándares y la reutilización de componentes externos.

A continuación se presenta un resumen de las conclusiones más significativas del proyecto, las cuales serán expuestas en el capítulo VI.

- No se obtuvo evidencia de que la comunidad académica haya abordado aún, la temática de este proyecto.
- El uso de coreografías para modelar procesos de negocios se encuentra aún en la fase de definición de estándares.
- En nuestra opinión la especificación OWL-S se presenta como candidato a convertirse en el estándar que describa semánticamente a los servicios.
- El grafo de ejecución inscripto dentro de una coreografía, nos provee semántica acerca de la macro operación.
- El componente RECONFIGURADOR funciona como un autómata y las tareas realizadas por él son trazables.

II ESTADO DEL ARTE

En un principio, el estudio del estado del arte, se enfocó en la búsqueda de información referente a la detección de los cambios en los servicios. El resultado de esta actividad es que no se encontraron propuestas concretas. Sin embargo, se identificó que se necesitaban estándares y herramientas relacionadas con metadata semántica y descripción de coreografías.

Para la definición de la metadata fue necesario investigar sobre ontologías. Entre las tecnologías y estándares estudiados se tomó como base de la solución OWL-S [21], que presenta una descripción semántica de los servicios. Es aquí donde está el punto de conexión entre el proyecto y la investigación del uso de semántica para autómatas.

Con respecto a la descripción de coreografías, ya se tenía definido el lenguaje a utilizar (THALIA). Fue necesario investigar esta especificación ya que en ella existe información útil para el análisis de los cambios de los servicios.

Por más detalle ver el Anexo II – Estado del Arte [37].

II - 1 CONCEPTOS GENERALES

Los siguientes conceptos formalizan lo presentado en el punto I-1.

- *Operación:* una operación o servicio es una función que está perfectamente definida, auto contenida e independiente, en su ejecución, del contexto o estado de ejecución de otros servicios.

- *SOA:* SERVICE ORIENTED ARCHITECTURE. Una arquitectura orientada a servicios, tiene por cometido esencial ambientar las condiciones para albergar una colección de servicios y dotarlas de la capacidad de comunicarse entre si y con el exterior en forma segura y bajo las condiciones de calidad de servicio establecidas. La comunicación puede ser desde un simple tráfico de datos a, en el otro extremo de la complejidad, que dos o más servicios interactúen entre si o para la realización de actividades coordinadas (coreografías de software).

- *Macro Operación:* conjunto de operaciones o servicios, más una lógica definida que dicta como interactúan los mismos para lograr cierta funcionalidad. Una macro operación es en si misma otro servicio que puede ser reutilizado como tal o compuesto con otros.

- *Ontología:* "una ontología define los términos y las relaciones básicas para la comprensión de un área, así como las reglas para combinar los términos para definir las extensiones del vocabulario" (Neches).

II - 2 ESTÁNDARES Y TECNOLOGÍAS

La investigación sobre ontologías comenzó con la búsqueda de distintos estandares y tecnologías asociadas a la web semántica, entre ellas: RDF, DAML, OWL y OWL-S.

RESOURCE DESCRIPTION FRAMEWORK (RDF) es un lenguaje para describir información sobre recursos en la web, pero no posee la capacidad de representar ontologías. Para este fin se crean otros lenguajes que extienden RDF, como ser DARPA AGENT MARKUP LANGUAGE (DAML) y OWL.

En la búsqueda por estandarizar DAML, surge OWL como recomendación de la W3C. De esta manera OWL es la tecnología sucesora de DAML.

Uno de los objetivos es describir semánticamente los servicios. Para esto existen dos alternativas: definir una ontología o reutilizar una ya existente. De la investigación se concluye que OWL-S se enfoca en resolver esta problemática. Se decidió utilizar OWL-S por basarse en un lenguaje estándar (OWL), y que al parecer tiende a convertirse en el estándar en esta materia.

II - 2.1 *Lenguajes para la descripción de coreografías*

El lenguaje utilizado fue THALIA, un lenguaje declarativo para definir una macro operación. La definición del mismo puede encontrarse en la tesis [2].

La especificación de una macro operación, aporta semántica respecto a la lógica de interacción entre los distintos servicios participantes, y es una fuente para entender la relación y dependencia entre los distintos parámetros de los mismos.

Esta especificación se encuentra elaborada en un formato propietario, no conocido hasta ahora, ni tampoco extiende la funcionalidad de otro tipo de especificaciones. Se investigó acerca de parsers y tecnologías relacionadas, que transformen un archivo de texto plano en una estructura de datos en memoria. Teniendo en cuenta que el entorno de desarrollo utilizado fue Microsoft .NET, se utilizó una herramienta de uso libre, denominada GRAMMATICA [35].

La industria provee alternativas para la especificación de coreografías. Una de ellas es BPEL4WS BUSSINESS PROCESS ELEMENTARY LANGUAGE FOR WEB SERVICES, la cual es promocionada por las empresas mas representativas de la industria del software.

II - 2.2 *Lenguajes para la definición de ontologías*

Se utilizó el **L**enguaje de **O**ntología para la **W**eb (OWL: Web Ontology Language) [20], el cual es un lenguaje de marcado semántico para definir y compartir ontologías, orientadas a su uso en la Web. El mismo apunta a ser el estándar para la definición de ontologías en el marco de la Web Semántica, siendo ya una recomendación de la W3C.

Básicamente la definición de una ontología utilizando OWL, consiste en definir clases, propiedades entre clases e instancias de clases y propiedades de forma tal que describan los conceptos de una realidad en particular.

Debido a la amplitud de este tema (OWL) y el gran número de aplicaciones en las que se puede utilizar, en la siguiente sección se presenta OWL-S, una ontología específica para describir las propiedades y capacidades de un Web Service.

II - 2.3 *Ontologías para la descripción de servicios*

OWL-S [21] es una ontología definida con el lenguaje OWL. La idea fundamental de OWL-S es describir, al nivel de aplicación, qué es lo que hace un Web Service, no solamente como lo hace. Cuando se dice al nivel de aplicación, se quiere dar a entender que está solucionado el mecanismo o protocolo para publicar, descubrir, describir conceptual y funcionalmente, y transportar la información de un Web Service. Se dará además, una idea de cómo se comunica este lenguaje con el protocolo actualmente utilizado para describir funcionalmente un Web Service, WSDL.

Por lo tanto, OWL-S proporciona a los proveedores de Web Services, un conjunto base de constructores de lenguaje de marcado, que permiten la descripción de propiedades y capacidades de los Web Services en una forma no ambigua y capaz de ser consumida por autómatas. De esta forma, el uso de OWL-S, facilitará la tarea de automatización de servicios tales como descubrimiento de servicios, ejecución, inter-operación, composición y monitoreo de los mismos.

A continuación se describirá como está constituido OWL-S, y se bajará el nivel de detalle en aquellas partes de mayor interés para el proyecto. Esta ontología define un conjunto de clases y propiedades específicas para la definición de Web Services. Dicha definición esta estructurada en forma jerárquica, comenzando por la clase SERVICE en el tope de la jerarquía.

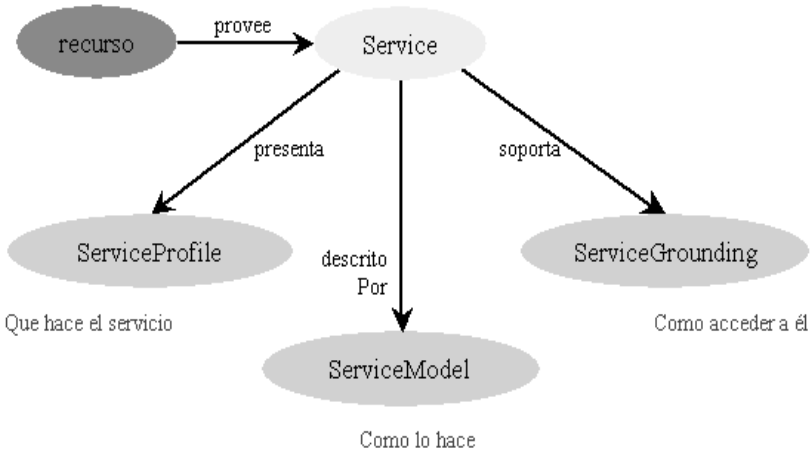

Figura II-1 Descripción jerárquica de la ontología OWL-S

Se debe saber que es lo que requiere un Web Service de los agentes que lo consumen y de quien los provee, es por esta razón que la ontología contiene la clase SERVICEPROFILE. Entonces se dice que la clase SERVICE presenta un SERVICEPROFILE. Esta clase presenta cierta información que puede ser utilizada por el agente para determinar si el Web Service cumple con sus necesidades y si satisface las restricciones de seguridad, ubicación, calidad, etc.

Por otro lado se debe saber como trabaja el Web Service, es por esta razón que la ontología contiene la clase SERVICEMODEL. La misma describe el flujo de tareas y un posible camino de ejecución del servicio. Por ello se dice que un SERVICE es descrito por un SERVICEMODEL. Esta clase permite a su agente consumidor:

- Realizar un análisis detallado de cómo el servicio cumple con sus necesidades.
- Componer un servicio desde múltiples Web Services que cumplen una tarea específica.
- Coordinar las tareas de múltiples agentes.
- Monitorear la ejecución del Web Service.

En resumen, SERVICEPROFILE provee de la información necesaria para que un agente descubra el Web Service, mientras que SERVICEMODEL provee de la información suficiente para que el agente haga uso de él.

Por último un SERVICE implementa un SERVICEGROUNDING que provee detalles de cómo inter-operar con el servicio vía mensajes. Cada una de estas especificaciones describen una parte del área del conocimiento concerniente a los Web Services.

El grounding se relaciona fuertemente con la especificación WSDL. Las especificaciones de estos dos lenguajes son complementarias, OWL-S no sustituye a WSDL. Los lenguajes se solapan en lo que en WSDL se designan como tipos abstractos, que caracterizan las entradas y salidas del servicio. OWL-S provee la capacidad de darle sentido semántico a las definiciones abstractas, mientras que por otro lado no tiene la capacidad de expresar la información de ligado (binding) que tiene WSDL. En la siguiente figura se muestra a modo de esquema esta relación.

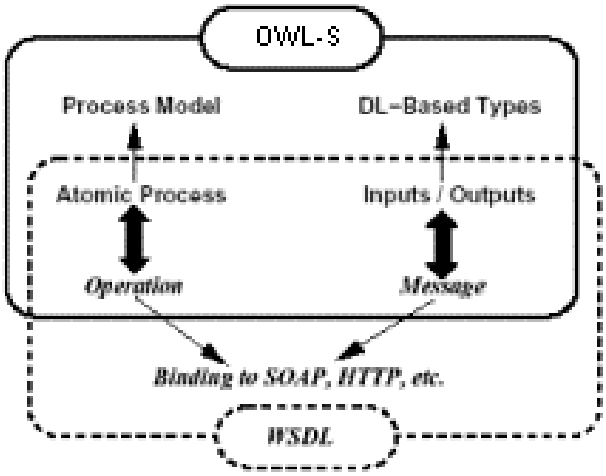

Figura II-2 Relación entre OWL-S y WSDL

Se investigaron varias herramientas relacionadas a la especificación de ontologías, algunas de ellas son: Protegee, Jena, DAML dotnetAPI, RACER. Protegee es un IDE genérica para la creación y edición de ontologías que soporta varios lenguajes, entre ellos OWL. Jena es un API para la edición de ontologías a nivel de programación en java, DAML dotnetAPI es un porte parcial de la especificación Jena en la tecnología .NET. RACER es un motor de inferencia sobre ontologías, implementado en java que funciona como un servicio.

Si bien estas herramientas sirven para el manejo de ontologías, las mismas no se ajustaban exactamente a las necesidades del proyecto. Por esta razón se implementaron los componentes de software ajustados a la realidad del proyecto. Por más información consultar el Anexo II – Estado del Arte [37].

II - 3 APORTES Y CONCLUSIONES

OWL-S es una especificación bien estructurada, pero debió extenderse para utilizarla en el contexto del proyecto.

OWL-S al igual que THALIA se puede utilizar para describir coreografías. No obstante, esta funcionalidad no se utilizó en el proyecto, debido a que el uso de THALIA fue un prerrequisito del mismo.

Las investigaciones en el área de adaptación a la evolución de los servicios, se encuentran aún en sus primeros pasos. No existen proyectos concretos relacionados con este tema, sin embargo las bases teóricas han sido investigadas en forma individual, como ser ontologías, web semántica, coreografías.

III PRESENTACIÓN DE LA SOLUCIÓN

En este capítulo se presenta la solución a los objetivos planteados. Se describe la especificación, diseño y arquitectura del componente implementado, RECONFIGURADOR. Además se muestran las modificaciones realizadas en los repositorios ya implementados. Estos temas se detallan en el Anexo III - Especificación y Diseño del Reconfigurador [39].

En la implementación del modelo ORQUESTADOR-ADAPATADOR no son tomados en cuenta los cambios que el desarrollador de un servicio pueda realizar. El objetivo de implementar el RECONFIGURADOR e integrarlo al sistema, es lograr que el sistema se adapte a dichos cambios de forma automática.

En otras palabras, permitir que, dado algún cambio en un servicio, el sistema pueda ser capaz de reflejar esos cambios en su metadata y seguir adelante con su cometido, sin necesidad de conocimiento o intervención humana. Así el ORQUESTADOR podrá ejecutar con éxito una macro operación sin tener conocimiento de que esta ha cambiado, y que de otra forma (si no existiera el RECONFIGURADOR) no hubiese podido ejecutar.

La solución propuesta consiste por un lado, en agregar a los servicios información semántica, y a la vez persistir dicha información en los repositorios. Por otra parte, se tiene un componente de software que monitorea los servicios, comparando su información semántica contra lo almacenado en los repositorios. Una vez que los cambios son detectados y clasificados, el algoritmo de reconfiguración analiza cada cambio en forma individual. Sin embargo, se requieren varias iteraciones por este algoritmo, ya que un cambio puede ser resuelto en el contexto de varios cambios simultáneos, dentro de una macro operación. Como resultado, se obtendrá una nueva especificación de la macro operación adaptada a los cambios.

III - 1 ARQUITECTURA Y DISEÑO DEL RECONFIGURADOR

La arquitectura del RECONFIGURADOR se basa en tres capas jerárquicas. En la de más alto nivel, se encapsulan los sub-sistemas que resuelven el algoritmo de reconfiguración. El resto de las capas están subordinadas a esta ofreciéndole funciones útiles para su tarea.

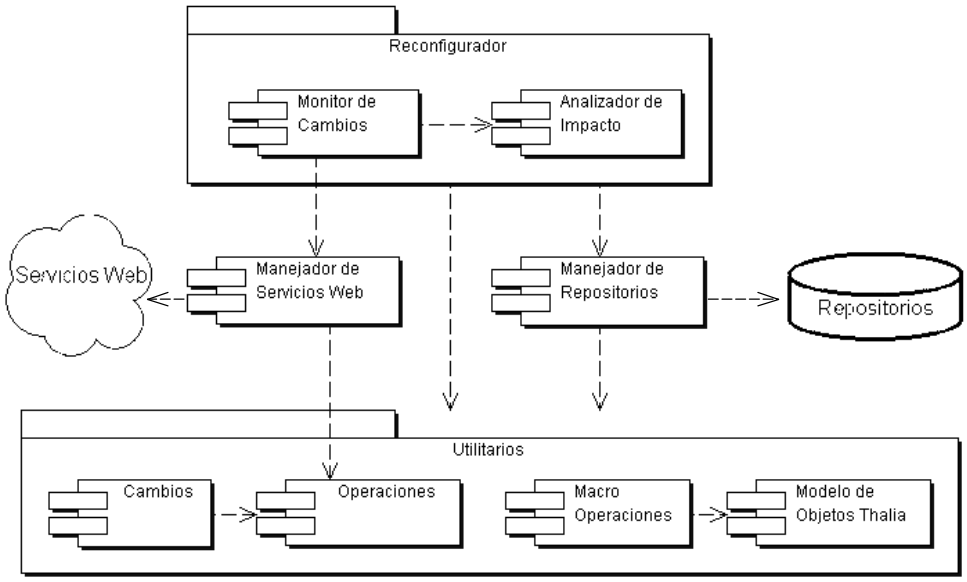

Figura III-1 : Arquitectura del Reconfigurador

La capa Reconfigurador se divide en dos sub-sistemas. El primero, MONITOR DE CAMBIOS, se encarga únicamente de detectar los cambios en los servicios. El segundo, ANALIZADOR DE IMPACTO, se encarga de analizar el impacto de los cambios y tomar las decisiones pertinentes.

El ANALIZADOR DE IMPACTO necesita determinar el alcance del cambio, para esto hace uso del MODELO DE OBJETOS THALIA, que le permite manejar el grafo de ejecución inscripto en la coreografía que describe la macro operación.

Además necesita realizar modificaciones en los repositorios, ya sea porque tiene la posibilidad de reconfigurar una macro operación, en algunos casos

cambiando la especificación THALIA, o para dejar registrados los cambios. Es mediante el componente MANEJADOR DE REPOSITORIOS que realiza estas tareas.

Si el MONITOR DE CAMBIOS detecta al menos un cambio, invoca al ANALIZADOR DE IMPACTO indicando, mediante una estructura de datos, los cambios encontrados para cada servicio. El MONITOR DE CAMBIOS interactúa con el MANEJADOR DE REPOSITORIOS para consultar por las operaciones existentes, e interactúa con el ANALIZADOR DE IMPACTO para que éste último pueda resolver qué hacer con los cambios encontrados.

III - 2 SEMÁNTICA DE LOS SERVICIOS

Para poder determinar la evolución de los servicios, surge la necesidad de darle semántica a los mismos. Para esto se exige al implementador del servicio, que provea un profile OWL-S que describa el servicio.

La solución planteada utiliza una versión extendida de la especificación OWL-S. La misma consiste en definir una propiedad nueva a las clases ontológicas que describen un parámetro de un servicio. Se extendió la clase ParameterDescription, agregando una nueva propiedad llamada defaultValue. Esto se debe a que podemos reconfigurar un servicio, si se agrega un parámetro y el mismo tiene un valor por defecto.

III - 3 REPOSITORIOS

Para cada servicio utilizado en el sistema, se publica un archivo OWL-S con el Profile del mismo. Todo cambio en el servicio debe estar reflejado en esta especificación. El MONITOR DE CAMBIOS compara la información que está persistida en los repositorios con la información publicada.

Es por esta razón que se modificaron los repositorios utilizados en los proyectos antecesores a este. Se extendió el esquema de la base de datos, para incluir lo referente a esta metadata semántica.

Por otro lado, la implementación del modelo ORQUESTADOR-ADAPTADOR no persistía en los repositorios las coreografías de las macro operaciones. En el contexto de este proyecto se modificaron los repositorios para incluir esta información, además de indicar si una macro operación se encuentra o no

disponible. Esto fue necesario para lograr que el ORQUESTADOR, consuma la coreografía reconfigurada de forma transparente.

Otra modificación a los repositorios, es la concerniente al log persistido con los cambios realizados en el algoritmo de reconfiguración. Esta información estructurada le permite al administrador del sistema, obtener trazabilidad de las acciones tomadas en forma automática por el RECONFIGURADOR.

Los repositorios se definen en el Anexo VII – Repositorios [43].

III - 4 MODELO DE OBJETOS THALIA

Debido a que el objetivo principal del ANALIZADOR DE IMPACTO, es manipular macro operaciones, para lograr adaptarse a los cambios, es necesario contar con un sub-sistema que brinde funcionalidades para dicha manipulación. Para esto se implementa el MODELO DE OBJETOS THALIA , el cual se encarga de parsear, editar y serializar una especificación THALIA. Lo más importante del mismo, es que su diseño logra desacoplar por completo el lenguaje THALIA, obteniendo un sistema extensible, donde se podría sustituir la especificación de coreografías utilizada.

El ANALIZADOR DE IMPACTO hace uso del grafo de ejecución inscripto en la macro operación, para determinar como afectan los cambios y encontrar posibles soluciones. Es tarea del MODELO DE OBJETOS THALIA obtener dicho grafo.

III - 5 CLASIFICACIÓN DE CAMBIOS

En la siguiente figura se presentan los posibles cambios en los atributos de un servicio y la acción tomada por el algoritmo en cada caso. La columna *resultado* indica si la re-configuración es posible o no. Por ejemplo, si se tiene un servicio que es transaccional y cambia el atributo IDSERVICIOROLLBACK indicando un servicio de rollback que no existe, la macro operación deberá quedar deshabilitada (Resultado = Error) ya que no podrá cumplir con la funcionalidad de transaccionalidad. Si por el contrario, el nuevo servicio de rollback existe, entonces la nueva configuración es posible (Resultado = Éxito).

Atributo	Valor Anterior	Valor Actual	Resultado
Transaccionalidad	Si	No	Éxito
Transaccionalidad	No	Si	Advertencia
Estado	Cualquiera	No disponible	Error
Estado	Cualquiera	Disponible	Éxito
Estado	Cualquiera	Testing	Advertencia
Timeout	-	Comparar el total de timeouts: Si TIMEOUT_MACRO < MAX_TIMEOUT_TOTAL_POR_RUTA	Advertencia
Timeout	-	Comparar el total de timeouts: Si TIMEOUT_MACRO > MAX_TIMEOUT_TOTAL_POR_RUTA	Éxito
Sincronismo	Si	No	Advertencia
Sincronismo	No	Si	Advertencia
IdServicioRollBack	-	Cambia y no existe una operación con el nuevo identificador	Error
IdServicioRollBack	-	Cambia y existe una operación con el nuevo identificador	Éxito
IdServicioRollBack	Defindo	No definido	Error
VersionServicio	-	Si cambia	Advertencia

Figura III-2 : Cambios en los atributos del servicio

En la siguiente figura se presentan los posibles cambios en los parámetros de un servicio y la acción tomada por el algoritmo en cada caso. La columna *resultado* indica si la re-configuración es posible o no.

Retomando el ejemplo de la macro operación Viaje, supóngase que el servicio reserva_auto cambia, y el cambio consiste en que se le agrega el parámetro de entrada nombre_de_cliente. Este cambio tiene solución y consiste en asignar el valor del parámetro nombre_de_cliente, obtenido en la ejecución del servicio reserva_hotel, al nuevo parámetro nombre_de_cliente del servicio reserva_auto. Este caso de reconfiguración, se puede apreciar en la segunda fila de la tabla.

Tipos de Cambio	Situación Anterior	Acción tomada	Resultado
Se quita un parámetro de entrada	Le pasaba el valor de una variable o constante	Saco el parámetro de la invocación al servicio	Éxito
Se agrega un parámetro de entrada	Tengo una variable en la m.o. definida y cargada antes del llamado al servicio que se refiere al mismo concepto	Agrego el parámetro en la invocación al servicio, asignándole dicha variable.	Éxito
Se agrega un parámetro de entrada sin valor por omisión	No tengo una variable en la m.o. definida y cargada antes del llamado al servicio que se refiere al mismo concepto	-	Error
Se agrega un parámetro de entrada con valor por omisión	No tengo una variable en la m.o. definida y cargada antes del llamado al servicio que se refiere al mismo concepto	Agrego el parámetro en la invocación al servicio, asignándole el valor por omisión.	Éxito
Cambia la naturaleza de un parámetro de entrada	Tengo una variable en la m.o. definida y cargada antes del llamado al servicio que se refiere al nuevo concepto	Cambio al nuevo parámetro en la invocación al servicio	Éxito
Cambia la naturaleza de un parámetro de entrada sin valor por omisión	No tengo una variable en la m.o. definida y cargada antes del llamado al servicio que se refiere al nuevo concepto	-	Error
Cambia la naturaleza de un parámetro de entrada con valor por omisión	No tengo una variable en la m.o. definida y cargada antes del llamado al servicio que se refiera al nuevo concepto	Asigno al parámetro el valor por omisión	Éxito
Se quita un parámetro de salida	No lo utilizo en el resto de la m.o.	Saco el parámetro del retorno de la invocación al servicio	Éxito
Se quita un parámetro de salida	Lo utilizo, es decir, es parámetro de entrada a otro servicio posterior ó es la salida de la m.o.	-	Error
Se agrega un parámetro de salida	-	Agrego una nueva variable al retorno de la invocación, aunque no la utilice	Éxito
Cambia la naturaleza de un parámetro de salida	Lo utilizaba	-	Error
Cambia la naturaleza de un parámetro de salida	No lo utilizaba	-	Éxito

Figura III-3 : Cambios en los parámetros de entrada y salida del servicio

III - 6 ALGORITMO DE RECONFIGURACIÓN

El ANALIZADOR DE IMPACTO toma como entrada una lista de cambios. Cada elemento de esta lista representa un cambio en alguna operación, y a su vez cada operación puede ser utilizada por una o más macro operaciones. El ANALIZADOR DE IMPACTO agrupa estos cambios por macro operación. Esto es debido a que un cambio puede ser resuelto en el contexto de una macro operación, pero no en otra.

Si se detecta más de un cambio en la macro operación, podría ocurrir que considerando uno de los cambios de manera aislada, el mismo no tuviese solución. Pero al considerar ambos cambios, se podría reconfigurar la macro operación. Tal es el caso en el cual una operación agrega un parámetro de entrada y otra agrega un parámetro de salida, cuyos conceptos ontológicos coincidan. Por esto el algoritmo realiza varias iteraciones, de modo de considerar varios cambios al mismo tiempo. Para ello analiza cada cambio intentando solucionarlo, y mientras algún cambio se solucionó en una nueva recorrida, intenta resolver nuevamente aquellos, que aún no han sido resueltos.

Si al finalizar una iteración, todos los cambios fueron resueltos, el algoritmo finaliza con éxito. Esto implica que la macro operación ha sido modificada, acción realizada al resolver cada cambio en forma individual. Si al finalizar una iteración, no se resolvió ningún cambio y al menos un cambio no fue resuelto, entonces el algoritmo finaliza sin éxito, dejando la macro operación deshabilitada.

Si la macro operación pudo ser reconfigurada con éxito, entonces se actualizan las operaciones que cambiaron y la nueva especificación THALIA, dejando la macro operación en estado DISPONIBLE. Si esto no fue así entonces no se persisten los cambios en las operaciones, pero se actualiza el estado de la macro operación a NO_DISPONIBLE. En ambos casos se persisten todos los cambios detectados.

En el Anexo VI - Análisis Detallado del Analizador de Impacto [42] se explica en profundidad este algoritmo.

IV IMPLEMENTACIÓN DE LA SOLUCIÓN

En este capítulo se dará una descripción del software implementado, con el fin de presentar las decisiones de implementación tomadas, y poder justificarlas adecuadamente.

En el capítulo anterior se presentaron los componentes implementados. El RECONFIGURADOR se implementó como un servicio de Windows, el cual monitorea periódicamente los profiles OWL-S publicados en la Web.

Durante el proceso de reconfiguración, el ANALIZADOR DE IMPACTO persiste información estructurada del estado del proceso, en una cola de mensajes. Esta información es obtenida por el componente MONITOR GRAFICO quien presenta la misma al administrador del sistema. De esta manera se puede monitorear el proceso de reconfiguración on-line.

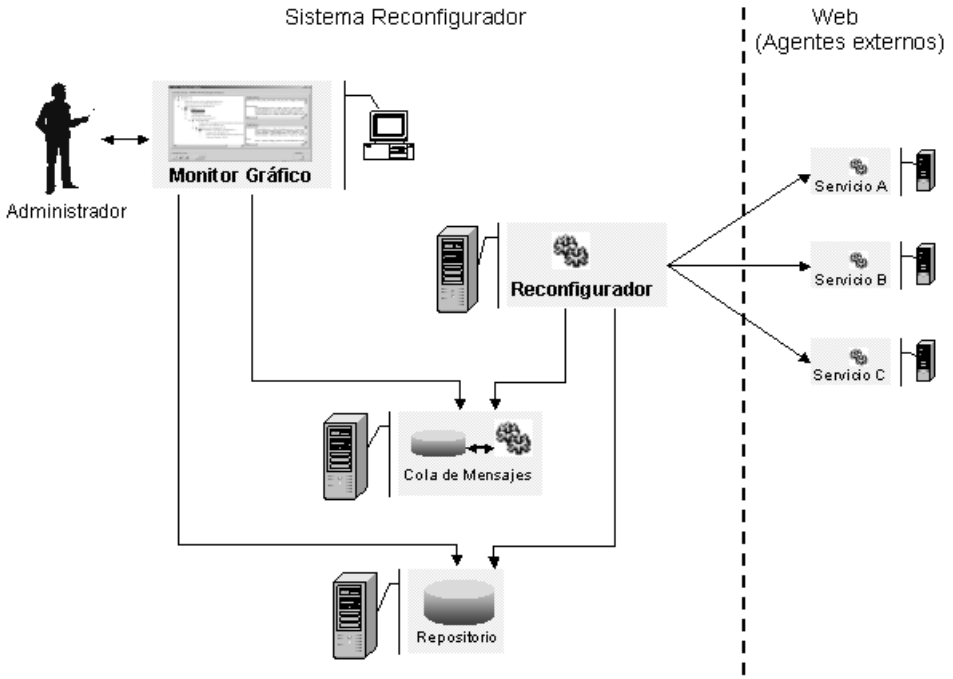

Figura IV-1 : Diagrama de comunicación entre Componentes

IV - 1 FRAMEWORK DE DESARROLLO

Para la implementación de la solución se utilizó la tecnología .NET por ser un requisito del proyecto. Las principales bibliotecas o componentes del framework utilizados en esta implementación son: manejo de archivos XML, ODBC, MessageQueuing y Win-Forms.

Debido al enfoque propietario de esta tecnología, resultó difícil encontrar bibliotecas o componentes de software de uso libre. Aún así, se encontró una biblioteca de uso libre para la manipulación de gramáticas, funcionalidad no provista por el ambiente de desarrollo.

IV - 2 IMPLEMENTACIÓN DE LA SEMÁNTICA DE UN SERVICIO

Cada servicio deberá publicar, además del archivo WSDL, un archivo de descripción semántica en formato OWL-S. En el mismo se describe la semántica de los atributos y parámetros del servicio.

En el Anexo VIII - Manual de Instalación, Implantación y Administración [44] se explica y se dan ejemplos de cómo se utiliza esta especificación en el contexto de la solución. A continuación se presenta la extensión que se realizó a esta especificación, para contemplar valores por omisión en los parámetros del servicio.

OWL, es extensible por definición, por lo tanto se extendió la clase ParameterDescription, definida en OWL-S, agregándole una nueva propiedad llamada defaultValue. La misma puede contener cualquier tipo de elemento, lo que es ideal para asignar valores por omisión en la definición de un parámetro. Esta extensión se logra con la siguiente declaración en OWL:

```
<owl:Property rdf:ID="defaultValue">
   <owl:domain rdf:resource="&profile;#ParameterDescription"/>
   <!-- <owl:range rdf:resource="&owl;#Class"/> -->
</owl:Property>
```

En esta declaración aparece comentado el rango de la propiedad, para dejar explícito que su valor puede ser cualquier concepto ontológico. El siguiente es un ejemplo de la declaración de un parámetro con valor por omisión.

```
<profile:input>
   <profile:ParameterDescription rdf:ID="pd2.1">
      <profile:parameterName>moneda</profile:parameterName>
      <profile:restrictedTo rdf:resource="&CCard;#Moneda"/>
       <defaultValue>Pesos_Uruguayos</defaultValue>
   </profile:ParameterDescription>
</profile:input>
```

IV - 3 IMPLEMENTACIÓN DE LOS REPOSITORIOS

Si bien un requisito del proyecto fue utilizar los repositorios existentes (definidos en los proyectos antecesores), fue necesario modificarlos agregando y eliminando algunas tablas y atributos.

Las entidades principales que se persisten en los repositorios son: macroservicios, servicios, parámetros de los servicios.

Una de las principales modificaciones fue la persistencia de la especificación THALIA de cada servicio. Esto fue una decisión importante para la integración con el componente ORQUESTADOR-ADAPTADOR.

Además se almacenó la información referida a los profiles de cada servicio. En la entidad de parámetros de cada servicio se incorporó el concepto ontológico de cada uno de ellos. De esta manera se persiste el estado semántico de los servicios.

Otra modificación importante fue el hecho de completar la metada de los servicios y macro operaciones en base a la especificación definida en la tesis [2].

IV - 4 PARSER THALIA

El hecho de obtener semántica de las macro operaciones motivó la utilización de la especificación THALIA, como un modelo de objetos en memoria. Dado que esta especificación es propietaria, y no extiende otra especificación conocida, fue necesario construir, un parseador de archivos para esta especificación. Es aquí donde aparece el tema de las gramáticas.

Para este requerimiento se utilizó una biblioteca llamada GRAMMATICA, desarrollada en el ambiente académico y de uso libre. Esta biblioteca nos permite generar un parser a partir de un archivo de especificación de la gramática libre de contexto, esta especificación se presenta en el Anexo IV – Arquitectura del Reconfigurador [40]. A partir de este archivo de especificación, se genera código que es incorporado al proyecto. Las clases generadas por este componente de software son: una con la definición de constantes con los tokens de nuestra gramática, otra clase que genera la lógica para reconocer cada token, otra que genera la lógica para reconocer cada producción, y la última que analiza la gramática y genera un método abstracto por cada producción o token reconocido.

Esta última clase es la que se tuvo que extender para agregar la funcionalidad esperada. Construir a partir de un archivo de especificación, un modelo de objetos en memoria, con la estructura de una macro operación representada en la especificación THALIA.

Toda esta lógica, engorrosa y difícil de entender dado su volumen, está encapsulada en una clase, totalmente desacoplada del modelo de objetos en sí. De esta manera la especificación de una macro operación puede cambiar, pero el modelo de objetos diseñado mantiene su compatibilidad. En este contexto se debería modificar solo el parser del archivo de especificación.

Esta decisión tiene un fuerte impacto en la solución final. Ya que el modelo de objetos de una macro operación, se utiliza para el trasiego de datos en todo el sistema. Un cambio en alguna de estas clases, puede provocar un fuerte impacto en el resto de los componentes implementados dentro de la solución.

IV - 5 MODELO DE OBJETOS THALIA

El modelo de objetos Thalia, es una representación mediante clases de una estructura de la realidad planteada por una especificación de una macro operación, la cual tiene incluídas operaciones, rutas, asignaciones, etc.

La decisión más fuerte en este punto fue dejar el modelo de objetos puro. Esto significa que cada clase tiene sus atributos y métodos de acceso, pero en estas clases no se incluye lógica necesaria para el manejo de esta estructura. Las operaciones de las que hace uso el RECONFIGURADOR, se encapsularon en una clase aparte, fuertemente acoplada al modelo de objetos Thalia, en particular porque la extiende. Sin embargo es fácilmente modificable y extensible, debido a que otros componentes de software, pueden hacer uso del modelo de objetos Thalia, sin tener visibilidad sobre los métodos o funciones utilizadas por el componente RECONFIGURADOR. Estos métodos son necesarios para obtener la semántica tan importante para el algoritmo de reconfiguración, objetivo fundamental de la solución, pero que pueden carecer de utilidad en el contexto de otra aplicación.

IV - 6 AUDITORIA Y TRAZABILIDAD

IV - 6.1 *Monitor gráfico*

Una vez diseñado el componente Reconfigurador, se hizo necesario construir la interfase gráfica que permitiera auditar el proceso de reconfiguración. Esta interfase gráfica es de gran valor para el usuario administrador del sistema, pero planteó un gran desafío a la hora de la implementación: generar un componente débilmente acoplado con el Reconfigurador que presentara toda la información de su actividad.

En la siguiente figura se muestra el monitor gráfico en ejecución, donde se muestra el cambio en un parámetro de un servicio, el cual no ha podido ser resuelto, como se puede observar, se despliega en forma de árbol toda la información, destacando los cambios que no han podido ser resueltos.

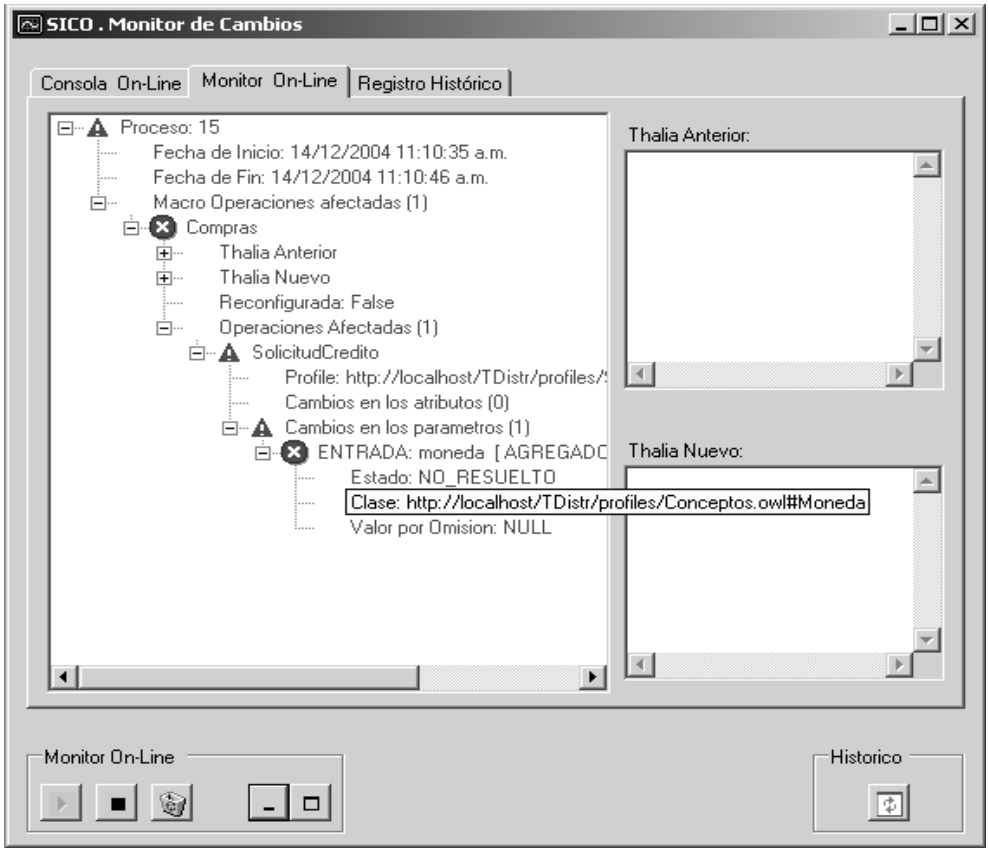

Figura IV-2 : Monitor Gráfico mostrando un cambio no resuelto

Otro detalle importante y útil, es la funcionalidad de desplegar el código THALIA antes y después de la reconfiguración de una macro operación. Esto permite analizar detalladamente los cambios efectuados en el código de la macro operación. En la siguiente figura se muestra el monitor gráfico en ejecución, donde se muestra el cambio en un parámetro de un servicio, el cual ha sido resuelto, como se puede observar, se despliega en forma de árbol toda la información, y además se presenta el código THALIA antes y después de la reconfiguración.

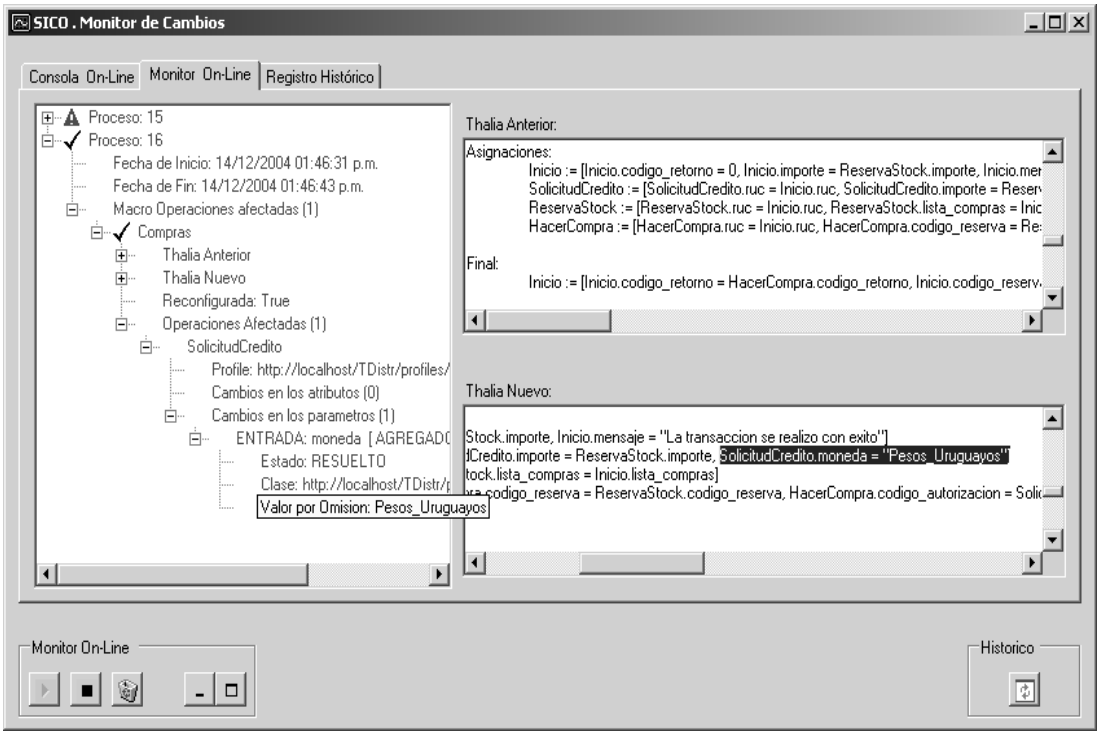

Figura IV-3 : Monitor Gráfico mostrando un cambio resuelto

La solución planteada se apoya fuertemente en la capacidad de comunicación asincrónica que provee MessageQueuing de .NET. MessageQueuing es básicamente una cola de mensajes, sobre la cuál se basa la comunicación asincrónica, en la cual se pueden depositar y obtener mensajes, que para la aplicación son objetos concretos. Es aquí donde se define una jerarquía de clases, que representan los posibles mensajes que se intercambian entre el componente reconfigurador y el componente que implementa la interfase gráfica.

De este modo se obtuvo una componente de software débilmente acoplada al Reconfigurador, fácil de extender, modificar o sustituir, sumados a los factores de calidad que hacen de un componente gráfico de administración útil y fácil de utilizar.

IV - 6.2 *Trazabilidad*

Otro de los puntos importantes para el administrador del sistema, es la trazabilidad. Trazabilidad a la hora de obtener información de las acciones tomadas por el autómata (el RECONFIGURADOR), el cuál toma decisiones en base a una serie de reglas, y produce una serie de cambios en la información del sistema, la cual debe ser auditada y controlada por el administrador.

La trazabilidad de esta información se apoya en los datos persistidos en los repositorios, la implementación de métodos de acceso a esta información, así como la información que va generando el RECONFIGURADOR en tiempo de ejecución.

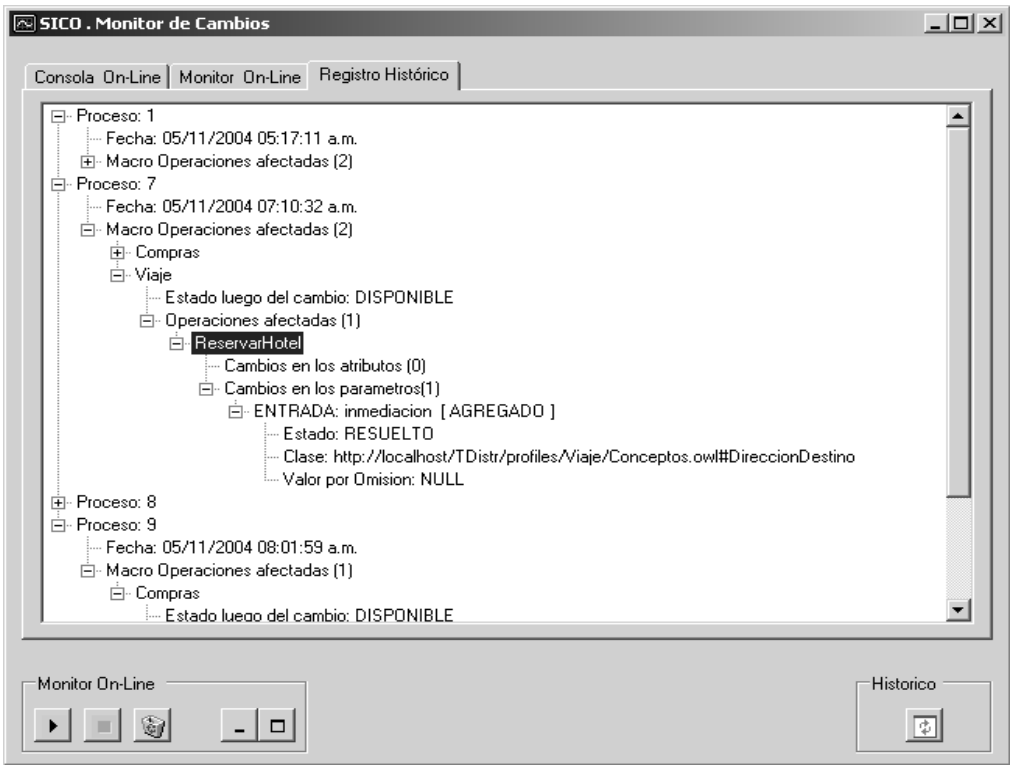

Figura IV-4 : Monitor Gráfico mostrando el registro histórico

Ante cada acción el RECONFIGURADOR va generando un log, tipificando la información, de modo de poder presentarla de forma correcta, ordenada y amigable en la interfase gráfica de monitoreo. Como se mencionó antes, esta

información es depositada en una cola de mensajes por parte del RECONFIGURADOR, para que luego el monitor gráfico la presente en una interfase amigable y útil para el administrador del sistema.

De esta manera se logra una trazabilidad on-line, apoyada por la cola de mensajes y una trazabilidad histórica, apoyada por el repositorio.

V CASO DE ESTUDIO

En este capítulo se analiza el caso de estudio Compras, dando una explicación resumida del mismo. Presentando una serie de cambios y la forma en que estos son resueltos. En el Anexo V - Casos de Estudio [41] se incluyen todos los casos de estudio del proyecto.

El cometido de la macro operación Compras es brindar la posibilidad de comprar ciertos productos a cierta empresa. En el siguiente diagrama se presenta el caso de estudio.

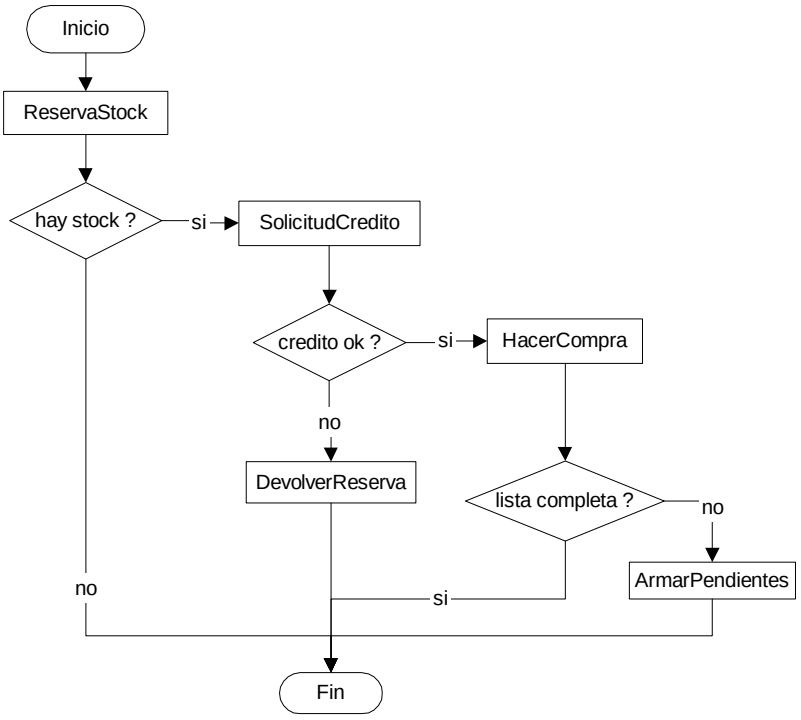

Figura V-1 : Macro Operación Compras

La lógica de la macro operación comienza con la ejecución del servicio RESERVASTOCK donde se hará la reserva de los productos solicitados. Uno de los parámetros de retorno será el importe de los productos reservados, por lo tanto luego se hará uso del servicio SOLICITUDCREDITO quién se encargará de

determinar si el cliente tiene crédito suficiente para satisfacer el importe. En caso de crédito insuficiente, se procederá a realizar la devolución de la reserva llamando a DEVOLVERRESERVA. En el caso de que se autorice el crédito, se procederá a realizar efectivamente la compra llamando a HACERCOMPRA. Finalmente si hubo algún producto que no se pudo reservar, se deberá armar una lista de pendientes, esto se logra llamando al servicio ARMARPENDIENTES.

V - 1.1 *Cambios a resolver*

Para este ejemplo se analizan dos instancias de cambio, la primera es un cambio que no podrá resolverse, por lo tanto, el RECONFIGURADOR deshabilitará la macro operación. La otra instancia de cambio sí tendrá solución, por lo cual se mostrará cómo el sistema es capaz de resolver el mismo.

V.1.1.1 Se agrega un parámetro de entrada sin valor por omisión

Se agrega el parámetro de entrada ***moneda*** sin valor por omisión en el servicio SOLICITUDCREDITO. Se detecta que este parámetro tiene el valor ontológico MONEDA. Según el grafo de ejecución y la metadata conceptual de los servicios restantes, no se cuenta con tal concepto al momento de llamar al servicio SOLICITUDCREDITO, por lo tanto, no existe solución automática para este cambio. El resultado esperado será que la macro operación no podrá reconfigurarse y pasará a estar deshabilitada. En la siguiente figura se muestra cómo el sistema detecta el cambio y despliega los resultados.

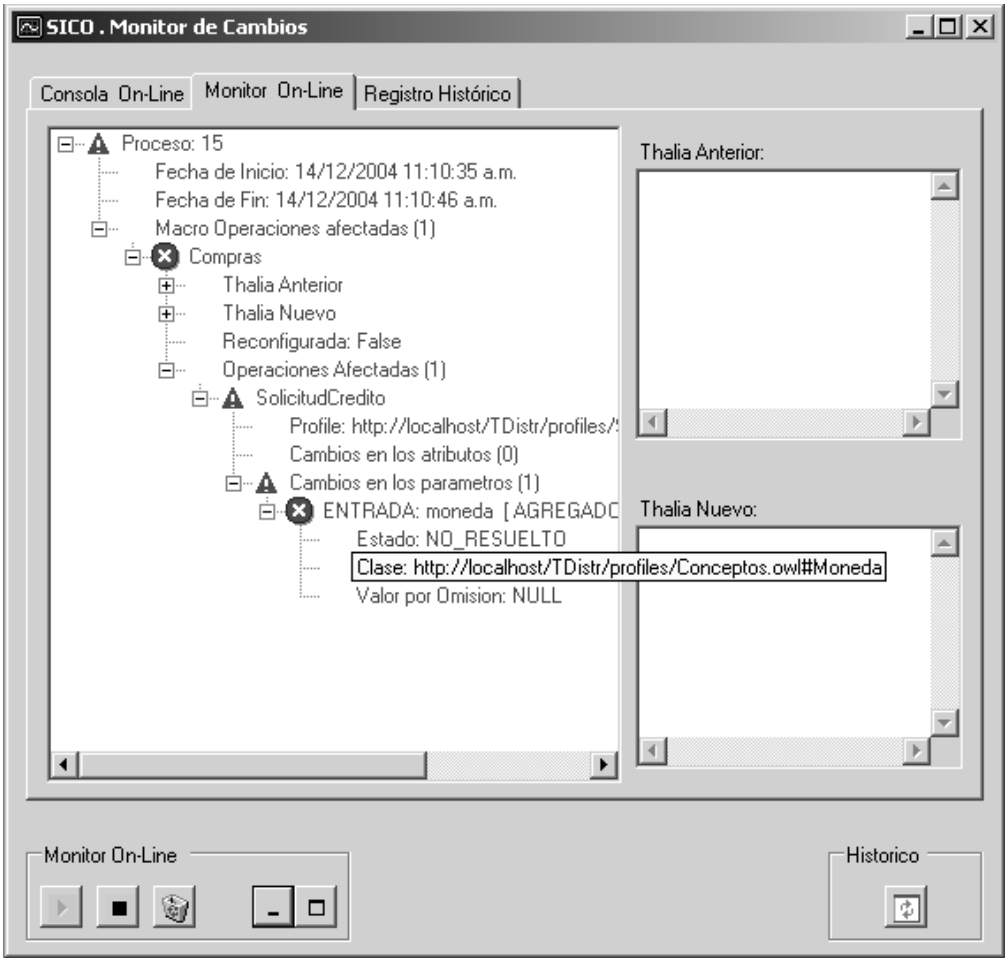

Figura V-2 : Monitor Gráfico mostrando un cambio no resuelto

V.1.1.2 Se agregan múltiples parámetros en varios servicios

Se agrega el parámetro de entrada ***moneda*** sin valor por omisión en el servicio SOLICITUDCREDITO. Además se agrega el parámetro de salida ***moneda*** a la operación RESERVASTOCK quien calcula el importe de la compra. En esta situación estamos frente a un cambio múltiple en el cual la especificación actual de la macro operación ya no es válida, sin embargo, existe una solución posible.

Se sabe que ambos parámetros comparten el mismo concepto, por lo cual, se puede utilizar el valor del parámetro de salida de la operación RESERVASTOCK, como parámetro de entrada en la operación SOLICITUDCREDITO. Para esta asignación no solo se utiliza el conocimiento acerca de los conceptos de la realidad que describen los parámetros, si no que además, se utiliza el conocimiento acerca de la secuencia de ejecución definida en la macro operación. Pues se debe estar seguro que, a la hora de utilizar el valor de un parámetro de salida, el mismo esté siempre definido.

El resultado esperado será una macro operación reconfigurada con éxito. En la siguiente figura se muestra como el sistema detecta el cambio y devuelve los resultados, mostrando además la nueva especificación del código THALIA generado que refleja dichos cambios.

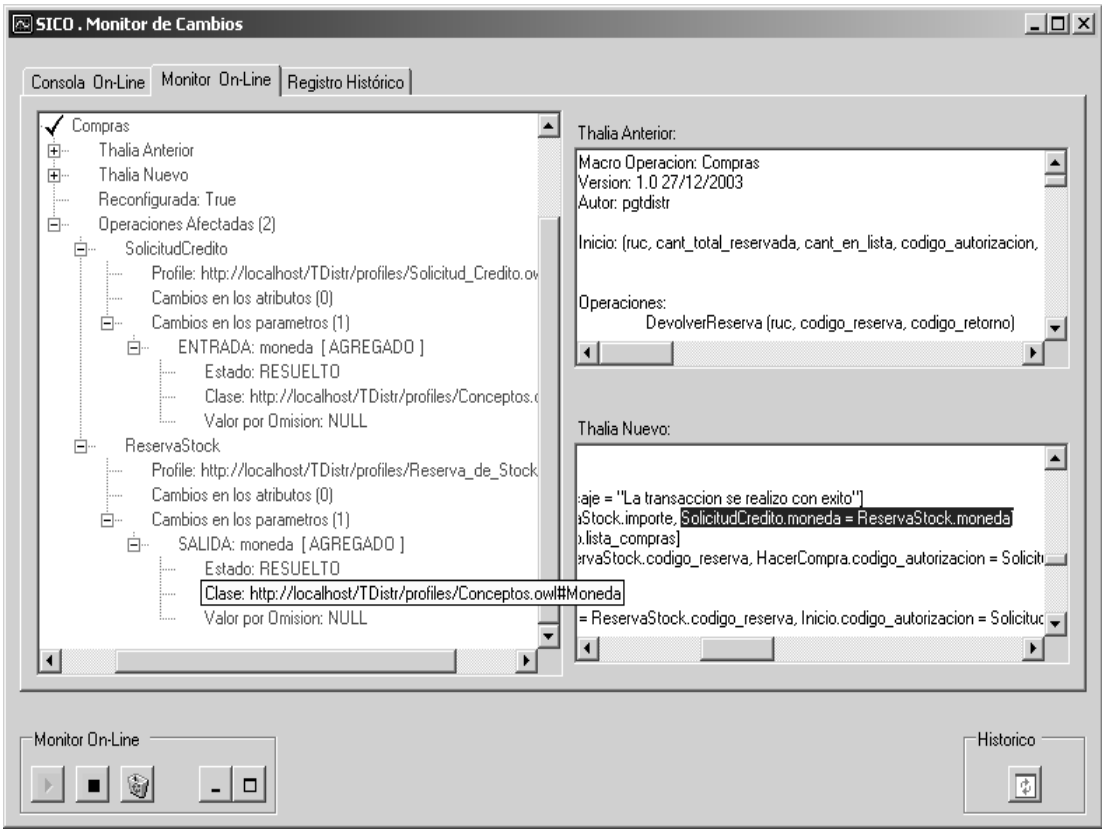

Figura V-3 : Monitor Gráfico mostrando un cambio múltiple resuelto

VI CONCLUSIONES

Situémonos en el contexto de la Web semántica y los sistemas de información orientados al business to business, en el cual los sistemas se presentan como fuentes de información, desperdigados en un marco de trabajo fuertemente desacoplado como es la word wide web. En la medida que se vaya adoptando el consumo de conjuntos de servicios integrados en procesos de negocios a nivel de web, se irá haciendo cada vez más necesario el estudio de la detección y adaptación de los sistemas a la evolución de los servicios, que ofician como fuentes de información. Este fue el cometido principal del proyecto. Sin embargo, no se obtuvo evidencia de que la comunidad académica haya abordado aún, la temática del mismo.

El uso de coreografías para modelar procesos de negocios, se observa que está fuertemente promocionado por la industria. Sin embargo esta materia se encuentra aún en la etapa de definición de estándares, que cumplan con todas las expectativas desarrolladas por las distintas tendencias que han trabajado en esta temática.

En nuestra opinión la especificación OWL-S se presenta como candidato a convertirse en el estándar que describa semánticamente a los servicios.

El hecho de haber extendido la especificación OWL-S en el contexto de este proyecto, hace a la solución extensible y compatible con cualquier otra tecnología que siga dicho estándar.

Se analizó en profundidad la casuística de la evolución de los servicios, logrando, para aquellos cambios resolubles por un programador, una automatización completa.

La política de trazabilidad implementada en la solución, permite al administrador del sistema tener el control total en todo momento, sobre los cambios ejercidos sobre la metadata.

Se logró un componente que se integra perfectamente con los ya implementados, ya que la compatibilidad hacia atrás era un objetivo planteado. En varias oportunidades se hizo una puesta en común con el grupo del proyecto

anterior en lo que respecta a los repositorios, ya que el mismo fue modificado para poder ajustarlo a los requerimientos de este proyecto.

En cuanto al lenguaje de coreografías THALIA, se reutilizó un componente de uso libre para el desarrollo de un parser.

El grafo de ejecución inscripto dentro de una coreografía, nos provee semántica acerca de la macro operación, información vital para el análisis y resolución de los cambios en los servicios.

Se trabajó sobre la especificación THALIA, pero el diseño modular de la solución nos independiza del mismo. De esta forma se podría fácilmente sustituir o extender este componente para el uso de otros lenguajes de descripción de coreografías.

La implementación de la solución se basó en la reutilización. Esto se puede ver en la utilización de estándares (OWL, OWL-S), herramientas de uso libre (GRAMMATICA) y componentes antes desarrollados como el acceso a los repositorios.

Se lograron componentes de software de uso final, no simples prototipos. La documentación asociada a ellos favorece la evolución de los mismos.

El componente RECONFIGURADOR que corre como un servicio de windows, se encuentra bajamente acoplado a la interfase gráfica de monitoreo del sistema.

VII TRABAJOS FUTUROS

Quedan planteadas las siguientes áreas de investigación:

- Diseñar la metadata semántica que describa la funcionalidad de los servicios, y no solo los atributos y parámetros de estos. Para esto se puede utilizar el potencial de OWL-S.

- Implementar una interfase gráfica para la administración de los repositorios y la metadata y extender el sub-sistema MANEJADOR DE REPOSITORIOS para implementar todas las operaciones definidas en la tesis [2].

- Investigar los factores de calidad inherentes a los servicios, que se ajusten al contexto de este proyecto.

VIII REFERENCIAS

PROYECTOS RELACIONADOS

[1] Proyecto SICO
http://www.fing.edu.uy/inco/grupos/csi/esp/Proyectos/Sico/

[2] Tesis de Maestría "Integración de Aplicaciones encapsuladas para el desarrollo de Sistemas de Información Cooperativos".
http://www.fing.edu.uy/~jabin Ing. Jorge Abin de María , Montevideo, Uruguay, año 2002.

[3] Jorge Abin, Fernando Rodriguez, Proyecto Thalia;
IX Jornadas de Informática e Investigación Operativa (2004) InCo - F. Ing. (UdelaR)
http://www.fing.edu.uy/~jabin

[4] Proyecto de Grado, Integración de Sistemas Legados usando Web Services
Biblioteca del INCO - FING.

[5] Proyecto de Grado, Transacciones Distribuidas en la Web
http://www.fing.edu.uy/~pgtdistr

[6] Proyecto de Grado, Diseño de la Metadata Semántica para un Orquestador de Operaciones Encapsuladas, Una Ontología para inferir cambios a nivel de Servicios
http://www.fing.edu.uy/~pgmetsem

WEB SERVICES

[7] UDDI
http://www.uddi.org

[8] WSDL
http://www.w3.org/TR/wsdl

ONTOLOGÍAS

[9] Ontologías de tareas y métodos
http://www.cse.ohio-state.edu/~chandra/Ontology-of-Tasks-Methods.PDF

[10] OntoWeb
http://ontoweb.aifb.uni-karlsruhe.de

RDF

[11] Tutorial RDF
http://www.w3.org/TR/rdf-primer/

[12] Tutorial RDF
http://www710.univ-lyon1.fr/~champin/rdf-tutorial/

LENGUAJES PARA EL MANEJO DE ONTOLOGÍAS

[13] OIL
http://www.ontoknowledge.org/oil

[14] DAML
http://www.daml.org

[15] Especificación DAML
http://www.daml.org/2001/03/daml+oil-index.html

[16] Especificación DAML-S
http://www.daml.org/services/daml-s/0.9

[17] Documentación de DAML-S
http://www.daml.org/services/daml-s/0.9/daml-s.html

[18] Más información sobre DAML-S
http://www.icsi.berkeley.edu/~snarayan/ISWC2002.pdf

http://citeseer.ist.psu.edu/cache/papers/cs/26319/http:zSzzSzwww.softagents.ri.cmu.eduzSzpaperszSzISWC2002.pdf/paolucci02semantic.pdf

[19] Primer versión de OWL
http://www.w3.org/2003/08/owl-pressrelease

[20] Ultima versión de OWL
http://www.w3.org/2004/OWL/

[21] OWL-S
http://www.daml.org/services/owl-s/1.0/

[22] Mas información sobre OWL-S
http://www.daml.org/services/owl-s/1.0/owl-s.html

http://www.w3.org/2001/sw/WebOnt/

http://www.w3.org/TR/owl-guide/

[23] Diferencias entre DAML y OWL
http://www.daml.org/2002/06/webont/owl-ref-proposed#appD

HERRAMIENTAS PARA EL MANEJO DE ONTOLOGÍAS

[24] Varios proyectos y aplicaciones con OWL
http://www.w3.org/2004/OWL/#projects

[25] Librería de ontologías
http://www.daml.org/ontologies/

http://www.schemaweb.info/default.aspx

[26] Protege
http://protege.stanford.edu/

[27] Jena
http://jena.sourceforge.net/index.html

[28] HP Labs – Web Semántica
http://www.hpl.hp.com/semweb/

[29] DAML dotNET API
http://www.daml.org/2002/10/dotnetAPI/

[30] SNOBASE
http://www.alphaworks.ibm.com/tech/snobase

[31] AKT Portal
http://www.aktors.org/akt/

[32] BioPax
http://www.biopax.org/

[33] RACER
http://www.sts.tu-harburg.de/~r.f.moeller/racer/

[34] JTP
ftp://ksl.stanford.edu/pub/jtp

GRAMATICAS

[35] Biblioteca de uso libre, Grammatica
http://www.nongnu.org/grammatica/

[36] Curso de teoría de lenguajes
http://www.fing.edu.uy/inco/cursos/teoleng/

IX ANEXOS