Universidad de Buenos Aires

Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Tesis

Correlación y excepciones en lenguajes de orquestación de servicios

Christian Roldán
Director: Hernán Melgratti

Diciembre 2011

Dirección: Pabellon I, Ciudad Universitaria – C1428EGA
Tel: (54.11) 4576-3359 — Conmutador: (54.11) 4576-3390 al 96 (int 701/702) —
E-mail: croldan86@gmail.com

Agradecimientos

A Hernán, por acompañarme todo este tiempo, formarme e incluso brindarme su amistad. La cantidad de horas que dedicó para que hoy esté acá, difícilmente puedan ser agradecidas en un párrafo. Un gran director pero sobre todas las cosas, una gran persona.

A Eduardo y Guido, por sus ganas de formar parte de este momento, su compromiso, comentarios y sugerencias fueron importantísimos para este trabajo.

A Hugo y Mónica, mis viejos. Ellos, que además de darme la vida me dieron la oportunidad de estudiar, apoyando cada de decisión tomada y siempre al lado mío.

A mis tres hermanos, por darme sus palabras de aliento y mostrarme que ellos iban a estar siempre conmigo.

A Yesi, por ser mi compañera en esta vida y tomarme de la mano bien fuerte cuando más necesitaba, por las risas y lágrimas compartidas todos estos años. No me alcanzaría la vida para compensarle todos estos años al lado mío.

A mi Tío y Mono, por contarme lo gratificante y divertido que es estudiar en Exactas. Por su frase: "En exactas te van a enseñar a pensar".

A Lea, por estar al tanto de cada examen y hacer un esfuerzo terrible por entender el plan de carreras, el contenido de cada materia. El amigo que siempre mandó un mensaje antes de cada examen.

A Nati y Nico, los amigos que me dejó esta carrera. Horas y horas haciendo trabajos prácticos con Nico, viajando de Zona Oeste a Zona Sur, creyendo que no llegabamos y después... Final feliz para ambos. Nati, que si bien fue siempre más adelante que yo, siempre estuvo ahí, para darme un consejo, una palabra de aliento y una práctica resuelta por ella misma.

A Juli, Tom, Alex, Fede, Kivi, Susana, Lucho, Ferpa y Marcos personas que conocí en diferentes momentos de la carrera y fueron importantes para estar hoy acá.

Resumen

Los conjuntos de correlación son primitivas que permiten identificar instancias en los lenguajes de orquestación de servicios. Un conjunto de correlación es un conjunto de propiedades usado para asociar el mensaje que un servicio recibe con alguna instancia activa que el servicio haya creado. De esta manera, cada vez que un servicio recibe un mensaje, este determina a qué instancia del servicio debería entregarle el mensaje en cuestión. A partir de una implementación concreta para WS-BPEL, propondremos un modelo formal que describa el uso de conjuntos de correlación y las excepciones asociadas a estos. En este trabajo estudiaremos diferentes sistemas de tipos que nos permiten asegurar que los programas construidos son libres de excepciones de correlación.

Índice general

Índice de figuras

Capítulo 1

Introducción

La noción de instancia de un servicio es un concepto clave cuando se trabajan con servicios. En tiempo de ejecución, son las instancias de los servicios las que interactúan. Cada servicio se define, a partir de un flujo de actividades que es usado para la creación de instancias. Cuando el servicio recibe un mensaje de entrada que se corresponde con una actividad de inicio de su definición, procede a la creación de una nueva instancia. De esta manera, una instancia puede ser descripta como el residuo de la ejecución parcial de ese flujo de actividades. Por ejemplo, un servicio que maneja órdenes de compra crea una instancia nueva cuando recibe una nueva orden de compra. Todos lo mensajes posteriores deberán ser entregados a la instancia ya creada, la cual deberá ser identificada de alguna manera. Por ejemplo, cuando el cliente envía el pago de la orden de compra, el mensaje debería ser entregado a la instancia correcta. Los lenguajes de orquestación (como la especificación WS-BPEL [5]) ofrecen diferentes alternativas para identificar instancias como por ejemplo Dynamic Endpoint (se puede consultar en WS-ADDRESSING [4]) o conjuntos de correlación. La idea principal de los conjuntos de correlación es que los mensajes llevan una identificación de la instancia a la que deben ser entregadas. De esta manera, cuando un servicio recibe un mensaje, compara el valor del mensaje recibido con la identificación asociada a cada instancia, entregando el mensaje a la instancia que corresponda. Existen en la bibliografía algunos modelos formales como SOCK [7], COWS [10], y BLITE [11] que se encargan de describir el manejo de correlaciones. Sin embargo, tales modelos no mencionan nada acerca de las excepciones relacionadas a los conjuntos de correlación que sí aparecen en la especificación de BPEL y son en este sentido, incompletos.

1.1. Objetivos

En este trabajo estudiaremos formalmente la relación que existe entre los conjuntos de correlación y las excepciones vinculadas a estos. Para ello, presentaremos un modelo formal, en forma de cálculo de procesos, que refleja el comportamiento

de una implementación particular de WS-BPEL. Esta construcción usará de base, modelos existentes como SOCK, COWS y BLITE. A diferencia de estos cálculos no buscaremos realizar una formalización íntegra de todo un lenguaje de orquestación ya que estos lenguajes son complejos e incluyen muchas primitivas (por ejemplo, compensaciones, operaciones two-way, manejo de errores y terminación) que no están estrechamente vinculadas al mecanismo de correlación. En consecuencia nos limitaremos a formalizar el núcleo de primitivas afectadas por el mecanismo de correlación.

Existen diferentes implementaciones de WS-BPEL como Oracle BPEL Process Manager [17], ActiveBPEL Engine [15] y Apache ODE [16] las cuales difieren de manera considerable(ver artículo [11]). Por este motivo, al diferir de nuestro cálculo utilizaremos como referencia tanto la especificación BPEL como sus implementaciones.

Para ello diseñaremos una serie de programas que nos permitirán inferir el comportamiento de los conjuntos de correlación y las excepciones en dichas implementaciones.

Finalmente, buscaremos encontrar una manera de caracterizar programas que gocen de ciertas propiedades de interés en el uso de las primitivas de correlación.

1.2. Contribución de la tesis

A partir de este trabajo, se construyeron diferentes servicios de prueba que permitieron inferir el comportamiento real de los mecanismos de correlación de una implementación particular de BPEL. En particular, el resultado observado nos permitió identificar puntos donde la especificación resulta ambigua, o poco clara y la implementación particular BPEL responde a veces de manera no esperada (por ejemplo, la identificación de instancias no es unívoca).

Se desarrolló un modelo formal de correlaciones que explica la interacción del mecanismo de correlación y excepciones asociadas. Asimismo el modelo exhibe características únicas respecto a otras propuestas como SOCK, COWS y BLITE, y se corresponde más fielmente con el comportamiento de implementaciones tales como ActiveBPEL. El modelo formal se presenta como un cálculo de procesos capaz de describir la naturaleza de las excepciones que se asocian a los mecanismos de correlación.

Se formaliza la noción de servicios libre de excepciones evitables, es decir, servicios que garanticen que no arrojarán excepciones. Se propone un sistema de tipos simple que permite caracterizar aquellos servicios diseñados de manera tal que evitan arrojar excepciones asociadas al mecanismo de correlación. El resultado principal de la tesis es mostrar que un servicio definido como bien tipado es libre de excepciones evitables.

Los resultados de este trabajo aparecen publicados en acta de conferencia como: *Melgratti, H and Roldán, C. On correlation sets and correlation exceptions in ActiveBPEL. Proceedings de 6th International Symposium on Trustworthy Global*

Computing (TGC 2011). Aachen, Germany, September 9-10, 2011 (aparecerá como Lecture Notes in Computer Science).

1.3. Estructura de la tesis

La tesis está organizada de la siguiente manera. En el capítulo 2 definimos las nociones de servicios, conjuntos de correlación y excepciones, conceptos indispensables para abordar el problema. Luego describimos la especificación WS-BPEL y comentamos algunas de sus implementaciones más conocidas. Finalmente, mencionaremos algunas propuestas existentes en la literatura para definir formalmente la semántica de BPEL (haciendo particular énfasis en los cálculos de procesos). En el capítulo 3 mostramos las conclusiones principales de la evaluación de la implementación del mecanismo de correlación en una implementación concreta de BPEL (ActiveBPEL). El capítulo 4 presenta el cálculo de procesos, llamado Cab. Se define su sintaxis y la semántica operacional en términos de un sistema de transiciones etiquetadas e ilustraremos mediante ejemplos las principales características del cálculo. En el capítulo 5 definiremos la noción de servicios libres de excepciones de correlación y daremos un sistema de tipos que los caracteriza estáticamente. Por último, en el capítulo 6 presentaremos las conclusiones y hablaremos sobre trabajos futuros.

Capítulo 2

Servicios, Correlaciones y Excepciones

Servicios, conjuntos de correlación y excepciones son conceptos claves de nuestro trabajo, necesarios para entender el problema que nos proponemos estudiar. En este capítulo introduciremos cada uno de estos conceptos y además mencionaremos trabajos previos aparecidos en la literatura que fueron utilizados como base de nuestra investigación.

2.1. Servicios

Un **servicio web**, según la W3C, es un sistema de software diseñado para permitir la interoperatibilidad máquina a máquina en una red [1], definiendo interoperatibilidad como la capacidad de un programa para acceder a múltiples sistemas diferentes. En términos más simples, diremos que un servicio web es un sistema de software que permite la comunicación entre diferentes máquinas, con diferentes plataformas y entre programas distintos.

Esta comunicación se logra a través de la adopción de los siguientes estándares abiertos:

- XML (eXtensible Markup Language) es un lenguaje que permite estructurar, almacenar y transferir información. Separa la estructura del contenido y permite que el desarrollo sea modular. Al igual que el HTML, se basa en un texto plano y etiquetas, con la diferencia de que XML define las etiquetas en función del tipo de dato que está describiendo y no, como en HTML, a la apariencia final que tendrán en pantalla.

- SOAP (Simple Object Access Protocol) es un protocolo sin estado de intercambio unidireccional de mensajes, sobre una red (como HTTP, MIME, SMTP). Las aplicaciones pueden crear patrones de interacción (por ejemp-

lo, petición/respuesta, petición/respuestas múltiples, etc.) pero siempre en un solo sentido.

- WSDL (Web Services Description Language) es un lenguaje basado en XML para describir servicios web. Permite describir la interfaz pública de los servicios web, especificando los protocolos y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo.
- UDDI (Universal Description, Discovery and Integration) es uno de los estándares básicos de los servicios web diseñado para ser interrogado por mensajes SOAP y proveer acceso a documentos de WSDL. Es un repositorio en el cual podemos buscar cuáles son los servicios web disponibles.

En general nos referimos a servicios web cuando hablamos de clientes y servidores que se comunican entre sí usando mensajes XML que siguen el estándar SOAP. Los servicios emplean una infraestructura que proporciona un mecanismo para localizar servicios Web y obtener una descripción de tales servicios que definen cómo se usan y los formatos estándar de conexión con los cual comunicarse.

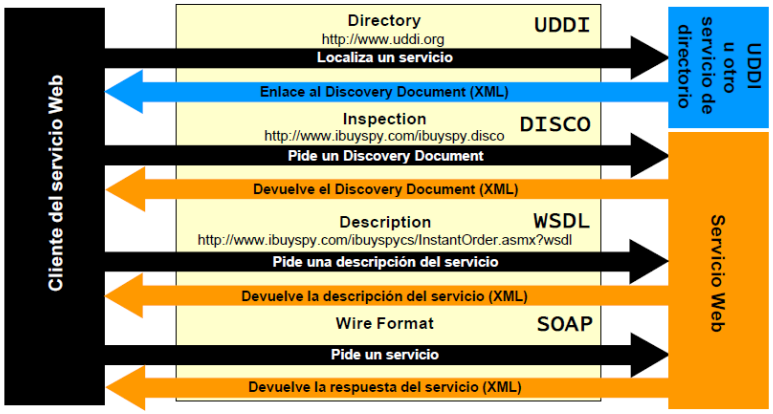

Figura 2.1: Infraestructura Servicio Web

Finalmente, un servicio web tendrá como objetivo implementar una determinada lógica de negocio y publicar una interface que otros clientes (posiblemente servicios) puedan invocar. Una representación gráfica de la arquitectura se muestra en la Figura 2.1.

2.2. Composición de Servicios

Una ventaja importante de los servicios web es que posibilitan una visión modular para resolver problemas de negocio. Existen escenarios, dentro de las problemáticas de negocio, donde contar con un único servicio web no alcanza para satisfacer las

necesidades que el negocio plantea. Por consiguiente, es necesario integrar varios servicios web simples en un servicio compuesto que permita responder a los requerimientos planteados. La tarea de componer servicios para obtener otros más complejos se conoce como *Composición de Servicios*.

Existen dos modelos para describir servicios compuestos:

1. **Orquestación** define la implementación de un servicio en particular. En la orquestación existe un mecanismo de control centralizado responsable de la invocación y la colaboración entre servicios. Con la orquestación, el proceso es siempre controlado desde la perspectiva de una de las partes del negocio.

2. **Coreografía** se centra en la colaboración entre servicios, describiendo de manera formal el intercambio de mensajes (sincrónicos o asincrónicos) y el papel que juega cada servicio en la interacción. Es importante notar que esta técnica oculta detalles internos de implementación.

La orquestación representa siempre el control desde la perspectiva de una de las partes. Esto lo distingue de la coreografía, que se centra en definir la colaboración y permite a cada parte involucrada describir su papel en la interacción. La orquestación y coreografía intentan describir aspectos relacionados con la creación de procesos de negocio que involucran varios tipos de Software, por ejemplo sistemas de planificación de recursos empresariales o ERP (por sus siglas en inglés, Enterprise resource planning), aplicaciones web, etc.

Visto gráficamente (ver Figura 2.2), la orquestación, permite diseñar procesos de negocio ejecutables que pueden interactuar (a nivel de mensaje) tanto con software interno como externo. Por otra parte, la coreografía es mucho más colaborativa, ya que permite trazar las secuencias de mensajes que intercambian los participantes del proceso de negocio en lugar de centrarse en los mensajes que intercambian los diversos programas de software que implementan al proceso de negocio.

Figura 2.2: Orquestación−Coreografía

Consideremos el escenario dónde un cliente encarga un pedido a cierto proveedor. Para poder llevar a cabo el pedido, el proveedor deberá consultar con el almacén para saber sí hay stock suficiente. En caso de existir stock se confirmará el pedido,

en caso contrario se informará que no puede realizarse el pedido por falta de stock. Una vez confirmado el pedido, el cliente deberá realizar el pago al proveedor, el que finalmente avisa al depósito para que realice el envío del pedido.

En este ejemplo, estamos dando una visión de la composición según el enfoque de la orquestación ya que sólo describimos acciones de ejecución del lado del *Proveedor* como se ve en la Figura 2.3.

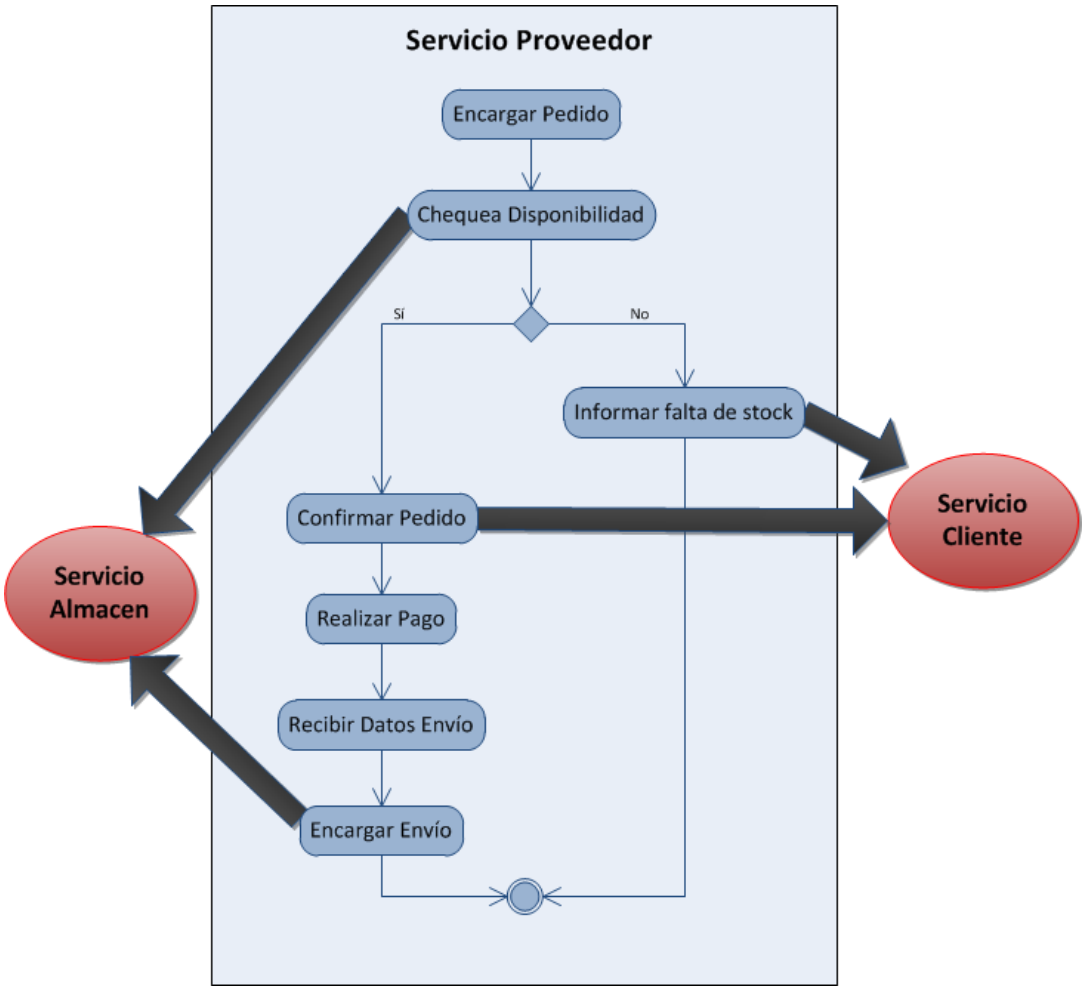

Figura 2.3: Ejemplo para la compra de producto. Visión de Orquestador

Siguiendo el mismo ejemplo, la coreografía nos permite describir los mensajes que se envían el cliente, el proveedor y el almacén como se muestra en la Figura 2.4. Además permitirá reutilizar cada uno de estos servicios en diferentes aplicaciones, requiriendo implementar reglas de coordinación distintas para cada aplicación.

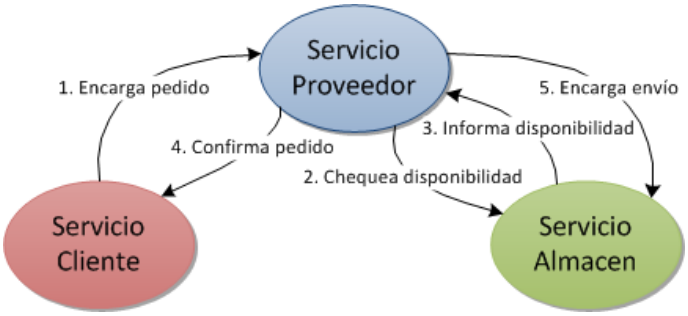

Figura 2.4: Ejemplo para la compra de producto. Visión Coreográfica

2.3. WS-BPEL

WS-BPEL es un lenguaje de orquestación estandarizado por OASIS (Organisation for the Advancement Structured Information Business Transaction Protocol) para la composición de servicios Web que permite a los desarrolladores crear programas que automatizan las interacciones entre los servicios Web, jugando un papel clave en las Arquitecturas Orientadas a los Servicios (por sus siglas en inglés, SOA). Para comprender su funcionamiento, imaginemos un proceso de negocio determinado, que tiene una entrada A y una salida B. Dicho proceso, como la mayoría de ellos, posee a su vez muchos procesos internos que se van activando de acuerdo a ciertos parámetros y valores de entrada. Un proceso BPEL sería el encargado de orquestar, y por lo tanto dirigir la ejecución de tales procesos internos de una manera ordenada. La ventaja de contar con un estándar para BPEL radica en que puede ser implementado en una gran variedad de motores de ejecución de diferentes proveedores.

BPEL es un formato XML que proviene de la convergencia entre el WSFL (Web Service Flow Language) de IBM y XLANG de Microsoft.

El lenguaje permite definir un proceso dando el orden de ejecución de un conjunto de actividades, los socios (partners) involucrados en el proceso y los mensajes que se intercambian los socios.

Las primitivas para la construcción de un proceso BPEL permiten declarar variables, socios, actividades, conjuntos de correlación, compensaciones, manejo de eventos y actividades. Las variables se utilizan para almacenar valores. Los socios representan los servicios restantes que participan en la composición de servicios. A partir de estos se especifica el rol que juega cada parte.

BPEL está basada en un modelo llamado “workflow”que especifica el orden en que se ejecutan los conjuntos de actividades. Existen dos tipos de actividades, básicas y estructuradas. Las actividades básicas son actividades que no contienen otras actividades (acciones atómicas), representan un paso en el ciclo de vida del proceso. Algunas actividades básicas son:

- *receive*: espera la llegada de un mensaje.

- *invoke*: invoca una operación de algún socio. Esta operación puede ser $one-way$ (define una variable de entrada) ó $request-response$ (define una variable de entrada y una de salida).
- *assign*: asigna un nuevo valor a una variable.
- *reply*: genera y envía la respuesta de una operación de invoke asociada.
- *wait*: espera un cierto lapso de tiempo.

Las actividades estructuradas permiten estructurar y definir actividades compuestas, como por ejemplo:

- *sequence*: contiene una colección de actividades que deben llevarse a cabo de manera secuencial.
- *flow*: especifica actividades que pueden ejecutarse en paralelo.
- *if*: permite especificar la ejecución de una actividad dependiendo del valor de una condición.
- *pick*: permite que el proceso espere la ocurrencia de un evento (llegada de algún mensaje).
- *while*: un ciclo contenedor de actividades que se ejecutan mientras el valor de una condición evalúa verdadero.

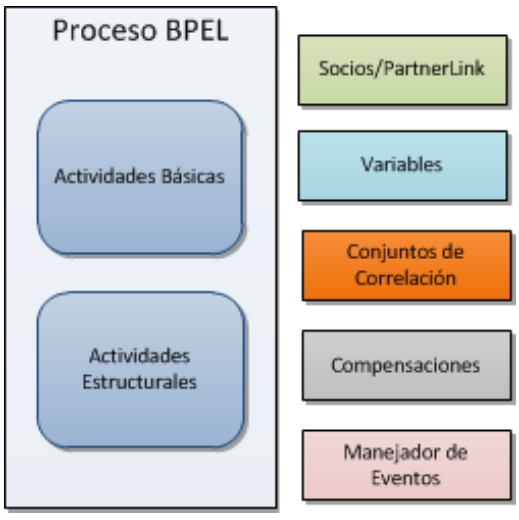

Figura 2.5: Un proceso en BPEL

Compensaciones y manejo de eventos es algo que queda afuera del alcance de este trabajo, pero puede consultarse en [5]. Los conjuntos de correlación, eje de este trabajo, es un tema que abordaremos con mayor profundidad en el capítulo 5.

2.4. Servicios en WS-BPEL

Un servicio en un lenguaje de orquestación puede verse de manera análoga a la definición de una clase en el paradigma de objetos. Así como una clase es una estructura que especifica propiedades y métodos, el servicio es una estructura que especifica el orden de actividades a ejecutar. Por otra parte, una clase es responsable de crear objetos a partir de sus constructores, mientras que un servicio será responsable de crear *instancias* a partir de alguna actividad inicial de su definición.

Un concepto fundamental en la orquestación de servicios resulta ser la creación e identificación de instancias, y esto se debe al hecho de que muchas instancias de un servicio pueden concurrentemente interactuar con diferentes clientes. Cada servicio ofrece un protocolo de mensajes diseñado para crear instancias de forma tal que cuando un servicio recibe un mensaje que coincide con una actividad de inicio de su definición, procede a crear una nueva instancia.

Consideremos un servicio que dado dos naturales se encarga de devolver la suma de sus cuadrados. Para esto, el servicio cuenta en su definición con una secuencia de actividades básicas como se muestra en la Figura 2.6−1. Notar que el resto de las primitivas (como los conjuntos de correlación) no son incluídos en este ejemplo. La actividad inicial se llama *Recibir SumarCuadrados* y se corresponde con la primitiva *receive* de la especificación de WS-BPEL. Esta actividad será la encargada de crear instancias a partir de dos naturales pasados como parámetros. Cuando el servicio recibe el mensaje para *SumarCuadrados*, este se encarga de crear una instancia reemplazando las variables de estado con los valores 2 y 3 como se muestra en la Figura 2.6−2.

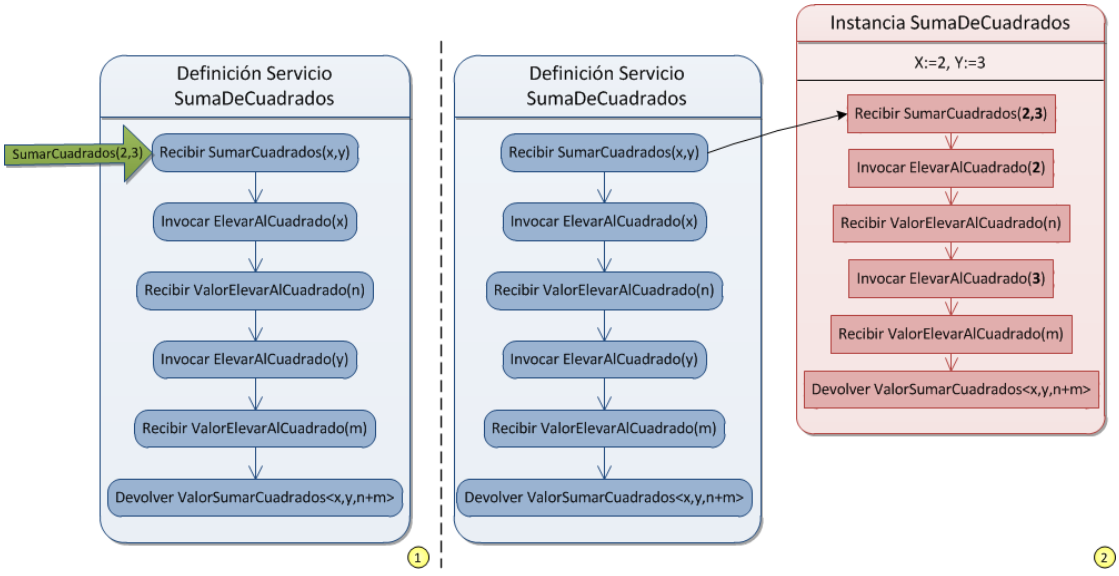

Figura 2.6: Creación de Instancias

Consideremos el caso en que el servicio recibe el mensaje *Recibir ValorElevar-*

AlCuadrado sin existir una instancia activa. Al no existir una instancia activa, no hay quién reciba el mensaje, por lo tanto el mensaje no puede ser entregado. Sin embargo el mensaje recibido no se ignora, la entrega se realizará en el momento que la instancia necesite ejecutar la actividad asociada a dicho mensaje. De esta manera, luego que la instancia ejecute la actividad *Invocar ElevarAlCuadrado*, el servicio entregará el mensaje guardado a la instancia correspondiente.

2.5. Conjuntos de Correlación

En el paradigma de objetos la colaboración que existe entre los objetos se lleva a cabo a través del envío de mensajes. Para procesar los mensajes es necesario hallar la declaración del método que se pretende ejecutar, el proceso de establecer la asociación entre el mensaje y el método a ejecutar se llama *method dispatch*. De manera análoga, los servicios se comunican entre sí a partir del intercambio de mensajes. Cuando un servicio recibe un mensaje deberá establecer un mecanismo que asocie el mensaje recibido con la instancia que puede procesar dicho mensaje. Los lenguajes de orquestación (como el estándar WS-BPEL [5]) ofrecen diferentes alternativas para identificar instancias, como por ejemplo Dynamic Endpoint(se puede consultar en WS-ADDRESSING [4]) o conjuntos de correlación. La idea principal detrás de los conjuntos de correlación es poder llevar, a través de mensajes (llamados mensajes correlacionados), la identificación de la instancia con la que se está interactuando. De esta manera, cuando un servicio recibe un mensaje correlacionado, el servicio compara el valor de correlación con las instancias que tiene activas, si existe alguna que coincide con el valor, se le entrega el mensaje.

Los conjuntos de correlación sólo pueden iniciarse una única vez durante la vida útil de la instancia del servicio al que pertenece, como son las constantes en los lenguajes de programación estructurados. Una vez iniciado, el conjunto de correlación *debe* conservar sus valores, considerándolos como parte de la identidad de la instancia de un proceso de negocio.

Los conjuntos de correlación se pueden utilizar en todas las actividades de intercambio de mensajes ($< receive >, < reply >, < onMessage >, < onEvent > y < invoke >$). Estos cuentan con una propiedad llamada *initiate* que puede tomar los valores:

- *yes*: indica que debe inicializar la variable de correlación.
- *no*: indica que no debe inicializar la variable de correlación (la misma debe haber sido inicializada previamente).
- *join*: indica que se debe inicializar la variable de correlación si aún no fue inicializada.

Retomando el ejemplo del servicio que permite calcular la suma de cuadrados a partir de dos naturales, veamos qué ocurre cuando existe más de una invocación.

La definición del servicio cuenta con una secuencia de actividades donde la primera actividad de *receive* se encarga de crear la instancia e inicializar las variables de estado a partir de los parámetros formales (como se mostró en la Figura 2.6 de la Sección 2.4). Además, se utiliza un servicio *Cuadrado*, encargado de elevar un número al cuadrado.

Luego de haberse creado la instancia a partir del mensaje $SumarCuadrados(2, 3)$, se invoca al servicio *Cuadrado* con el parámetro formal 2. Este se encargará de devolver el valor 4. Supongamos ahora que el servicio recibe el mensaje $SumarCuadrados(4, 5)$. Al recibir este mensaje, el servicio creará una nueva instancia para calcular la suma de los cuadrados de 4 y 5, existiendo de esta manera dos instancias del servicio *SumaDeCuadrados* como se muestra en la Figura 2.7.

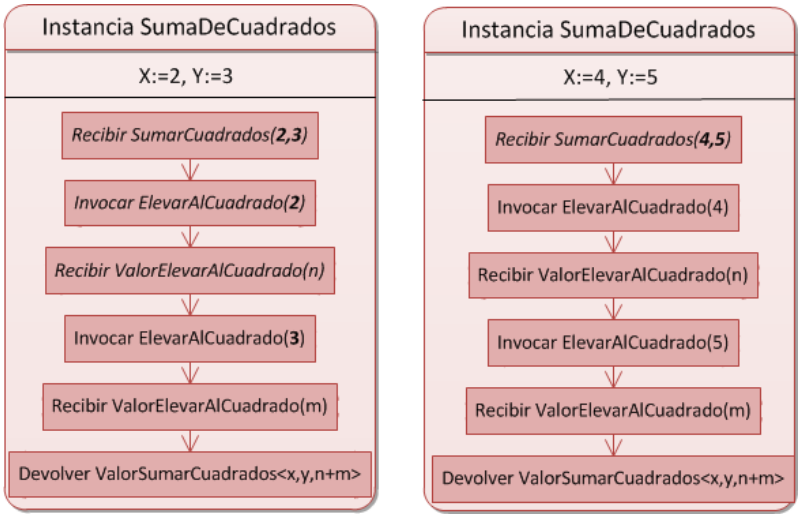

Figura 2.7: Instancias Concurrentes del servicio SumaDeCuadrados

Estas dos instancias se encargaran de invocar al servicio *Cuadrado*, a partir de los mensajes $ElevarAlCuadrado(3)$, correspondiente a la primera instancia creada, y $ElevarAlCuadrado(4)$, correspondiente a la segunda instancia creada. Estos dos mensajes crearán dos instancias del servicio cuadrado (como se muestra en la Figura 2.8).

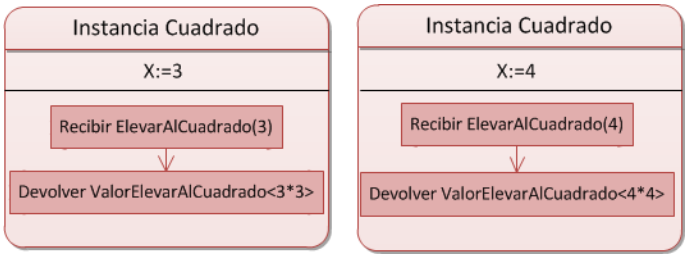

Figura 2.8: Instancias Concurrentes del servicio Cuadrado

Supongamos ahora que tales instancias mediante las operaciones $ValorElevarAlCuadrado$ retornan los resultados. Al no existir una manera de identificar instancias en el servicio *SumaDeCuadrados*, puede ocurrir que los mensajes de respuesta se intercambien, devolviendo 9 a la instancia que la invocó con el parámetro formal inicializado en 4 y 16 a la instancia que la invocó con el párametro formal inicializado en 3 (ver Figura 2.9). Este problema surge por no tener un mecanismo que nos permita identificar destinatarios de los mensajes.

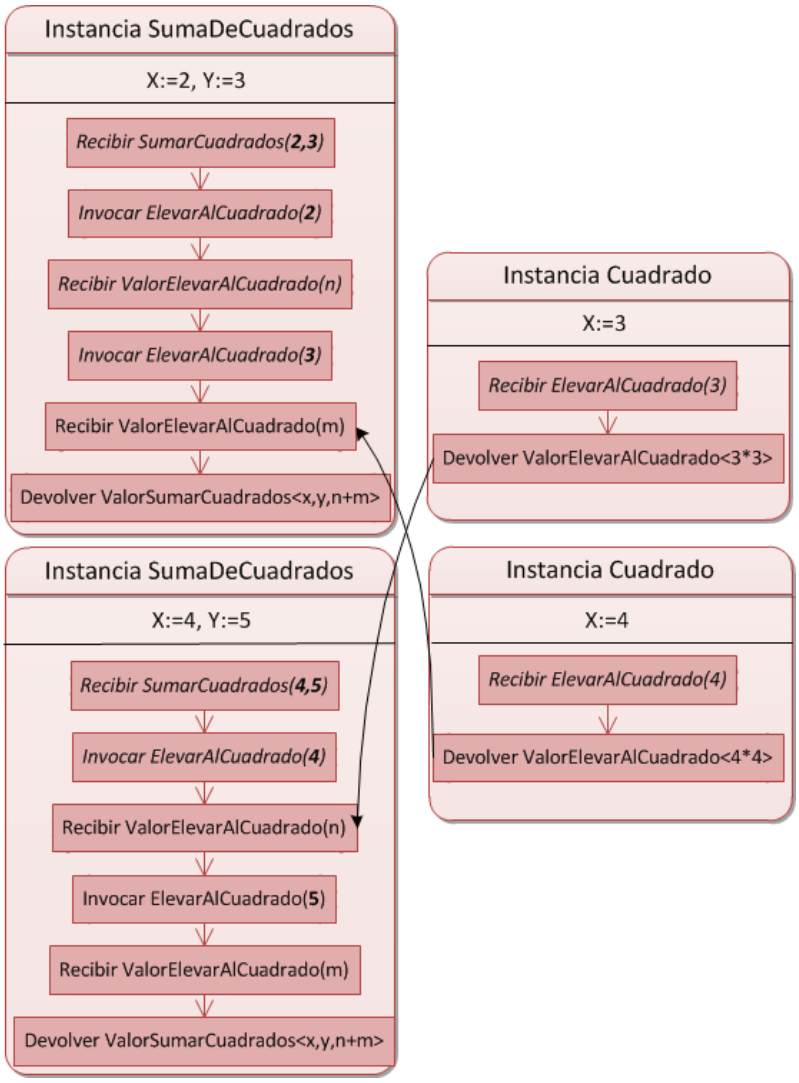

Figura 2.9: Invocaciones concurrentes sin correlaciones

La manera de solucionar este problema es a través del uso de primitivas de correlación. Construyamos los servicios SumaDeCuadrados y Cuadrado de forma tal, que cada uno identifique las instancias por medio de los conjuntos de correlación. Para SumaDeCuadrado podemos definir una variable de correlación (por ejemplo

id) ó bien utilizar las dos variables de estado (es decir, x e y) que además nos sirvan para identificar la instancia. De manera análoga para el servicio Cuadrado. Dado que los valores de los parámetros formales nos alcanzan para identificar las instancias de SumaDeCuadrados y Cuadrado utilizaremos estas variables como variables de correlación.

Entonces, la instancia de Cuadrado identificada por el valor 3 devolverá 9 a la instancia del servicio SumaDeCuadrados que la haya invocado con el valor 3, en este caso será la instancia identificada por los valores de correlación $x := 2$ e $y := 3$. De manera análoga para la instancia de Cuadrado identificada por el valor 4. En la Figura 2.10 se muestra la definición de los servicios con variables de correlación (notar que los nombres de las variables de correlación están subrayados) y la entrega correcta de mensajes.

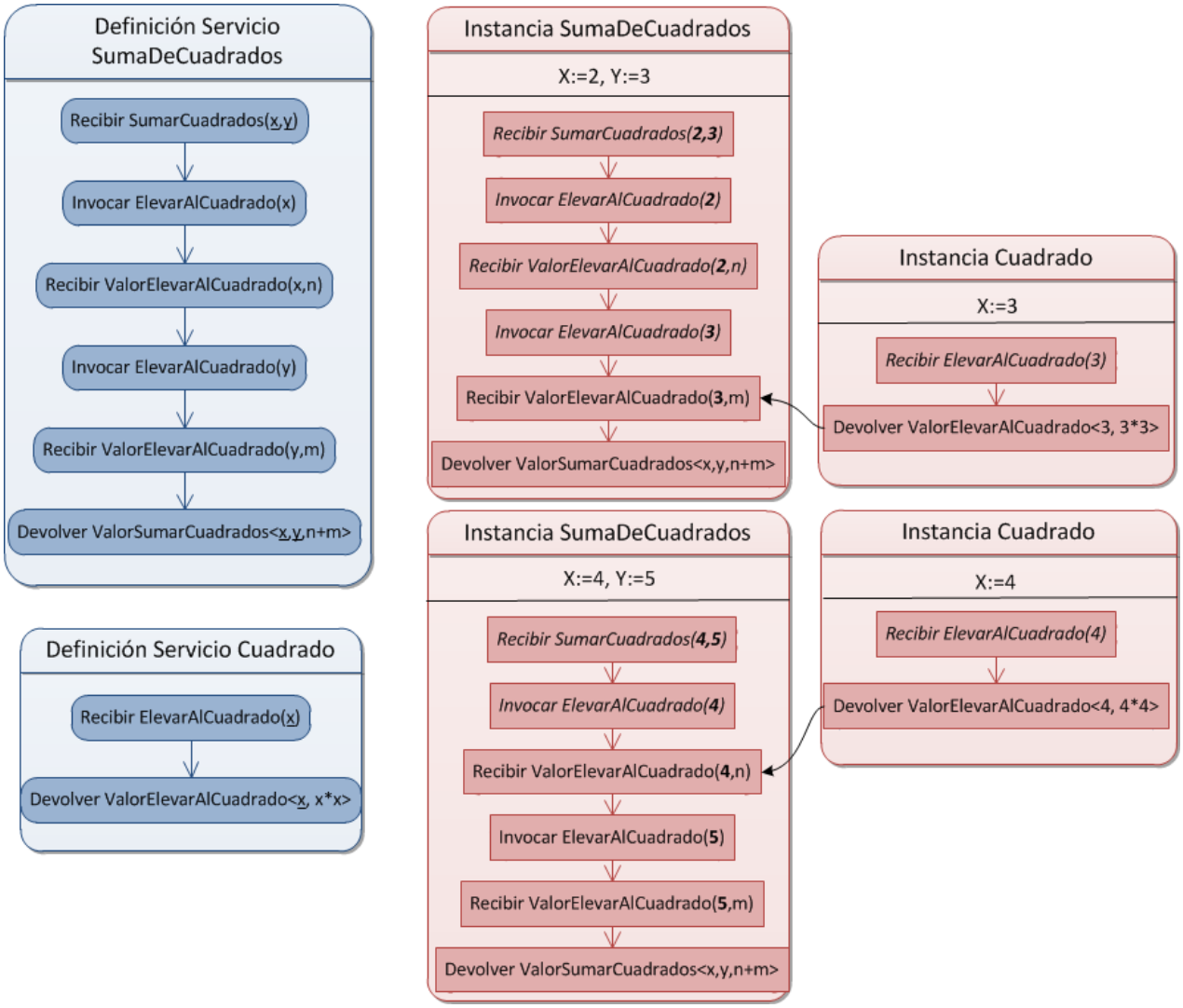

Figura 2.10: Invocaciones concurrentes usando correlaciones

2.6. Excepciones

Algunos lenguajes de programación cuentan con la posibilidad de manejar o controlar los errores que se producen en tiempo de ejecución a través del *manejo de excepciones*. Estos proveen una forma estructurada de atrapar situaciones inesperadas, como también errores predecibles o resultados inusuales. Todas esas situaciones son llamadas excepciones.

Para manejar situaciones anómalas en el manejo de correlaciones, el mecanismo de correlación provee un conjunto de excepciones específicas. Por un lado excepciones relacionadas con las operaciones que pueden inicializar el conjunto de correlación, como es el caso de la excepción *CorrelationViolation*. Por otro lado, existen excepciones que están relacionadas con el orden de activación de tareas, como por ejemplo las excepciones *Conflicting Receive* y *Ambiguous Receive*.

En la sección anterior mencionamos que los conjuntos de correlación llevan una propiedad llamada *initiate* que indica si la actividad debe o no inicializar la variable de correlación. La especificación dice que la excepción *CorrelationViolation* debe ser arrojada cuando:

1. La propiedad *initiate* está seteada en *yes* y la variable de correlación fue ya inicializada.

2. La propiedad *initiate* está seteada en *no* y ninguna actividad previa inicializó dicha variable.

La excepción *Conflicting Receive* ocurre cuando se encuentran en paralelo dos actividades que pueden realizar la misma acción. La especificación de WS-BPEL dice concretamente que: "*Si dos o más acciones de receive para el mismo socio, nombre de operación y conjunto de correlación son activadas simultáneamente durante la ejecución, entonces la excepción bpel:conflictingReceive DEBE ser arrojada*".

Por otro lado *Ambiguous Receive* ocurre cuando una instancia activa concurrentemente dos o más actividades que pueden recibir un mismo mensaje pero difieren en las variables de correlación. Según la especificación de WS-BPEL, "*Si una instancia de proceso de negocio activa simultáneamente dos o mas IMAs [inbound message activities] para el mismo socio, operación pero diferente conjunto de correlación, y la correlación de múltiples de estas actividades igualan a un mensaje de entrada, entonces la excepción bpel:ambiguousReceive DEBE ser arrojada por todos los IMAs cuyos conjuntos de correlaciones igualen al mensaje entrante*".

La principal diferencia entre Conflict Receive y Ambiguous Receive, es que la primera ocurre cuando una instancia activa en paralelo dos acciones idénticas que pueden ejecutarse de manera no determinística mientras que la segunda necesita que las actividades en paralelo tengan diferentes variables de correlación inicializadas con el mismo valor y la excepción ocurre al momento de invocar a cualquiera de las dos actividades. Ambiguos Receive ocurre en el momento que se procesa el mensaje que podría ser manejado indistintamente por cualquiera de las actividades en conflicto.

Para entender mejor esto, supongamos que tenemos un servicio encargado de publicar un artículo científico para una revista. Para la publicación necesita que previamente dos científicos revisores realicen la revisión del artículo. Estos revisores leerán separadamente el artículo y tomarán una decisión individual acerca de si el artículo merece ser publicado o no.

El orquestador se encargará de invocar a los aprobadores a partir de la invocación secuencial del servicio encargado de aprobar el artículo. Una vez realizada las dos invocaciones, se esperará que cada aprobador determine si aprueba o no el artículo. Para esto tendremos dos actividades en paralelo (*recibirAprobacion*) con la variable de correlación *aprobador*. Una vez recibidas las aprobaciones se invoca al autor del artículo para informarle el estado del mismo. Gráficamente:

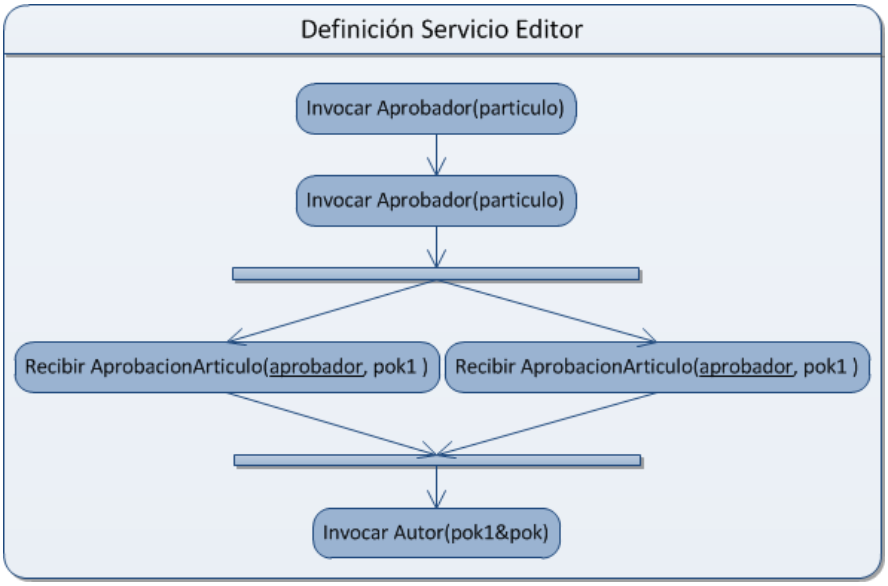

Figura 2.11: Excepciones Conflicting Receive

El problema que nos encontramos es que realizada la segunda invocación quedan en paralelo dos actividades semánticamente iguales, y eso produce que se arroje la excepción Conflicting Receive.

En cambio, si utilizamos dos variables de correlación distintas para identificar a cada aprobador (*aprobador*1 y *aprobador*2) tendríamos dos actividades distintas. Sin embargo, supongamos que los dos aprobadores se identifican con el mismo valor de correlación, en este caso cuando intenten aprobar el artículo nos encontraremos con dos actividades concurrentes cuyo nombre de operación es el mismo, el socio con el cual están colaborando es el mismo y el valor asociado a la instancia también. Por lo tanto la excepción Ambiguous Receive es arrojada.

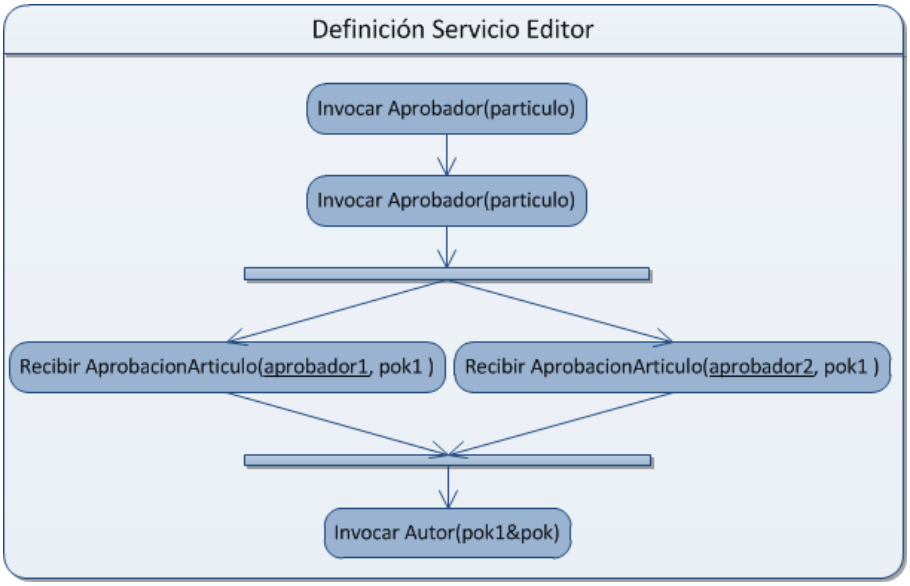

Figura 2.12: Excepciones Ambiguous Receive

2.7. Implementaciones para BPEL

El estándar BPEL cuenta con diversas implementaciones desarrolladas por distintas organizaciones. Sin embargo, se sabe que tales implementaciones presentan diferencias notorias de comportamiento. En esta sección comentaremos algunas de estas diferencias que han sido reportadas en [11] donde se comparan distintas implementaciones para BPEL considerando aspectos principales del estándar (como ser conjuntos de correlación, manejo de compensaciones, etc) y un conjunto de ejemplos que han sido especificados formalmente en BLITE [11]. Este trabajo considera: Oracle BPEL Process Manager [17], ActiveBPEL Engine [15] y Apache ODE [16]. Remarcamos que Apache ODE y ActiveBPEL son proyectos *open source* mientras que Oracle BPEL Process Manager es distribuido bajo la licencia Oracle Technology Network Developer.

A continuación presentamos informalmente ejemplos utilizados para la comparación y las conclusiones reportadas, en particular, las referidas al manejo de instancias:

1. Un proceso definido a partir de dos actividades de *receive* consecutivas que tienen la misma variable de correlación.

   a) Oracle BPEL Process Manager se encargará de crear dos instancias, una instancia por cada *receive*.

   b) Apache ODE creará una única instancia.

   c) ActiveBPEL actuará de manera no determinística, es decir, tendremos escenarios donde creará dos instancias (como el caso de Oracle BPEL) y escenarios donde creará una única instancia (como en Apache ODE).

2. Existen escenarios donde el orden con el que se ejecutan las invocaciones de ciertas operaciones no se corresponde con el orden en el que el servicio espera dichas invocaciones. Es decir, un servicio que para llevar a cabo cierta tarea necesita recibir dos mensajes (por ejemplo o_1 y o_2) de manera consecutiva, pero las invocaciones llegan en el orden inverso (es decir, primero o_2 y luego o_1). Cuando esto ocurre, no sabemos si el primer mensaje fue recibido ya que la instancia aún no fue creada (la actividad encargada de crear una instancia es la que está asociada a la operación o_1).

   El resultado es que las tres implementaciones tienen el mismo comportamiento. El mensaje se mantiene hasta tanto se cree la instancia que pueda recibirlo.

3. La especificación de WS-BPEL permite el uso de múltiples actividades de inicio ([5], Sección 10.4) que, sin un diseño adecuado del servicio puedan dar origen a excepciones del tipo conflicting receive. El ejemplo en cuestión tiene dos actividades de *receive* correlacionadas llamadas o_1 y o_2. Supongamos ahora que existen dos servicios distintos que realizan una invocación a cada operación (es decir un servicio invoca a la operación o_1 y el otro a la operación o_2). Una vez que se recibe la invocación a la operación o_1, el servicio creará una instancia que se quedará a la espera de la invocación de la actividad o_2. Ahora bien, cuando el segundo servicio invoque a la operación o_2, puede ocurrir que la instancia termine, o bien que cree una nueva instancia.

   a) Oracle BPEL Process Manager se encargará de arrojar una excepción del tipo conflicting receive.

   b) Apache ODE no permite múltiples actividades de inicio.

   c) ActiveBPEL le entregará el mensaje a la instancia activa.

	Oracle BPEL	ActiveBPEL 3	Apache ODE
Conjuntos de correlación (Ej. 1)	+	+	+
Conflictos en receive consecutivos (Ej. 1)	-	+	+/-
Mensajes persistidos (Ej. 2)	+	+	+
Multiples actividades de inicio (Ej. 3)	-	+	-

Cuadro 2.1: Comparación de implementaciones, fuente: [11]

Los ejemplos anteriores ilustran que las implementaciones existentes de WS-BPEL exhiben en muchos casos comportamientos diversos, en gran medida debido al hecho de que la especificación de WS-BPEL es informal. El cuadro 2.1 resume el comportamiento observado para cada implementación de WS-BPEL notando con "+" cuando la implementación se corresponde con la especificación y con "-" cuando la implementación no se corresponde.

La siguiente sección presentará distintas propuestas realizadas a fin de dar una definición formal de distintos aspectos de BPEL.

2.8. Semántica formal para BPEL

La especificación de BPEL está definida en lenguaje natural y como consecuencia existen inconsistencias, ambigüedades e incompletitud en varios puntos de esta (pueden consultarse en [18]). La forma para eliminar estos problemas es formalizando la descripción de BPEL a partir de modelos formales o semiformales.

2.8.1. Modelos de BPEL

Existen diversas propuestas que intentan dar una definición formal de BPEL o algunos aspectos relevantes del estándar. En esta sección comentaremos sobre varios modelos propuestos para definir la semántica de BPEL.

Las redes de Petri son un modelo formal para la concurrencia. Una red de Petri es un grafo bipartito dirigido en el que cada nodo es un lugar que puede contener *tokens* ó una transición. Existen muchas propuestas en la literatura (como por ejemplo [20], [21], [22]) que definen la semántica formal de BPEL asociando a cada proceso BPEL una red de Petri. En consecuencia, las técnicas de verificación desarrolladas para las redes de Petri, como por ejemplo determinar si hay deadlocks, pueden ser utilizadas para verificar propiedades de procesos BPEL. No existen formalizaciones de BPEL con redes de Petri que tengan en cuenta correlaciones.

También se han propuesto traducciones de BPEL en Promela, un pseudolenguaje de procesos (ver [23], [24], [25]). De esta manera, se utiliza SPIN para probar propiedades LTL sobre procesos BPEL.

Un cálculo de procesos (o algebra de procesos) es un lenguaje minimal que hace foco en la interacción, comunicación y sincronización entre procesos. Existen muchas propuestas en la literatura que utilizan cálculos de procesos para dar una definición formal de las primitivas de BPEL. En la próxima sección comentamos más detalladamente algunas de estas propuestas.

Las máquinas de estado abstracto (sus siglas en inglés ASMs) han sido usadas para modelar diferentes lenguajes. Una ASM básica se compone de un conjunto finito de reglas de transición. Cada regla de transición consiste de dos partes: una expresión booleana y una conjunto finito de asignaciones. ASMs también es usado para modelar procesos en BPEL. Farahbod, Glässer and Vajihollahi modelaron aspectos como actividades básicas y estructuradas, correlaciones, manejador de eventos y fallas utilizando ASMs (ver [26],[27],[28],[29]).

Un último modelo que comentaremos es un autómata finito determinístico utilizado para modelar las actividades BPEL (ver [30], [31],[32]). El estado del autómata es anotado con una expresión booleana. Esa expresión captura como el proceso BPEL interactúa con el entorno. El hecho que el autómata sea determinístico permite representar una semántica bien definida sin presentar ambigüedades.

2.8.2. Cálculos de procesos para BPEL

Los cálculos de procesos son formalismos propuestos para modelar sistemas concurrentes. Sus fundamentos matemáticos permiten establecer abstracciones entre los elementos reales de los sistemas y los componentes básicos del cálculo, lo que facilita la verificación de propiedades sobre los sistemas modelados.

Un cálculo de procesos se define dando la sintaxis y la semántica del lenguaje de manera formal. La sintaxis consiste en definir el conjunto de términos ó expresiones válidas del lenguaje, mientras que la semántica determina el significado de los programas. En general, la semántica se define de manera operacional a través de un conjunto de reglas que establecen los cambios de estados de un programa y su interacción con el ambiente.

Dado que en esta tesis estudiaremos los conjuntos de correlación utilizando cálculos de procesos, a continuación describiremos a grandes rasgos tres cálculos formales para la composición de servicios que incorporan mecanismos de correlación: SOCK [7], COWS [10] y BLITE[11]. Hacemos notar que si bien estos cálculos incorporan mecanismos de correlación, ninguno incluye excepciones que se originan a partir de las correlaciones, como define la especificación BPEL.

SOCK

SOCK es un cálculo de procesos donde el diseño de los servicios se descomponen en tres partes: *comportamiento*, *declaración*, y la *composición*. Cada una de estas partes puede ser diseñada independientemente la una de la otra.

La capa de comportamiento es un cálculo que describe el comportamiento de las primitivas que se utilizan para la comunicación de procesos y el orden en los que estas se ejecutan. Este orden está definido por los operadores de composición secuencial, el paralelo, el de elección bloqueante a la espera de un mensaje entrante. La segunda capa, declaración, considera el modo de la persistencia y los conjuntos de correlación. El modo de ejecución define la posibilidad de ejecutar las distintas instancias de un servicio en un orden secuencial o concurrente, el modo de persistencia permite declarar si una instancia[1] tiene un estado independiente o compartido con las demás instancias del servicio. Los conjuntos de correlación, serán quienes identifiquen a las instancias. Finalmente, la capa de composición será el cálculo que describa la interacción entre diferentes servicios.

SOCK considera interacciones de tipo unidireccional, como son las operaciones $one-way$ y $notification$, como así también las interacciones de tipo bidireccionales como $request-response$ y $solicit-response$.

En cuanto a los aspectos de correlación, las instancias están representadas explícitamente e identificadas unívocamente por los conjuntos de correlación. Esta identificación de instancias se produce en el momento que se recibe un mensaje sobre una operación de entrada.

[1] Las instancias en SOCK son llamadas sesiones.

COWS

Los elementos más importantes en la sintaxis de COWS son los socios y las operaciones que se realizan sobre estos. Un servicio puede ser identificado por diferentes nombres que le permitirán tomar distintos roles al momento de comunicarse con otros servicios (es decir, maneja conjuntos de correlación).

Los servicios son los encargados de crear instancias a partir de un mensaje de entrada. Sin embargo estos son capaces de recibir múltiples mensajes en cualquier orden, de forma tal que el primer mensaje recibido será el encargado de crear la instancia y los demás mensajes luego serán entregados. Cada instancia es identificada a partir de los valores de correlación.

El mecanismo para correlacionar mensajes es el de *Pattern-matching*. Cuando un servicio recibe un mensaje, realiza una operación de pattern-matching sobre todas las actividades de entrada que se encuentran listas. Formalmente esta operación retorna una sustitución (es decir, una función de variables en valores). Al momento de decidir qué actividad debe consumir el mensaje recibido, se elige no determinísticamente alguna de aquellas actividades para la cual la función de sustitución resultante del pattern-matching tiene dominio mínimo. De esta manera se elige siempre a la instancia que tiene más variables de correlación inicializadas.

Es importante destacar que en COWS se usa un mecanismo de prioridad donde si más de una instancia que activa una acción de receive puede tratar una invocación de algún socio, entonces la instancia que genera la menor cantidad de sustituciones será quién tome la solicitud. Para entender mejor este predicado supongamos que tenemos dos instancias activas que esperan la llegada del mensaje o_1 correlacionado por tres variables (por ejemplo x,y,z). Una instancia tendrá inicializada dos variables (por ejemplo x e y con los valores 1 y 2 respectivamente) y la otra instancia tendrá inicializada una sola variable (por ejemplo z con el valor 3). Cuando se realiza la invocación de $o_1(1, 2, 3)$ el predicado *match* con la primer instancia devolverá la sustitución de z por 3 mientras que con la segunda instancia realizará la sustitución de x por 1 e y por 2. En particular con la segunda instancia deberá realizar más sustituciones que la primer instancia, en consecuencia el mensaje será entregado a la primer instancia.

BLITE

BLITE es un cálculo de procesos propuesto como una versión de alto nivel de COWS, cuyas primitivas se acercan a las primitivas de BPEL. No obstante, hereda todas las decisiones de diseño hechas en COWS con respecto al manejo de correlaciones. En esencia, utiliza el mismo criterio de prioridad de actividades con mayor número de variables correlacionadas ya instanciada. Define una función auxiliar *match*, que devuelve un conjunto de sustituciones de variables correlacionadas logrando que dos tuplas $\vec{v}$ y $\vec{w}$ unifiquen. Para las reglas de comunicación y de creación de instancias existe un predicado auxiliar que se encarga de elegir la primi-

tiva *receive* de la instancia que tenga que inicializar la menor cantidad de variables de correlación.

Diferencia entre cálculos

Como pudimos ver, tanto SOCK, COWS y BLITE se encargan de formalizar los mecanismos de correlaciones. Sin embargo existen diferencias entre estos modelos. La diferencia más importante entre SOCK y COWS es que las instancias de un servicio se representan explícitamente en SOCK mientras que en COWS no. Es decir, en SOCK, un proceso P recién instanciado se asocia explícitamente con su estado S usando la notación [P, S], mientras que en COWS un proceso P recién instanciado simplemente se añade en paralelo con los otros procesos. Otra diferencia es que en SOCK la información de correlación puede cambiar durante la ejecución del proceso asignándole nuevos valores a las variables de correlación. Esto es posible dado que un proceso puede cambiar de socio de manera dinámica modificando la información correspondiente de las correlaciones.

Como dijimos anteriormente, BLITE es una variante de alto nivel respecto de COWS que hereda el mismo manejo de COWS para los conjuntos de correlación.

Capítulo 3

Evaluando Correlaciones en ActiveBPEL

Como ilustra el capítulo precedente, tanto las implementaciones de BPEL como sus modelos formales presentan diferencias notorias. A fin de contar con una implementación de referencia que nos permita definir la semántica de las excepciones de correlación, nuestro primer trabajo consiste en desarrollar un conjunto de servicios que nos permita inferir el comportamiento de las correlaciones y su adherencia a modelos formales propuestos anteriormente (presentados en la Sección 2.8.2).

En particular elegimos ActiveBPEL por ser open source y porque luego de una evaluación preliminar exhibió menos comportamiento que se aparta de la especificación.

A continuación diseñaremos e implementaremos programas con el fin de poder inferir el comportamiento de los mecanismos de correlación que son implementados en ActiveBPEL. Los programas a construir nos permitirán describir escenarios donde podremos concluir el manejo que existe sobre los conjuntos de correlación. Dado que nos interesa evaluar el comportamiento de los conjuntos de correlación nos enfocaremos en estudiar la entrega de mensajes que están correlacionados. En particular hay dos tipos de mensajes que se pueden enviar a un servicio para que este luego los entregue a la instancia correspondiente:

1. Mensajes asociados a actividades que *crean instancias* (las actividades iniciales de la definición). Cuando un servicio recibe un mensaje asociado a una actividad que crea una instancia, este mensaje es despachado creando una instancia. En la definición del servicio, un mensaje asociado a la creación de instancias puede aparecer:

   a) 1 vez, es decir como actividad inicial.

   b) Más de 1 vez. Un mensaje que aparece más de una vez en la definición de un servicio puede darse en un orden:

      1) Secuencial.

2) Paralelo.

2. Mensajes asociados a actividades que *no crean instancias*. Cuando se intenta entregar un mensaje que no crea instancia puede ocurrir que:

   a) La instancia haya sido creada.

   b) La instancia aún no haya sido creada.

Por lo tanto buscaremos ejemplos que nos muestren qué ocurre cuando se envían estos dos tipos de mensajes, explorando cada rama posible de nuestro análisis. Además, estudiaremos si el comportamiento observado es descripto por los cálculos SOCK, COWS y BLITE, tildando con una cruz en caso que la semántica describa el comportamiento observado.

Por último, para completar nuestro estudio, realizaremos ejemplos que permitan mostrar las excepciones relacionadas a los conjuntos de correlación. Los códigos fuente de los ejemplos listados a continuación se encuentran disponibles en `https://sites.google.com/site/correxcepactivebpel/download`.

Ejemplo 1:

- **Objetivos:** Observar si pueden existir instancias con el mismo valor de correlación.
- **Construcción:** Construyamos un servicio con dos acciones de receive(por ejemplo o_1 y o_2) asociadas a una única variable de correlación(x). Por cada invocación a la operación o_1 el servicio creará una nueva instancia. Dado que cada operación está correlacionada por la variable x, deberemos llamar a la operación inicial con un valor de correlación. Sin embargo, ¿Qué ocurre si llamamos a la operación o_1 con el valor de correlación de alguna instancia activa?
- **Escenario de uso:**
  1. invocar $o_1(1)$
  2. invocar $o_1(1)$
- **Resultado:** A partir de este ejemplo pudimos observar que podemos tener dos instancias con la variable de correlación inicializada con el mismo valor. Como consecuencia no podremos garantizar que no se mezclen los mensajes entre distintas instancias.
- **Relación con especificación:** La especificación no es clara acerca de la unicidad de conjuntos de correlación.
- **Concuerda con semántica:**

  ☐ **SOCK** ☒ **COWS** ☒ **BLITE**

Ejemplo 2:

- **Objetivos:** Observar creación de instancias cuando la actividad inicial también aparece como otra actividad.
- **Construcción:** Construyamos un servicio con dos acciones de receive para la misma operación (por ejemplo, o_1), y variable de correlación (x). Cuando se invoca a la primer operación o_1 se crea una nueva instancia que debería terminar luego de que se invoque a la segunda operación de *receive*. Al invocar por segunda vez a la operación o_1 con el valor de correlación de la instancia que ya fue creada ¿se creará una nueva instancia o la instancia creada previamente se completará?
- **Escenario de uso:**
  1. invocar $o_1(1)$
  2. invocar $o_1(1)$
- **Resultado:** El resultado depende de una variable externa (el tiempo que transcurre entre invocaciones). Puede o bien crear una nueva instancia ó terminar la instancia creada.
- **Relación con especificación:** La especificación no parece prohibir explícitamente este comportamiento.
- **Concuerda con semántica:**

  ☐ **SOCK** ☐ **COWS** ☐ **BLITE**

Ejemplo 3:

- **Objetivos:** Observar creación y completitud de instancias con múltiples actividades de inicio.
- **Construcción:** Construyamos un servicio con dos acciones de receive (por ejemplo, o_1 y o_2) en paralelo asociadas a una única variable de correlación (x). Cuando se invoca a este servicio a través de la acción o_1 se creará una instancia que para terminar deberá esperar la invocación de la operación o_2. Si llama a la operación o_2 con el mismo valor de correlación de la instancia creada ¿finaliza la ejecución de la instancia ya creada o se crea una nueva?
- **Escenario de uso:**
  1. invocar $o_1(1)$
  2. invocar $o_2(1)$
- **Resultado:** El resultado depende de una variable externa (el tiempo que transcurre entre invocaciones). Puede o bien crear una nueva instancia ó terminar la instancia creada.
- **Relación con especificación:** Igual que el ejemplo 2.
- **Concuerda con semántica:**

  ☐ **SOCK** ☐ **COWS** ☐ **BLITE**

Ejemplo 4:

- **Objetivos:** Observar el manejo de excepciones Ambiguous Receive.
- **Construcción:** Construyamos un servicio con una actividad inicial *receive*, cuyas variables de correlación son x e y. Luego de ejecutarse esta acción, se activan dos acciones de *receive* concurrentes, cuyas variables de correlación son x para una actividad e y para la otra. ¿Qué ocurre si el valor para x e y es el mismo?
- **Escenario de uso:**
  1. invocar $o_1(1)$
  2. invocar $o_2(1)$
- **Resultado:** Cuando el ambas variables de correlación tienen el mismo valor de correlación se arroja la excepción Ambiguous Receive.
- **Relación con especificación:** El comportamiento observado se corresponde con la especificación donde dice: "Sí una instancia de proceso de negocio activa simultáneamente dos o mas IMAs [inbound message activities] para el mismo socio, operación pero diferente conjunto de correlación, y la correlación de múltiples de estas actividades igualan a un mensaje de entrada, entonces la excepción bpel:ambiguousReceive DEBE ser arrojada por todos los IMAs cuyos conjuntos de correlaciones igualen al mensaje entrante".
- **Concuerda con semántica:**

  ☐ **SOCK** ☐ **COWS** ☐ **BLITE**

Ejemplo 5:

- **Objetivos:** Observar el manejo de excepciones Conflicting Receive.
- **Construcción:** Construyamos un servicio cuya actividad inicial es una acción de *receive* con una única variables de correlación x. Luego de ejecutarse activa dos acciones de entrada en paralelo con la correlación x.
- **Escenario de uso:**
  1. invocar $o_1(1)$
- **Resultado:** Luego de la primera invocación el servicio alcanza la excepción Conflicting Receive.
- **Relación con especificación:** El comportamiento observado se corresponde con la especificación donde dice: "Sí dos o más acciones de *receive* para el mismo socio, nombre de operación y conjunto de correlación son activadas simultáneamente durante la ejecución, entonces la excepción bpel:conflictingReceive DEBE ser arrojada."
- **Concuerda con semántica:**

  ☐ **SOCK** ☐ **COWS** ☐ **BLITE**

Ejemplo 6:

- **Objetivos:** Observar invocación de mensajes cuando la instancia aún no fue creada.
- **Construcción:** Construyamos un servicio con dos actividades secuenciales llamadas o_1 y o_2 con una única variable de correlación asociada. La primer actividad será la encargada de crear una instancia pero ¿Qué ocurre sí primero se la invoca con la operación o_2? ¿Se pierde ese mensaje si no hay una instancia creada?
- **Escenario de uso:**
  1. invocar $o_2(1)$
  2. invocar $o_1(1)$
- **Resultado:** Al momento de crearse una instancia, se le entregan a esta aquellos mensajes que fueron recibidos y no pudieron ser entregados. En caso que se haya invocado al servicio con la operación o_2 y luego o_1 y el mismo valor de correlación, entonces luego de crearse la instancia, se le entrega el primer mensaje y esta finaliza.
- **Relación con especificación:** La especificación no dice nada al respecto.
- **Concuerda con semántica:**
  ☐ **SOCK** ☐ **COWS** ☐ **BLITE**

Capítulo 4

CAB

En este capítulo presentaremos un cálculo de procesos llamado Correlaciones en ActiveBPEL (Cab). La semántica formal considera mecanismos básicos para la comunicación y composición de servicios haciendo foco en los conjuntos de correlación y las excepciones asociadas a estos. De esta manera, Cab será un versión simplificada de la especificación de WS-BPEL, diseñado a partir de las características que nos propusimos estudiar en este trabajo (conjuntos de correlación, excepciones asociadas a las correlaciones). Una vez presentada la gramática y la semántica, se procederá a mostrar cómo modelar escenarios concretos a partir del cálculo desarrollado.

4.1. Sintaxis

Sean O, S, V y A conjuntos infinitos enumerables tales que:

- O representa los nombres de operaciones, o el nombre de una operación y $o \in O$.
- S representa los nombres de servicios, s el nombre de un servicio y $s \in S$.
- V representa variables de datos y $x, y, \ldots \in V$.
- A representa constantes, donde $a, a', \ldots, b, b', \ldots \in A$.

Asumiremos que v puede ser una variable de datos o una constante, es decir $v \in A \cup V$. Usaremos $\vec{v}$ como lista de variables, es decir una notación compacta de $< v_1, v_2, \ldots, v_n >$ con $n \geq 0$.

Definición 4.1.1. *Un conjunto de correlación C es un conjunto finito de variables de datos, $C \subset V$, y una instancia de correlación es una función parcial $c : V \mapsto A \cup \{\perp\}$. Para cualquier conjunto de correlación C, denotaremos con $c_\perp$ instancias de correlación sin inicializar, esto es 1) $dom(c_\perp) = C$ y 2) $c_\perp(x) = \perp \; \forall \; x \in C$.*

Notación 4.1.1. *Escribiremos instancias de correlación como un conjunto de pares, como por ejemplo:* $c = \{x \mapsto \perp, y \mapsto b\}$ *donde* $dom(c_\perp) = \{x, y\}$.

Definición 4.1.2. *Diremos que dos instancias de correlación* c_1 *y* c_2 *no* colisionan *si y sólo sí* $\forall x \in dom(c_1) \cap dom(c_2).(c_1(x) \neq \perp \wedge c_2(x) \neq \perp \Rightarrow c_1(x) \neq c_2(x))$.

Esta definición nos permite asegurar que toda variable compartida por instancias de correlación distinta, tienen valores distintos. Supongamos por ejemplo que tenemos dos instancias de correlación c_1 y c_2 y la variable de correlación x que pertenece al dominio de cada instancia, es decir al conjunto finito de variables de datos de la instancia de c_1 y c_2. Si x está inicializada en cada instancia con el mismo valor entonces diremos que esas dos instancias colisionan. En cambio si en una instancia x está inicializada con el valor 1 y en la otra instancia está inicializada con el valor 2, diremos que esas dos instancias no colisionan. Si x no estuviese inicializada en cualquiera de las dos instancias nos alcanzaría para decir que tampoco colisionan.

Definición 4.1.3. *Los términos de* CAB *estarán dados por la siguiente gramática:*

$$
\begin{array}{lll}
(\text{PROC}) & P ::= 0 \mid \sum_i o_i(\vec{x_i}); P_i \mid \overline{o}\langle\vec{v}\rangle \mid P|P \mid P;P \mid \textbf{if } v = v' \textbf{ then } P \textbf{ else } P \mid \textbf{rec}_X\ P \mid X \\
(\text{INST}) & I ::= 0 \mid c \triangleright [P] \mid I|I \\
(\text{MSJ}) & M ::= \emptyset \mid \overline{o}\langle\vec{a}\rangle|M \\
(\text{SIST}) & N ::= 0 \mid s_C^O\{P, I, M\} \mid N \| N
\end{array}
$$

Estos términos se encuentran divididos en cuatro categorías sintácticas. El primer grupo, (PROC), describe el flujo de actividades que compone a un proceso, el segundo, es decir (INST), describe conjuntos de instancias creadas por un servicio. El tercer grupo representa el conjunto de mensajes recibidos por un servicio que deben ser entregados a las instancias correspondientes. Finalmente (SIST) define a un sistema ya sea como un único servicio o como la composición de varios servicios

Considere la siguiente descripción informal para cada término de la gramática:

Para (PROC)

- 0 representa un proceso que ha finalizado su ejecución.
- $\sum_i o_i(\vec{x_i}); P_i$ espera la invocación de alguna de las acciones de entrada $o_i(\vec{x_i})$ para luego continuar con el flujo P_i. Se corresponde con la actividad *pick*.
- $\overline{o}\langle\vec{v}\rangle; P$ denota la invocación de una operación o seguido del flujo de actividades descriptos por P. Se corresponde con la actividad *invoke*.
- $P|Q$ representa la composición en paralelo de los procesos P y Q. Se corresponde con la actividad $flow$.
- $P; Q$ denota la composición secuencial de los procesos P y Q. Se corresponde con la actividad *sequence*.

- **if** $v = v'$ **then** P **else** Q describe la elección condicional en el flujo de un proceso. Se corresponde con la actividad if.
- $\mathbf{rec}_X\ P$ describe la recursión utilizando al proceso P y a la variable de procesos X. En cada paso X es reemplazada por $\mathbf{rec}_X\ P$.
- X representa una variable de procesos.

Para (INST)

- 0 representa el conjunto vacío de instancias.
- $c \triangleright [P]$ denota una instancia de servicio cuyas variables de correlación se encuentran inicializadas según la descripción dada por la instancia de correlación c, y el estado de ejecución es el descripto por P.
- $I_1|I_2$ denota la unión de instancias.

Para (MSJ)

- $\emptyset$ es el conjunto vacío de mensajes.
- $\overline{o}\langle\vec{v}\rangle|M$ denota al conjunto de mensajes que se obtiene agregando $\overline{o}\langle\vec{v}\rangle$ al conjunto M.

Para (SIST)

- 0 representa el conjunto vacío de servicios.
- $s^O_C\{P, I, M\}$ donde s es el nombre del servicio, C el conjunto de variables de correlación, O el conjunto de operaciones que ofrece el servicio, P la definición del servicio, I el conjunto de instancias activas, y M el conjunto de mensajes recibidos que aún no fueron entregados a las instancias.
- $N_1||N_2$ es la composición paralela de servicios.

Veamos algunos ejemplos representativos para los términos introducidos. Supongamos que tenemos un servicio que cuenta con las operaciones o_1 y o_2 y queremos describir el flujo de actividades (PROC):

- si el servicio espera recibir la operación o_1 que recibe el parámetro x, escribiremos $o_1(x)$.
- si el servicio invoca a la operación o_2 con el conjunto de valor $\vec{v}$, escribiremos $\overline{o_2}\langle\vec{v}\rangle$.
- si se encuentran las operaciones en un orden secuencial, escribiremos: $o_1;o_2$.

- si se encuentran las operaciones en paralelo, escribiremos: $o_1 \mid o_2$.

Supongamos ahora que queremos describir al servicio que tiene dos instancias, en una instancia la única variable de correlación x está inicializada y en la otra instancia aún no. Lo escribiremos como: $\{x \mapsto a\} \triangleright [P] | \{x \mapsto \perp\} \triangleright [P]$. Finalmente cuando describamos un servicio a la espera de despachar a una instancia el mensaje $\overline{o_1}\langle b\rangle$ lo escribiremos como: $s_x^{o_1}\{o_1(x); \overline{o_2}\langle\vec{v}\rangle, \{x \mapsto a\} \triangleright [o_1(x); \overline{o_2}\langle\vec{v}\rangle], \overline{o_1}\langle b\rangle\}$.

Definición 4.1.4. *La regla de precedencia establece que la composición secuencial tiene mayor precedencia que la composición en paralelo. Por ejemplo:* $o_1(x); o_2(x) | o_3(x)$ *es equivalente a escribir* $(o_1(x); o_2(x)) | o_3(x)$.

4.2. Términos bien formados

A continuación se introduce algunas nociones auxiliares que serán utilizadas en la definición de *Términos bien formados.*

Definición 4.2.1. *Un proceso* P *tiene un conjunto de nombres de operaciones de entrada y otro de salida, denotados por* $in(P)$ *y* $out(P)$ *respectivamente.*

La definición de *in* se define inductivamente como:

$\mathbf{in}(0) = \emptyset$	$\mathbf{in}(\overline{o}\langle\vec{v}\rangle) = \emptyset$
$\mathbf{in}(\sum_i o_i(\vec{x_i}); P_i) = \bigcup_i \{o_i\} \cup \mathbf{in}(P_i)$	$\mathbf{in}(P_1 \mid P_2) = \mathbf{in}(P_1) \cup \mathbf{in}(P_2)$
$\mathbf{in}(P_1; P_2) = \mathbf{in}(P_1)$	$\mathbf{in}(if\ v = v'\ then\ P_1\ else\ P_2) = \mathbf{in}(P_1) \cup \mathbf{in}(P_2)$
$\mathbf{in}(\mathbf{rec}_X\ P) = \mathbf{in}(P)$	$\mathbf{in}(\mathrm{X}) = \emptyset$

La definición de *out* se define inductivamente como:

$\mathbf{out}(0) = \emptyset$	$\mathbf{out}(\overline{o}\langle\vec{v}\rangle) = \{o\}$
$\mathbf{out}(\sum_i o_i(\vec{x_i}); P_i) = \mathbf{out}(P_i)$	$\mathbf{out}(P_1 \mid P_2) = \mathbf{out}(P_1) \cup \mathbf{out}(P_2)$
$\mathbf{out}(P_1; P_2) = \mathbf{out}(P_1)$	$\mathbf{out}(if\ v = v'\ then\ P_1\ else\ P_2) = \mathbf{out}(P_1) \cup \mathbf{out}(P_2)$
$\mathbf{out}(\mathbf{rec}_X\ P) = \mathbf{out}(P)$	$\mathbf{out}(\mathrm{X}) = \emptyset$

Consideremos los siguientes ejemplos donde P se escribe como:

- $o_1(x); o_2(x)$, las operaciones de *in* serán denotadas por el conjunto $\{o_1, o_2\}$ y las operaciones de *out* por el conjunto vacío($\emptyset$).

- $o_1(x); \overline{o_2}\langle a\rangle | o_3(x)$, las operaciones de *in* serán denotadas por el conjunto $\{o_1, o_3\}$ y las operaciones de *out* por el conjunto $\{o_2\}$.

Definición 4.2.2. *Un sistema* N *tiene un conjunto de nombres de operaciones de entrada y otro de salida, denotados por* $in(N)$ *y* $out(N)$ *respectivamente.*

La definición de *in* se define inductivamente como:

$$\mathbf{in}(0) = \emptyset \quad \mathbf{in}(s_C^O\{P, I, M\}) = \mathbf{in}(P) \quad \mathbf{in}(N_1|N_2) = \mathbf{in}(N_1) \cup \mathbf{in}(N_2)$$

La definición de *out* se define inductivamente como:

$$\mathbf{out}(0) = \emptyset \quad \mathbf{out}(s_C^O\{P, I, M\}) = \mathbf{out}(P) \quad \mathbf{out}(N_1|N_2) = \mathbf{out}(N_1) \cup \mathbf{out}(N_2)$$

Definición 4.2.3. *Denotaremos con subj(M) al conjunto de nombre de operaciones que aparecen en el conjunto de mensajes de M.*

La definición de *subj* se define inductivamente como:

$$\mathbf{subj}(\emptyset) = \emptyset \quad \mathbf{subj}(\overline{o}\langle\vec{v}\rangle|M) = \{o\} \cup \mathbf{subj}(M)$$

Definición 4.2.4. *Un proceso P es bloqueado en receive si y sólo si P cumple con las siguientes formas:*

$P = \sum_i o_i(\vec{x_i}); P_i, o$
$P = P_1|P_2$ *donde* P_1 *y* P_2 *bloqueados en receive* $, o$
$P = P_1; P_2$ *donde* P_1 *bloqueado en receive*

Esta definición nos permite decir cuando el flujo de actividades comienza con actividades de *receive*.

Definición 4.2.5. *Un conjunto de instancias I de un servicio está correlacionado por un conjunto de correlación C, denotado por $C \blacktriangleright I$, si y sólo si*

$$C \blacktriangleright 0 \qquad \frac{dom(c) = C}{C \blacktriangleright c \triangleright [P]} \qquad \frac{C \blacktriangleright I_1 \quad C \blacktriangleright I_2}{C \blacktriangleright I_1|I_2}$$

Esta definición permite describir los conjuntos de correlación que tiene asociada cada instancia. Una instancia activa ($c \triangleright [P]$) correlacionada por la función c requiere que el dominio sobre el que está definida la función sea el conjunto de correlación C. La unión de instancias, estarán correlacionadas por el mismo conjunto de correlación. Finalmente, para el conjunto de instancias vacío no hay ninguna restricción que mencionar.

Definición 4.2.6. *Un proceso P es bien formado si: $in(P) \cap out(P) = \emptyset$.*

Es decir, diremos que un proceso P es bien formado cuando este no se comunique consigo mismo. El flujo de actividades $o_1(x)|o_2(x); \overline{o_3}\langle a\rangle$ denota a un proceso bien formado. Un ejemplo de procesos que no son bien formados considere $o_1(x)|\overline{o_1}\langle a\rangle$, el problema aquí es que el proceso mediante la operación o_1 podría hablar y escucharse a sí mismo (sincronización interna) ya que la operación de entrada y de salida resulta ser la misma.

Definición 4.2.7. *Una instancia $c \triangleright [P]$ es bien formada si y solo si P es bien formado.*

Definición 4.2.8. *Un sistema $s_C^O\{P, I, M\}$ es bien formado si y solo se cumplen las siguientes condiciones:*

1. *La definición del servicio P es un proceso bien formado y bloqueado en receive.*
2. *Todas las instancias en I están bien formadas.*
3. *Las instancias I son correlacionadas por C, es decir, $C \blacktriangleright I$;*
4. *Las operaciones de entrada que aparecen en las instancias de los servicios y en la definición del servicio P son declaradas en las operaciones que provee el servicio, es decir, $in(I) \cup in(P) \subseteq O$. Notar que una instancia describe una ejecución parcial del flujo de actividades P, sin embargo esta restricción alcanza para asegurarnos que existe una correspondencia entre las actividades de entrada tanto de las instancias, como la de los procesos con las operaciones que ofrece el servicio.*
5. *Los nombres de operaciones de entrada que aparecen en el conjunto de mensajes pendientes son declarados en el conjunto de operaciones que provee el servicio, es decir, $subj(M) \subseteq O$.*

Un sistema es bien formado si y sólo si no hay conflictos en cuanto a las operaciones de entrada entre los diferentes servicios, es decir, para toda composición de servicios $N \| N'$ se cumple que $in(N) \cap in(N') = \emptyset$.

La condición *bien formado* permite garantizar ciertas propiedades implícitas de los lenguajes de orquestación de servicios. La condición 1 nos permite asegurar que los servicios están definidos como procesos bloqueados en receive donde la actividad de inicio será una primitiva de *receive* encargada de crear una instancia. Las condiciones 2, 3 y 4 definen restricciones sobre la ejecución parcial del flujo de actividades. La condición 5 asegura que los mensajes guardados puedan ser entregados a las instancias. Ejemplos de sistemas bien formados:

- $s_{\{x\}}^{\{o_1,o_2\}}\{o_1(x)|o_2(y); \overline{o_3}\langle v\rangle, 0, \emptyset\}$
- $s_{\{x\}}^{\{o_1,o_2\}}\{o_1(x); o_2(y), 0, \overline{o_2}\langle b\rangle\}$
- $s_{\{x\}}^{\{o_1,o_2\}}\{o_1(x); o_2(y), \{x \mapsto a\} \triangleright [o_2(y)], \overline{o_2}\langle b\rangle\}$

Ejemplo de sistemas que no son bien formados:

- $s_{\{x\}}^{\{o_2,o_3\}}\{\overline{o_1}\langle a\rangle; o_2(x); o_3(y), 0, \emptyset\}$, el flujo de actividades P no es bien formado ya que no es un proceso bloqueado en receive. Viola condición 1.

- $s^{\{o_1\}}_{\{x\}}\{P, \{x \mapsto a\} \triangleright [o_1(x)|\overline{o_1}\langle a\rangle], \emptyset\}$, la instancia no es bien formada ya que el flujo de actividades permite comunicarse consigo mismo. Viola condición 2.
- $s^{\{o_1,o_2\}}_{\{x\}}\{o_1(x); o_2(y), \{y \mapsto a\} \triangleright [o_2(y)], \overline{o_2}\langle b\rangle\}$, la variable de correlación y no pertenece al conjunto de correlación C. Viola condición 3.
- $s^{\{o_1\}}_{\{x\}}\{o_1(x); o_2(y), \{x \mapsto a\} \triangleright [o_2(y)], \overline{o_2}\langle b\rangle\}$, el proceso P y la instancia I tienen definida una operación de entrada o_2 que el servicio no considera en su definición de operaciones de entrada O. Viola condición 4.
- $s^{\{o_1,o_2\}}_{\{x\}}\{o_1(x); o_2(y), 0, \overline{o_3}\langle b\rangle\}$, hay un mensaje guardado que no se corresponde al flujo de actividades de la definición del servicio. Viola condición 5.

4.3. Semántica Operacional

A continuación daremos algunas nociones útiles para definir la semántica operacional.

4.3.1. Variables libres y ligadas

En los procesos $\sum_i o_i(\vec{x_i}); P_i$ la ocurrencia de las variables $\vec{x_i}$ están ligadas con alcance P_i.

Notación 4.3.1. *Escribiremos fn(P) para los nombres de las variables libres en P.*

Notación 4.3.2. *Escribiremos bn(P) para los nombres de las variables ligadas en P.*

Veamos los siguientes ejemplos:

- $\mathrm{fn}(\overline{o_1}\langle x\rangle|o_2(z); 0) = \{x\}$
- $\mathrm{bn}(\overline{o_1}\langle x\rangle|o_2(z); 0) = \{z\}$
- $\mathrm{bn}(o_1(x, y); 0|o_2(z); 0) = \{x, y, z\}$

A continuación daremos la definición inductiva de variables libres y ligadas para instancias y sistemas (notar que en el conjunto de mensajes M no aparecen variables, por lo tanto son vacíos):

$fn(0) = \emptyset$	$bn(0) = \emptyset$
$fn(\{c\} \triangleright [P]) = fn(P) \setminus dom(c)$	$bn(\{c\} \triangleright [P]) = bn(P) \cup dom(c)$
$fn(I_1\|I_2) = fn(I_1) \cup fn(I_2)$	$bn(I_1\|I_2) = bn(I_1) \cup bn(I_2)$
$fn(s^O_C\{P, I, M\}) = fn(P) \cup fn(I) \setminus C$	$bn(s^O_C\{P, I, M\}) = bn(P) \cup bn(I)$
$fn(N_1\|\|N_2) = fn(N_1) \cup fn(N_2)$	$bn(N_1\|\|N_2) = bn(N_1) \cup bn(N_2)$

Remarcaremos que cualquier nombre $x \in C$ actúa como un ligador en $s^O_C\{P, I, M\}$. Por ejemplo, todas las ocurrencias de x en $N = s^O_{\{x\}}\{o(x,y).P, \{x \mapsto a\} \triangleright [Q], M\}$ están ligadas a la variable de correlación x. Notar que x in $o(x,y).P$ es también ligada al nombre de la correlación y no puede ser α-renombrada sin renombrar la variable de correlación, es decir $N \equiv_\alpha s^O_{\{z\}}\{o(z,y).P[z/x], \{z \mapsto a\} \triangleright [Q[z/x]], M\}$ para cualquier variable fresca z. Contrariamente, $N \not\equiv_\alpha s^O_{\{x\}}\{o(z,y).P[z/x], \{x \mapsto a\} \triangleright [Q], M\}$. Por otra parte, sólo vamos a considerar sistemas para los cuales las únicas variables compartidas entre prefijos de entrada son las variables de correlación.

4.3.2. Sustitución

Consideraremos $P\{x_0/v_0, ..., x_n/v_n\}$ como la sustitución simultánea de las ocurrencias libres de x_i por v_i.

También usaremos instancias de correlación como sustituciones. Cuando un conjunto de correlación es aplicado sobre un término, se sustituye la variable sólo si la imagen es valor distinto de $\perp$. Para instancias donde $c = \{x \mapsto \perp, y \mapsto b\}$, $(\overline{o}\langle x, y\rangle; P)c = (\overline{o}\langle x, y\rangle; P)[b/y] = \overline{o}\langle x, b\rangle; P[b/y]$.

4.3.3. Operador de actualización

Sean c_1 y c_2 instancias de correlación, definiremos el operador de actualización $\_[\_]$ tal que $dom(c_1[c_2]) = dom(c_1)$ y

$$c_1[c_2](x) = \begin{cases} c_1(x) & \text{if } x \notin dom(c_2) \vee (x \in dom(c_2) \wedge (c_1(x) = c_2(x) \vee c_2(x) = \perp)) \\ c_2(x) & \text{if } c_1(x) = \perp \wedge x \in dom(c_2) \\ undefined & \text{en los casos contrarios} \end{cases}$$

Dadas dos instancias de correlación c_1 y c_2, el operador de actualización $c_1[c_2](x)$ permitirá actualizar variables de correlación que no fueron inicializadas en c_1. Supongamos por ejemplo que tenemos: $c_1 = \{x \mapsto \perp, y \mapsto b\}$ y $c_2 = \{x \mapsto a\}$, el operador de actualización actualizará el valor de x con el valor a. Si tuviéramos el caso en que x ya fue inicializada en c_1 y x pertenece al dominio de c_2, entonces x debería no estar definida en c_2 o bien, estar inicializada con el mismo valor que en c_1. Por ejemplo: $c_1 = \{x \mapsto a, y \mapsto b\}$ y $c_2 = \{x \mapsto a\}$ ó $c_1 = \{x \mapsto a, y \mapsto b\}$ y $c_2 = \{x \mapsto \perp\}$. Lo que no debería ocurrir es que x esté inicializada con valores distintos, es decir: $c_1 = \{x \mapsto a, y \mapsto b\}$ y $c_2 = \{x \mapsto c\}$.

4.3.4. Sistema de transiciones etiquetadas

La semántica operacional de CAB estará definida para términos bien formados a partir de un sistema de transiciones etiquetadas y de la equivalencia estructural de procesos. Dado que nuestro objetivo es dar una semántica para CAB que describa el comportamiento de servicios que alcanzan excepciones debido al mecanismo de

correlación, extenderemos la sintaxis de los procesos para representar excepciones alcanzadas durante la ejecución:

$$(\text{PROC}) \quad P ::= \ldots \mid \dagger \mid \ddagger$$

† describe la excepción *ambiguous receive* y ‡ describe la excepción *conflicting receive*.

Definición 4.3.1. *La equivalencia estructural, escrita* $\equiv$*, es la relación de equivalencia que satisface los axiomas de la asociatividad y conmutatividad para* $|$*,* $+$*,* $\|$*. El* 0 *resulta ser la identidad para* $|$ *(sobre* (PROC) *e* (INST)*) y para* $||$*. El operador ; cumple el axioma de asociatividad y además se cumple que:*

$$0; P \equiv P \qquad \dagger; P \equiv \dagger \qquad \dagger \mid Q \equiv \dagger \qquad \ddagger; P \equiv \ddagger \qquad \ddagger \mid Q \equiv \ddagger$$

Donde Q *no contiene a* † *ni a* ‡*.*

La composición en paralelo de mensajes también es asociativa, conmutativa y el $\emptyset$ *resulta ser su identidad.*

El sistema de transiciones etiquetadas considera las siguientes acciones:

$$\alpha ::= x(v) \mid \overline{x}\langle v \rangle$$
$$\lambda ::= \alpha \mid \tau$$

Como descripción informal de cada acción:

- $o(v)$ estará asociada a acciones de input (*receive*).
- $\overline{o}\langle v \rangle$ estará asociado a acciones de output (*invoke*).
- τ acción no observable.

Las transiciones para los sistemas y las instancias son etiquetadas por la acción α, mientras que las transiciones para los procesos son etiquetadas por los pares α, c, donde α es una acción y c es una instancia de correlación. En este caso, una transición $P \xrightarrow{\alpha,c} P'$ denota el hecho que una proceso P reduce a P' a partir de α y actualizando la variable de correlación a partir de c. La semántica para los procesos está definida usando una relación de transición auxiliar $P \overset{\alpha,c}{\longmapsto} P'$. El significado de las etiquetas es igual a $\xrightarrow{\alpha,c}$. La principal diferencia entre $\overset{\alpha,c}{\longmapsto}$ y $\xrightarrow{\alpha,c}$ es que la primera no describe la ocurrencia de excepciones mientras que la segunda sí.

El sistema de transiciones etiquetadas para los procesos, instancias y servicios de CAB, es definido por las reglas descriptas en la Figura 4.1. A continuación comentaremos informalmente las reglas del sistema de transición.

PROCESOS

(IN)
$$\sum_i o_i(\vec{x}_i); P_i \xrightarrow{o_i(\vec{v}),\vec{x}_i \mapsto \vec{v}} P_i\{\vec{x}_i/\vec{v}\}$$

(OUT)
$$\overline{o_i}\langle\vec{a}\rangle \xrightarrow{\overline{o_i}\langle\vec{a}\rangle,\emptyset} 0$$

(PAR)
$$\frac{P_1 \xrightarrow{\alpha,c} P_1'}{P_1|P_2 \xrightarrow{\alpha,c} P_1'|P_2}$$

(SEQ)
$$\frac{P_1 \xrightarrow{\alpha,c} P_1'}{P_1; P_2 \xrightarrow{\alpha,c} P_1'; (P_2 c)}$$

(REC)
$$\frac{P[\mathbf{rec}_X\ P/_X] \xrightarrow{\alpha,c} P'}{\mathbf{rec}_X\ P \xrightarrow{\alpha,c} P'}$$

(THEN)
$$\mathbf{if}\ a = a\ \mathbf{then}\ P_1\ \mathbf{else}\ P_2 \xrightarrow{\tau,\emptyset} P_1$$

(ELSE)
$$\frac{a \neq b}{\mathbf{if}\ a = b\ \mathbf{then}\ P_1\ \mathbf{else}\ P_2 \xrightarrow{\tau,\emptyset} P_2}$$

(NO-EXCP)
$$\frac{P \xrightarrow{\alpha,c} P' \quad P \not\xrightarrow{\alpha,c_1} \dagger \quad P \not\xrightarrow{\alpha,c_1} \ddagger}{P \xrightarrow{\alpha,c} P'}$$

(AMB-REC-EXCP)
$$\frac{P_1 \xrightarrow{o(\vec{v}),c_1} P_1' \qquad P_2 \xrightarrow{o(\vec{v}),c_2} P_2' \qquad c_1 \neq c_2}{P_1|P_2 \xrightarrow{o(\vec{v}),\emptyset} \dagger}$$

(CONF-REC-EXCP)
$$\frac{P \xrightarrow{\alpha,c} P' \quad P' \equiv P_1|P_2 \quad P_1 \xrightarrow{o(\vec{v}),c_1} P_1' \qquad P_2 \xrightarrow{o(\vec{v}),c_1} P_2'}{P \xrightarrow{\alpha,c} \ddagger}$$

INSTANCIAS

(CORR)
$$\frac{P \xrightarrow{\alpha,c'} P' \qquad c[c']\ \text{definido}}{c \triangleright [P] \xrightarrow{\alpha} c[c'] \triangleright [P']}$$

(I-PAR)
$$\frac{I_1 \xrightarrow{\alpha} I_1'}{I_1|I_2 \xrightarrow{\alpha} I_1'|I_2}$$

SISTEMAS

(SVC-IN)
$$\frac{o \in O}{s_C^O\{P, I, M\} \xrightarrow{o(\vec{v})} s_C^O\{P, I, \overline{o}\langle\vec{v}\rangle|M\}}$$

(NEW)
$$\frac{C_\perp \triangleright [P] \xrightarrow{o(\vec{v})} c \triangleright [P']}{s_C^O\{P, I, \overline{o}\langle\vec{v}\rangle|M\} \xrightarrow{\tau} s_C^O\{P, I \mid c \triangleright [P'], M\}}$$

(DISPATCH)
$$\frac{I \xrightarrow{o(\vec{v})} I'}{s_C^O\{P, I, \overline{o}\langle\vec{v}\rangle|M\} \xrightarrow{\tau} s_C^O\{P, I', M\}}$$

(SVC-NON-IN)
$$\frac{I \xrightarrow{\alpha} I' \qquad \alpha \neq o(\vec{v})}{s_C^O\{P, I, M\} \xrightarrow{\alpha} s_C^O\{P, I', M\}}$$

(S-PAR)
$$\frac{N_1 \xrightarrow{\alpha} N_1'}{N_1 \| N_2 \xrightarrow{\alpha} N_1' \| N_2}$$

(COMM)
$$\frac{N_1 \xrightarrow{o(\vec{v})} N_1' \quad N_2 \xrightarrow{\overline{o}\langle\vec{v}\rangle} N_2'}{N_1 \| N_2 \xrightarrow{\tau} N_1' \| N_2'}$$

Figura 4.1: Sistema de transición por etiquetas para CAB.

Reglas de Procesos

- La regla (IN) indica que un proceso bloqueado en receive, puede seleccionar una de sus ramas realizando la acción de entrada correspondiente. Notar que el segundo elemento de la etiqueta contiene la función parcial $\vec{x_i} \mapsto \vec{v}$ que asigna el valor $\vec{v}$ a la variable de correlación $\vec{x_i}$ (esta información será utilizada en la regla (CORR) para asegurar que la asignación de variables correlacionadas es consistente con la correlación que identifica la instancia).

- La regla,(OUT), indica que una operación de salida reduce a 0.

- La regla (PAR) dice que en una composición en paralelo un subtérmino puede automáticamente ejecutar un paso de computación.

- La regla (SEC) indica que dado dos términos en un orden secuencial, siempre reducirá el primero de ellos. Notar que la sustitución aplica también al segundo termino P_2. Esto es útil para computar un caso como $a(x) + b(x); \overline{c}\langle x \rangle$.

- La regla (REC) se despliega (*unfolds*) la definición de P sustituyendo la variable de procesos X por $\mathbf{rec}_X\ P$.

- La regla (THEN) describe el caso en que el proceso elige internamente la guarda verdadera, dado que el caso en que la guarda es falsa es decripta por la regla (ELSE).

- La regla (NO-EXCP) indica que un proceso P puede ejecutar una acción α sin arrojar una excepción siempre que P pueda realizar dicha acción (primer premisa) y el resultado de realizar dicha acción no lleve a una excepción del tipo $AmbiguousReceive$ y $ConflictingReceive$ (segunda y tercer premisa).

- La regla (AMB-REC-EXCP) indica que una composición en paralelo reduce a este tipo de excepción siempre y cuando se activen dos o más actividades que pueden recibir el mismo mensaje de entrada teniendo diferentes variables de correlación.

- La regla (CONF-REC-EXCP) indica que un término P arroja una excepción del tipo conflicting-receive, cuando dicho término reduce a otro por medio de una acción α y este nuevo término posee al menos dos actividades paralelas que son indistinguibles a la hora de recibir un mensaje.

La semántica se define para procesos bien formados donde † y ‡ no occurren a la izquierda de la relación de transición auxiliar. Existen dos diferencias importantes entre (AMB-REC-EXCP) y (CONF-REC-EXCP). La primer diferencia es que la excepción ambiguous-receive es lanzada cuando el flujo de actividades P intenta realizar una acción de entrada que puede ser tratada por diferentes caminos, mientras que la excepción conflicting-receive es lanzada cuando P realiza una acción α (notar

que puede ser cualquier acción) y luego habilita dos acciones que pueden tratar una misma solicitud. La segunda diferencia es la condición sobre la instanciación de las variables recibidas. La regla (CONF-REC-EXCP) requiere que las variables de correlación sean las mismas, mientras que (AMB-REC-EXCP) exige diferentes instanciaciones (notar que usa diferentes instancias de correlación, c_1 y c_2).

Reglas de Instancias

- La regla (CORR) describe que una instancia de un servicio $c \triangleright [P]$ puede realizar una acción siempre que sea consistente con el valor de correlación de la instancia (condición $c[c']$ definido). Recordar que $c[c']$ está definido siempre que c y c' coincidan en los valores asignados a las variables.

- La regla,(I-PAR), describe que dado un conjunto de instancias reducen a un nuevo término con las mismas instancias salvo que una se verá afectada por la acción α. Notar que la especificación de ActiveBPEL no describe ninguna restricción que asegure que el valor de las instancias debe ser diferente, por lo tanto, una acción podrá ser llevada a cabo por cualquier instancia que sepa responder dicha acción. Esta regla introduce el no determinismo que se verá ilustrado en detalle en la sección 4.4, ejemplo 3.

Reglas de Sistemas

- La regla (SCV-IN) describe que cualquier acción de entrada será agregado al conjunto de mensajes. Notar que para esto es necesario que esa acción de entrada pertenezca al conjunto de operaciones que provee el servicio.

- La regla,(NEW), describe que la creación de instancias se realizará a partir de despachar algún mensaje guardado en el conjunto de mensajes recibidos. Esta regla va a llevarse a cabo, siempre que el mensaje $\overline{o}\langle v \rangle$ se corresponda con una actividad inicial de P. Notar que la nueva instancia es creada a partir de $C_\perp \triangleright [P]$, es decir con todas las variables de correlación sin inicializar.

- La regla,(DISPATCH), describe la entrega de un mensaje a alguna de las instancias activas. Es importante notar que la selección de la instancia está dado por la regla (I-PAR) que es no determinística.

- La regla,(SCV-NON-IN), describe el comportamiento de un servicio para acciones que no son de entrada, es decir un paso interno τ ó un output de alguna instancia.

- La regla,(S-PAR), describe el efecto de una computación que ejecuta la acción independientemente del resto.

- La regla,(COMM), describe el efecto de dos servicios que comunicaran ó sincronizan cuando un servicio que realiza una acción de entrada y un servicio que realiza una acción de salida sobre la misma operación(uno habla y el otro escucha).

Notar que trabajamos módulo congruencia estructural a pesar de que no escribimos explícitamente la regla de inferencia. Por ejemplo omitimos escribir la simétrica para la regla del paralelo.

Notación 4.3.3. *Escribiremos* $\Rightarrow$ *para denotar la relación* $\Rightarrow = \bigcup_{\alpha} \xrightarrow{\alpha}$. *Como abuso de notación, escribiremos* $\Rightarrow$ *para* $\Rightarrow = \bigcup_{\alpha,c} \xrightarrow{\alpha,c}$. *Finalmente escribiremos* $\Rightarrow^n$ *para la composición secuencial de n pasos de* $\Rightarrow$ *y* $\Rightarrow^*$ *para la clausura reflexiva y transitiva de* $\Rightarrow$.

4.3.5. Consideraciones para el Cálculo

Luego de dar la semántica operacional remarcamos decisiones de diseño tomadas principalmente para mantener el cálculo lo más simple posible:

1. Las excepciones no se propagarán a los clientes.

2. Sólo consideraremos operaciones $one - way$. No obstante el cálculo puede extenderse con operaciones $call - return$ en manera análoga a $SOCK$ (usando puertos nuevos).

3. No llevaremos correlación sobre los outputs ya que el foco del trabajo es estudiar las excepciones relacionadas a los conjuntos de correlación definidas sobre operaciones de entrada.

4. La regla (SCV-IN) hace que la comunicación sea asincrónica. En un primer paso por la regla (SCV-IN) el servicio acepta la invocación guardando el mensaje en el conjunto de mensajes recibidos. Luego utiliza el mensaje, ya sea creando una instancia o despachándolo. El único modo para que el servicio se bloquee en output es que no exista un proceso que provea una definición para esa operación.

5. Abstraemos la cola de mensajes como conjuntos, dado que no nos interesa identificar el orden en el que llegan los mensajes.

4.4. Ejemplos en CAB

A continuación se describirán ejemplos en CAB.

Ejemplo 1. Considere el siguiente sistema, compuesto por dos servicios:

$$\begin{aligned} N = \quad & s^{\{o_1,o_2\}}_{\{x\}}\{o_1(x,y); o_2(x,z); \overline{o}\langle y,z\rangle, 0, \emptyset\} \\ \| \; & s'^{O}_{C}\{Q, c \triangleright [\overline{o_1}\langle a,b\rangle; \overline{o_1}\langle d,e\rangle; \overline{o_2}\langle d,f\rangle; \overline{o_2}\langle a,c\rangle], M\} \end{aligned}$$

El servicio s provee de dos operaciones llamadas o_1 y o_2 y utiliza la variable de correlación x. No posee instancias activas y el conjunto de mensajes se encuentra vacío. Cada acción de *receive* (correspondientes a las operaciones o_1 y o_2) está correlacionada por la variable x (primer parámetro de cada operación). Por otra parte, el servicio s' posee una única instancia activa con operaciones de output que permiten sincronizar con el servicio s.

En este ejemplo, la única instancia activa de s' envía una solicitud $\overline{o_1}\langle a,b\rangle$. Dado que el servicio s es capaz de tomar dicha solicitud y llevarla a cabo ambos servicios sincronizan a partir de la regla (COMM). El servicio s, que no posee instancias activas, agrega el mensaje al conjunto de mensajes dejando al sistema como se muestra a continuación.

$$\begin{aligned} N \xrightarrow{\tau} \quad & s^{\{o_1,o_2\}}_{\{x\}}\{o_1(x,y); o_2(x,z); \overline{o}\langle y,z\rangle, 0, \overline{o_1}\langle a,b\rangle\} \\ \| \; & s'^{O}_{C}\{Q, c \triangleright [\overline{o_1}\langle d,e\rangle; \overline{o_2}\langle d,f\rangle; \overline{o_2}\langle a,c\rangle], M\} \end{aligned}$$

En este punto el servicio s tiene dos caminos. Por un lado es capaz de tomar el mensaje guardado y procesar la solicitud enviada por el servicio s'. Por otro lado, puede volver a sincronizar con s' a partir de la solicitud $\overline{o_1}\langle d,e\rangle$. Si consideramos el primer caso, entonces s creará una nueva instancia asociando a la variable de correlación, el valor pasado por parámetro $\{x \mapsto a\}$ por medio de la regla (NEW), como se muestra a continuación.

$$\begin{aligned} \xrightarrow{\tau} \quad & s^{\{o_1,o_2\}}_{\{x\}}\{o_1(x,y); o_2(x,z); \overline{o}\langle y,z\rangle, \{x \mapsto a\} \triangleright [o_2(x,z); \overline{o}\langle b,z\rangle], \emptyset\} \\ \| \; & s'^{O}_{C}\{Q, c \triangleright [\overline{o_1}\langle d,e\rangle; \overline{o_2}\langle d,f\rangle; \overline{o_2}\langle a,c\rangle], M\} \end{aligned}$$

De la misma manera, luego de dos pasos de reducción s activará una nueva instancia para la solicitud $\overline{o_1}\langle d,e\rangle$. Notar que la instancia $\{x \mapsto a\} \triangleright [o_2(x,z); \overline{o}\langle b,z\rangle]$ no es capaz de llevar a cabo $o_1(d,e)$ por lo tanto la única posibilidad que tiene s es la creación de una nueva instancia. El sistema reduce de la siguiente manera:

$$\begin{aligned} \xrightarrow{\tau}\xrightarrow{\tau} \quad & s^{\{o_1,o_2\}}_{\{x\}}\{o_1(x,y); o_2(x,z); \overline{o}\langle y,z\rangle, \{x \mapsto a\} \triangleright [o_2(x,z); \overline{o}\langle b,z\rangle] \mid \\ & \qquad\qquad\qquad\qquad \{x \mapsto d\} \triangleright [o_2(x,z); \overline{o}\langle e,z\rangle], \emptyset\} \\ \| \; & s'^{O}_{C}\{Q, c \triangleright [\overline{o_2}\langle d,f\rangle; \overline{o_2}\langle a,c\rangle], M\} \end{aligned}$$

Luego de dos pasos de reducción, las solicitudes de s' quedan guardadas en el conjunto de mensajes de s como se muestra a continuación.

$$\xrightarrow{\tau}\xrightarrow{\tau} \quad s^{\{o_1,o_2\}}_{\{x\}}\{o_1(x,y); o_2(x,z); \overline{o}\langle y,z\rangle, \{x \mapsto a\} \triangleright [o_2(x,z); \overline{o}\langle b,z\rangle] \mid \{x \mapsto d\} \triangleright [o_2(x,z); \overline{o}\langle e,z\rangle], \overline{o_2}\langle d,f\rangle | \overline{o_2}\langle a,c\rangle\}$$
$$\parallel \quad s'^{O}_{C}\{Q, c \triangleright [0], M\}$$

En este punto, el mensaje $\overline{o_2}\langle d,f\rangle$ será entregado a la instancia correlacionada por $\{x \mapsto d\}$, y el mensaje $\overline{o_2}\langle a,c\rangle$ será entregado a la instancia correlacionada por $\{x \mapsto a\}$, como se muestra a continuación.

$$\xrightarrow{\tau} \quad s^{\{o_1,o_2\}}_{\{x\}}\{o_1(x,y); o_2(x,z); \overline{o}\langle y,z\rangle, \{x \mapsto a\} \triangleright [o_2(x,z); \overline{o}\langle b,z\rangle] \mid \{x \mapsto d\} \triangleright [\overline{o}\langle e,f\rangle], \overline{o_2}\langle a,c\rangle\}$$
$$\parallel \quad s'^{O}_{C}\{Q, c \triangleright [0], M\}$$
$$\xrightarrow{\tau} \quad s^{\{o_1,o_2\}}_{\{x\}}\{o_1(x,y); o_2(x,z); \overline{o}\langle y,z\rangle, \{x \mapsto a\} \triangleright [\overline{o}\langle b,c\rangle] \mid \{x \mapsto d\} \triangleright [\overline{o}\langle e,f\rangle], \emptyset\}$$
$$\parallel \quad s'^{O}_{C}\{Q, c \triangleright [0], M\}$$

Es importante notar que no es posible entregar los mensajes de otra manera.

Ejemplo 2. Considere el siguiente sistema que muestra la posibilidad de un servicio que cuenta con varios conjuntos de correlación. La posibilidad de definir un servicio con múltiples conjuntos de correlación es una característica de los lenguajes de orquestación que suele ser usada para estructurar la comunicación con más de un servicio.

$$\begin{aligned} N \quad = \quad & s^{\{o_1,o_2\}}_{\{x,y\}}\{P, \{x \mapsto a, y \mapsto b\} \triangleright [(o_1(x,z) \mid o_2(y,w))], \emptyset\} \\ \parallel \quad & s_1{}^{0_1}_{C_1}\{P_1, c_1 \triangleright [\overline{o_1}\langle a,d\rangle; R_1], M_1\} \\ \parallel \quad & s_2{}^{0_2}_{C_2}\{P_2, c_2 \triangleright [\overline{o_2}\langle b,e\rangle; R_2], M_2\} \end{aligned}$$

En este caso, las instancias de s pueden identificarse usando independientemente x, y ó la combinación de ambas. En particular, la comunicación entre instancias de s_1 e instancias de s será a partir de la operación o_1 y la variable de correlación x, mientras que la comunicación con instancias de s_2 será a partir de la operación o_2 con la variable de correlación y.

Ejemplo 3. En ActiveBPEL puede ocurrir que existan diferentes instancias, cuyo valor asociado a la variable de correlación coincida. Este ejemplo permite ilustrar como la semántica de CAB se corresponde con el comportamiento de ActiveBPEL para estos casos. Considere el siguiente sistema

$$N \quad = \quad s^{\{o_1,o_2\}}_{\{x\}}\{o_1(x); o_2(x), 0, \emptyset\} \parallel s_1{}^{O_1}_{C_1}\{P_1, c_1 \triangleright [\overline{o_1}\langle a\rangle; \overline{o_1}\langle a\rangle], M\}$$

debe reducir después de dos pasos como se muestra a continuación.

$$N \quad \xrightarrow{\tau}\xrightarrow{\tau} \quad s^{\{o_1,o_2\}}_{\{x\}}\{o_1(x); o_2(x), 0, \overline{o_1}\langle a\rangle | \overline{o_1}\langle a\rangle\} \parallel s_1{}^{O_1}_{C_1}\{P_1, c_1 \triangleright [0], M\}$$

En este punto, s creará una nueva instancia correlacionada con la asignación $\{x \mapsto a\}$, como se muestra a continuación.

$$\xrightarrow{\tau} \quad s^{\{o_1,o_2\}}_{\{x\}}\{o_1(x); o_2(x), \{x \mapsto a\} \triangleright [o_2(x)], \overline{o_1}\langle a\rangle\} \parallel s_1{}^{O_1}_{C_1}\{P_1, c_1 \triangleright [0], M\}$$

Hasta aquí, el servicio s tiene una única instancia asociada con la correlación $\{x \mapsto a\}$ y un mensaje guardado $\overline{o_1}\langle a\rangle$. Notar que este mensaje no puede ser entregado a la única instancia activa de s, sin embargo el servicio s creará una nueva instancia a partir de la actividad de inicio de su definición, es decir, la operación de receive o_1. En consecuencia, el sistema quedará como se muestra a continuación

$$\xrightarrow{\tau} \quad s^{\{o_1,o_2\}}_{\{x\}}\{o_1(x); o_2(x), \{x \mapsto a\} \triangleright [o_2(x)] | \{x \mapsto a\} \triangleright [o_2(x)], \emptyset\} \parallel s_1{}^{O_1}_{C_1}\{P_1, c_1 \triangleright [0], M\}$$

Ahora s posee dos instancias con el mismo valor de correlación. Si consideramos ahora que s recibe un mensaje $\overline{o_2}\langle a\rangle$, el sistema reduce como se muestra a continuación.

$$\xrightarrow{o_2(a)} \quad s^{\{o_1,o_2\}}_{\{x\}}\{o_1(x); o_2(x), \{x \mapsto a\} \triangleright [o_2(x)] | \{x \mapsto a\} \triangleright [o_2(x)], \overline{o_2}\langle a\rangle\} \parallel \ldots$$

Es importante notar que el mensaje $\overline{o_2}\langle a\rangle$ será entregado de manera no determinística (uso de la regla I-PAR) a cualquiera de las dos instancias activas. Este ejemplo ilustra que el mecanismo de correlación que ofrece ActiveBPEL no asegura que existan dos instancias distintas identificadas por valores de correlación distintos, es decir, pueden existir instancias indistinguibles.

Ejemplo 4. Considere el siguiente sistema:

$$N = s^{\{o_1,o_2\}}_{\{x,y\}}\{o_1(x,y); (o_2(x)|o_2(y)), 0, \emptyset\}$$

y los siguientes pasos de computación:

$$\begin{array}{ll}
\xrightarrow{o_1(a,a)} & s^{\{o_1,o_2\}}_{\{x,y\}}\{o_1(x,y); (o_2(x)|o_2(y)), 0, \overline{o_1}\langle a,a\rangle\} \\
\xrightarrow{o_2(a)} & s^{\{o_1,o_2\}}_{\{x,y\}}\{o_1(x,y); (o_2(x)|o_2(y)), 0, \overline{o_2}\langle a\rangle | \overline{o_1}\langle a,a\rangle\} \\
\xrightarrow{\tau} & s^{\{o_1,o_2\}}_{\{x,y\}}\{o_1(x,y); (o_2(x)|o_2(y)), \{x \mapsto a, y \mapsto a\} \triangleright [o_2(x) \mid o_2(y)], \overline{o_2}\langle a\rangle\}
\end{array}$$

En este momento, el servicio s puede procesar el mensaje $\overline{o_2}\langle a\rangle$ entregándolo a la única instancia activa, la cual arrojará una excepción. Notar que el proceso reduce a partir de la regla (AMB-REC-EXCP).

$$\text{(AMB-REC-EXCP)}\ \frac{o_2(x) \xmapsto{o_2(a),\{x\mapsto a\}} 0 \qquad o_2(y) \xmapsto{o_2(a),\{y\mapsto a\}} 0 \qquad \{x \mapsto a\} \neq \{y \mapsto a\}}{o_2(x)|o_2(y) \xrightarrow{o_2(a),\emptyset} \dagger}$$

En consecuencia, el sistema reduce como se muestra a continuación.

$$\xrightarrow{\tau} \quad s^{\{o_1,o_2\}}_{\{x,y\}}\{o_1(x,y); (o_2(x)|o_2(y)), \{x \mapsto a, y \mapsto a\} \triangleright [\dagger], \emptyset\}$$

Ejemplo 5. El siguiente sistema muestra un escenario posible donde se arroja una excepción del tipo conflicting receive. Considere el sistema definido de la siguiente manera

$$N = s^{\{o_1,o_2\}}_{\{x\}}\{o_1(x); (o_2(x)|o_2(x)), 0, \emptyset\}$$

y la siguiente computación

$$\xrightarrow{o_1(a)} \quad s^{\{o_1,o_2\}}_{\{x\}}\{o_1(x); (o_2(x)|o_2(x)), 0, \overline{o_1}\langle a\rangle\}$$

La excepción es alcanzada cuando el servicio crea una nueva instancia a partir de despachar el mensaje $\overline{o_1}\langle a\rangle$.

(CONF-REC-EXCP)

$$\frac{o_1(x); (o_2(x)|o_2(x)) \xrightarrow{o_1(a),x\mapsto a} o_2(x)|o_2(x) \quad o_2(x) \xrightarrow{o_2(a),x\mapsto a} 0 \qquad o_2(x) \xrightarrow{o_2(a),x\mapsto a} 0}{P \xrightarrow{o_1(a),x\mapsto a} \ddagger}$$

Ejemplo 6. El siguiente escenario muestra que los mensajes que recibe un servicio pueden guardarse en el conjunto de mensajes recibidos de manera indefinida. Existen escenarios donde se invoca una actividad que no es una operación inicial (por lo tanto no se crea una instancia nueva), ni está disponible para ser ejecutada por ninguna instancia activa, sin embargo el mensaje queda guardado en el conjunto de mensajes. Por ejemplo,

$$\begin{array}{lll} N & = & s^{\{o_1,o_2\}}_{\{x,y\}}\{o_1(x); o_2(x), \{x \mapsto a\} \triangleright [0], \emptyset\} \parallel s_1{}^{O_1}_{C_1}\{P_1, c \triangleright [\overline{o_2}\langle a\rangle], M\} \\ \xrightarrow{\tau} & & s^{\{o_1,o_2\}}_{\{x,y\}}\{o_1(x); o_2(x), \{x \mapsto a\} \triangleright [0], \overline{o_2}\langle a\rangle\} \parallel s_1{}^{O_1}_{C_1}\{P_1, c \triangleright [0], M\} \end{array}$$

4.5. Ejemplo Final

En este capítulo presentaremos un escenario de negocios donde mostraremos el cálculo desarrollado. Comenzaremos con una descripción informal de un problema para luego presentar la representación formal del mismo.

4.5.1. Descripción informal

La publicación de un artículo en una revista científica es llevado a cabo a partir de superar una serie de etapas. Al comenzar, los autores elaboran un resumen (o *abstract*) del trabajo a publicar y se lo envían al editor para que lo considere. Luego, los autores comienzan la elaboración del artículo en si.

El trabajo de elaborar un artículo consta de dos tareas principales: la redacción del artículo en si, y la realización de los experimentos que comprueban la hipótesis planteada. Una vez que se completan ambas tareas, los autores pasan a la etapa final

de pulir los detalles del mismo (corrección ortográfica, diagramación, consistencia de estilos, etc.).

Una vez que el artículo está considerado listo por los autores, éstos lo envían nuevamente al editor de la revista. El editor delega la responsabilidad de la revisión del artículo en dos científicos revisores. Estos revisores leerán separadamente el artículo y tomarán una decisión individual acerca de si el artículo merece ser publicado o no. Si ambos revisores están de acuerdo en que el artículo debe ser publicado, se pasa a la siguiente fase sin más, en caso contrario se informa a los autores que el artículo no será considerado.

Cuando un artículo es aprobado para publicación, se requiere que el derecho de *copyright* del mismo sea cedido al editor. Para ello, se espera la autorización de cesión al editor por parte de los autores. Una vez que la cesión está en manos del editor, el artículo se incluye en la publicación y pasa a imprenta.

4.5.2. Participantes

- Autor
- Editor
- Aprobador

4.5.3. Descripción Formal

El problema de negocio será representado mediante la composición de servicios. En particular, definiremos un modelo de orquestación desde la visión del Editor. El sistema de servicios será compuesto por los siguientes servicios:

- El servicio Autor (SA)
- El servicio Editor (SE)
- El servicio Aprobador (SC)

Previamente a describir el comportamiento del sistema a partir del cálculo, describiremos la evolución del sistema en lenguaje natural:

1. El autor envía el resumen del trabajo al editor. El nombre del mensaje será obtenerResumen(or) y modelaremos el resumen mediante la variable *pResumen*.
2. El autor envía el artículo a través de la operación obtenerArticulo(oa), donde el contenido del artículo sera modelado mediante la variable *pArticulo*.
3. El editor envía el artículo a dos aprobadores distintos, esto se realizará a partir de la operacion invocarAprobador(ia) enviándo el artículo en cuestión.

4. Una vez leído el artículo el aprobador envía la aprobación del artículo de manera independiente al otro aprobador. Para esto tendremos la actividad recibirAprobacion(ra) con el parámetro *pOk* para modelar si fue o no aprobado.

5. El artículo será aprobado por los dos aprobadores y se le informará al autor a través de la operación articuloAprobado(aa). Notar que si lo hubiesen desaprobado se haría a través de la operación desaprobarArticulo(da).

6. Finalmente, el autor envía los derechos a través del mensaje obtenerDerechos(od).

Además de las variables de estado descriptas anteriormente, el servicio del editor contará con variables correlacionadas. Por un lado, la variable *cArticulo* utilizada para identificar al artículo a aprobar, y las variables *cAprobador1*,*cAprobador2* utilizadas para identificar a los aprobadores del artículo. Usaremos *vResumen, vArticulo, vOk* como las constantes usadas para modelar el contenido de las variables correspondientes a los parámetros *pResumen, pArticulo, pOk*.

4.5.4. Representación con el cálculo

El sistema será representado a partir de la composición de servicios:

$$N = SA \| \ SE \| \ SC$$

donde:

$$SC = \quad s_{CC}^{O}\{ia(pArticulo); \overline{ra}\langle vAprobador, vOk\rangle; , 0, \emptyset\}$$

$$SA = \quad s_{AC}^{O}\{Q_A, I_A, M\}$$

$$I_A = c\triangleright[\overline{or}\langle 1, vResumen\rangle; \overline{oa}\langle 1, vArticulo\rangle; (aa(cArticulo); \overline{od}\langle 1\rangle; +da(cArticulo);)]$$

$$SE = \quad s_{E\{carticulo,caprobador\}}^{\{or,oa,ra,od\}}\{Q_E, 0, \emptyset\}$$

$$\begin{aligned} Q_E \ = \ & or(cArticulo, pResumen); oa(cArticulo, pArticulo); \\ & \overline{ia}\langle pArticulo\rangle; \overline{ia}\langle pArticulo\rangle; \\ & (ra(cAprobador1, pOk1)|ra(cAprobador2, pOk2)); \\ & if(pOk1) \ then(if(pOk2) \ then \ \overline{aa}\langle cArticulo\rangle; od(cArticulo) \ else \ \overline{da}\langle cArticulo\rangle) \\ & else \ \overline{da}\langle cArticulo\rangle; \end{aligned}$$

- *El autor envía el resumen del trabajo al editor. El nombre del mensaje será obtenerResumen(or) y modelaremos el resumen mediante la variable pResumen*

La regla de COMM nos garantiza que el servicio autor(SA) y el servicio editor(SE) sincronizan a partir del mensaje $or(pAutor, pResumen)$. El servicio autor reduce como se muestra a continuación:

$$\xrightarrow{\tau} \quad s_{AC}^{O}\{Q_A, c \triangleright [\overline{oa}\langle 1, vArticulo\rangle; (aa(cArticulo); \overline{od}\langle 1\rangle; +da(cArticulo);], M\}$$

El servicio del editor en un paso de computación guarda el mensaje en el conjunto de mensajes. En un segundo paso, lo saca del conjunto de mensajes creando una nueva instancia correlacionada por el valor 1. De esta manera el artículo será identificado por el valor uno.

$$\begin{array}{ll} \xrightarrow{\tau} & s_{E\,\{pArticulo,pAprobador\}}^{\{or,oa,ra,od\}}\{Q_E, 0, \overline{oa}\langle 1, vArticulo\rangle;\} \\ \xrightarrow{\tau} & s_{E\,\{pArticulo,pAprobador\}}^{\{or,oa,ra,od\}}\{Q_E, I_E, \emptyset\} \end{array}$$

$$\begin{array}{rcl} I_E & = & \{cArticulo \mapsto 1\} \triangleright [oa(cArticulo, pArticulo); \\ & & \overline{ia}\langle pArticulo\rangle; \overline{ia}\langle pArticulo\rangle; \\ & & (ra(cAprobador1, pOk1)|ra(cAprobador2, pOk2)); \\ & & if(pOk1)\ then(if(pOk2)\ then\ \overline{aa}\langle cArticulo\rangle; od(cArticulo)\ else\ \overline{da}\langle cArticulo\rangle) \\ & & else\ \overline{da}\langle cArticulo\rangle;] \end{array}$$

- *El autor envía el artículo a través de la operación obtenerArticulo(oa), donde el contenido del artículo sera modelado mediante la variable pArticulo*

Por la regla COMM SA y SE vuelven a sincronizar como el primer paso, sin embargo SE si bien agrega el mensaje en el conjunto de mensajes y luego lo entrega, al despacharlo no crea una nueva instancia ya que existe una instancia activa con el valor de correlación 1. En consecuencia, se le entrega este mensaje a la instancia activa.

El servicio autor reduce quedando la instancia como:

$$\xrightarrow{\tau} \quad s_{AC}^{O}\{Q_A, c \triangleright [(aa(cArticulo); \overline{od}\langle 1\rangle; +da(cArticulo);], M\}$$

El servicio editor reduce luego de dos pasos de computación a:

$$\xrightarrow{\tau} \quad s_{E\,\{pArticulo,pAprobador\}}^{\{or,oa,ra,od\}}\{Q_E, I_E, \emptyset\}$$

$$\begin{array}{rcl} I_E & = & \{cArticulo \mapsto 1\} \triangleright [\overline{ia}\langle pArticulo\rangle; \overline{ia}\langle pArticulo\rangle; \\ & & (ra(cAprobador1, pOk1)|ra(cAprobador2, pOk2)); \\ & & if(pOk1)\ then(if(pOk2)\ then\ \overline{aa}\langle cArticulo\rangle; od(cArticulo)\ else\ \overline{da}\langle cArticulo\rangle) \\ & & else\ \overline{da}\langle cArticulo\rangle;] \end{array}$$

- *El editor envía el artículo a dos aprobadores distintos, esto se realizará a partir de la operacion invocarAprobador(ia) enviándo el artículo en cuestión.*

En esta instancia entra en juego el servicio aprobador(SC) responsable de crear una instancia distinta por cada invocación realizada por el servicio editor. De esta manera, SE envía los mensajes correspondientes a la invocación dejando la intancia como:

$$\xrightarrow{\tau}\xrightarrow{\tau} \quad s_{E\,\{pArticulo,pAprobador\}}^{\{or,oa,ra,od\}}\{Q_E, I_E, \emptyset\}$$

$$\begin{aligned} I_E \;=\; & \{cArticulo \mapsto 1\} \triangleright [(ra(cAprobador, pOk1)|ra(cAprobador, pOk2)); \\ & if(pOk1)\ then(if(pOk2)\ then\ \overline{aa}\langle cArticulo\rangle; od(cArticulo)\ else\ \overline{da}\langle cArticulo\rangle) \\ & else\ \overline{da}\langle cArticulo\rangle;] \end{aligned}$$

El SC por cada invocación realizará un paso de computación donde se guarda el mensaje en el conjunto de mensajes de llegada y luego un paso para crear la instancia.

$$\xrightarrow{\tau}\xrightarrow{\tau}\xrightarrow{\tau}\xrightarrow{\tau} s_C C^O\{Q_C, c \triangleright [ia(pArticulo); \overline{ra}\langle pOk\rangle;] \mid c \triangleright [ia(pArticulo); \overline{ra}\langle pOk\rangle;], \emptyset\}$$

- *Una vez leído el artículo el aprobador envía la aprobación del artículo de manera independiente al otro aprobador. Para esto tendremos la actividad recibirAprobacion(ra) con el parámetro pOk para modelar si fue o no aprobado.*

Cada instancia de aprobador se encarga de enviar el mensaje recibirAprobador con el parámetro *pOk* en *true*. De esta manera, el SC queda sin instancias activas. Por su parte, el servicio del editor recibe los mensajes correspondientes a cada aprobador. Asumiremos que cada aprobador envía el mensaje con un valor de correlación distinto. Por consiguiente el servicio editor reducirá quedando como:

$$s_{E\,\{pArticulo,pAprobador\}}^{\{or,oa,ra,od\}}\{Q_E, I_E, \emptyset\}$$

$$\begin{aligned} I_E \;=\; & \{cArticulo \mapsto 1, cAprobador1 \mapsto a, cAprobador2 \mapsto b\} \triangleright [\\ & if(true)\ then(if(true)\ then\ \overline{aa}\langle cArticulo\rangle; od(cArticulo)\ else\ \overline{da}\langle cArticulo\rangle) \\ & else\ \overline{da}\langle cArticulo\rangle;]; \end{aligned}$$

- *El artículo será aprobado por los dos aprobadores y se le informará al autor a través de la operación articuloAprobado(aa).*

A partir de la regla THEN, el servicio editor invoca al servicio autor mediante el mensaje *articuloAprobado*. La instancia del servicio editor luego de la invocación

$$I_E \;=\; \{cArticulo \mapsto 1, cAprobador1 \mapsto a, cAprobador2 \mapsto b\} \triangleright [od(cArticulo)];$$

- *Finalmente, el autor envía los derechos a través del mensaje obtenerDerechos(od).*

A partir de que el autor envía el mensaje que cede los derechos y el servicio del editor recibe ese mensaje, ambas instancias terminan.

Capítulo 5

Servicios libres de excepciones de correlación

En este capítulo analizaremos la definición de un sistema de tipos elemental que permite caracterizar estáticamente servicios que son libres de excepciones de correlación. Comenzaremos introduciendo la noción de sistemas libres de excepciones.

5.1. Sistemas libres de excepciones

Como se muestra en los ejemplos del capítulo anterior, existen escenarios en donde un servicio puede alcanzar una excepción. En particular, estas excepciones ocurren cuando el proceso activa concurrentemente dos o más *receives* sobre la misma operación. En los casos en que las acciones que se activan concurrentemente usan las mismas variables de correlación, entonces nos encontramos con una excepción del tipo conflicting-receive. En el caso en que las variables son distintas pero su valor es el mismo, se alcanza una excepción del tipo ambiguous-receive.

Las siguientes definiciones introducen formalmente estas nociones.

Definición 5.1.1. *Un servicio* $s_C^O\{P, 0, \emptyset\}$ *es libre de excepciones ambiguous-receive sii* $\forall I$ *tal que* $s_C^O\{P, 0, \emptyset\} \Rightarrow^* s_C^O\{P, I, M\}$*, entonces* $I \not\equiv c \triangleright [\dagger]||I'$.

La idea intuitiva de la definición es llamar a un servicio libre de excepciones ambiguous-receive si no hay manera de que el servicio reduzca en una cantidad arbitraria de pasos a un servicio compuesto por alguna instancia que alcanzó una excepción de tipo ambiguous-receive.

Definición 5.1.2. *Un servicio* $s_C^O\{P, 0, \emptyset\}$ *es libre de excepciones conflicting-receive sii* $\forall I$ *tal que* $s_C^O\{P, 0, \emptyset\} \Rightarrow^* s_C^O\{P, I, M\}$*, entonces* $I \not\equiv c \triangleright [\ddagger]||I'$.

Esta definición es análoga a la anterior pero considera excepciones de tipo conflicting-receive.

Definición 5.1.3. *Un servicio es libre de excepciones de correlación cuando es libre de excepciones ambiguous-receive y conflicting-receive.*

5.2. Sistema de tipos

Un sistema de tipos nos permitirá caracterizar estáticamente programas de forma tal que se puedan garantizar que los sistemas a construir sean libres de excepciones de correlación.

Nuestro sistema de tipos considera el siguiente juicio de tipado para procesos:

$$\Gamma \vdash P : \mathcal{T}$$

El tipo $\mathcal{T} : O \mapsto \mathcal{P}_f(\mathcal{V}^*)$ es una función parcial definida sobre el conjunto de nombres de operaciones que ocurren en P en el conjunto de partes de las cadenas finitas de nombres de variables. De esta manera, estaremos describiendo para cada operación de entrada, cuál es el conjunto de tuplas de variables que utilizan como parámetros. Por otra parte, Γ es una función parcial de variables de procesos a tipos, es decir, asigna un tipo a cualquier variable de proceso en P. Por ejemplo $\mathcal{T}(o) = \{\langle x, y\rangle, \langle z, x\rangle\}$, significa que la operación de entrada o ocurre en P con parámetros $\langle x, y\rangle$ ó $\langle z, x\rangle$.

La composición de tipos se define como $(\mathcal{T}_1 \oplus \mathcal{T}_2)(o) = \mathcal{T}_1(o) \cup \mathcal{T}_2(o)$.

5.2.1. Reglas de Tipado

La Figura 5.1 muestra las reglas de tipado. La idea principal es que las reglas de tipado coleccionan los parámetros formales que son utilizados por cada operación y se requiere ausencia de conflictos entre operaciones que pueden activarse concurrentemente(como se explica en detalle más adelante).

(ZERO)
$$\Gamma \vdash 0 : \emptyset$$

(OUTPUT)
$$\Gamma \vdash \overline{o}\langle\vec{v}\rangle : \emptyset$$

(X)
$$X \mapsto \mathcal{T}, \Gamma \vdash X : \mathcal{T}$$

(INPUT)
$$\frac{\Gamma \vdash P : \mathcal{T}}{\Gamma \vdash o(\vec{x}); P : \mathcal{T} \oplus \{o \mapsto \vec{x}\}}$$

(SUM)
$$\frac{\Gamma \vdash o_1(\vec{x_1}); P_1 : \mathcal{T}_1 \quad \ldots \quad \Gamma \vdash o_n(\vec{x_n}); P_n : \mathcal{T}_n}{\Gamma \vdash \Sigma_i o_i(\vec{x_i}); P_i : \bigoplus_i \mathcal{T}_i}$$

(SEQ)
$$\frac{\Gamma \vdash P_1 : \mathcal{T}_1 \qquad \Gamma \vdash P_2 : \mathcal{T}_2}{\Gamma \vdash P_1; P_2 : \mathcal{T}_1 \oplus \mathcal{T}_2}$$

(PAR)
$$\frac{\Gamma \vdash P_1 : \mathcal{T}_1 \qquad \Gamma \vdash P_2 : \mathcal{T}_2 \qquad \mathcal{T}_1 \between \mathcal{T}_2}{\Gamma \vdash P_1 | P_2 : \mathcal{T}_1 \oplus \mathcal{T}_2}$$

(REC)
$$\frac{X \mapsto \mathcal{T}, \Gamma \vdash P : \mathcal{T}}{X \mapsto \mathcal{T}, \Gamma \vdash \mathbf{rec}_X\ P : \mathcal{T}}$$

(IF)
$$\frac{\Gamma \vdash P_1 : \mathcal{T}_1 \qquad \Gamma \vdash P_2 : \mathcal{T}_2}{\Gamma \vdash \mathbf{if}\ v = v'\ \mathbf{then}\ P_1\ \mathbf{else}\ P_2 : \mathcal{T}_1 \oplus \mathcal{T}_2}$$

Figura 5.1: Tipando reglas

A continuación comentaremos informalmente las reglas de tipado:

- (ZERO) dado que en 0 no aparecen operaciones de entrada, el tipo de 0 es la función cuyo dominio es vacío.
- (INPUT) agrega al tipo de P la asociación entre la operación de entrada o y la tupla de parámetros $\vec{x}$ correspondientes al prefijo de entrada $o(\vec{x})$.
- (SUM) el tipo es la composición de los tipos de $o_i(\vec{x_i}); P_i$.
- (OUTPUT) dado que $\overline{o}\langle\vec{v}\rangle$ no contiene operaciones de entrada, el tipo es la función con dominio vacío.
- (SEQ) el tipo de dos términos compuestos secuencialmente es la composición de los tipos asignados a cada uno de los términos.
- (REC) el tipo de la recursión es el tipo del proceso P sobre el que se aplica la recursión. Notar que la variable X sobre la que se aplica la sustitución tiene que tener el mismo tipo que P, es por eso que se agrega al contexto *Gamma* la función que asigna a la variable de proceso el tipo.
- (X) el tipo de la variable de proceso X es el tipo que tiene esa variable en el contexto *Gamma*.
- (IF) el tipo es la composición de los tipos asignados a cada rama.
- (PAR) esta regla agrega una precondición sobre la compatibilidad $\between$ de los tipos de los términos que se encuentran en paralelo. Esta condición de compatibilidad establece restricciones sobre las operaciones que pueden activarse concurrentemente a fin de evitar excepciones de correlación. Estas condiciones se analizan en detalle en la siguiente sección.

5.3. Condición de Compatibilidad

En la regla de tipado (PAR) usamos una condición de compatibilidad genérico $\between$ entre dos tipos $\mathcal{T}_1$ y $\mathcal{T}_2$. En esta sección analizaremos tres definiciones posibles para $\between$ en relación a los tipos de excepciones que permiten identificar. En particular, consideramos:

$$\mathcal{T}_1 \between_c \mathcal{T}_2 = \forall o.\mathcal{T}_1(o) \cap \mathcal{T}_2(o) = \emptyset$$
$$\mathcal{T}_1 \between_a \mathcal{T}_2 = \forall o.\#(\mathcal{T}_1(o) \cup \mathcal{T}_2(o)) > 1 \Rightarrow (\mathcal{T}_1(o) = \emptyset \vee \mathcal{T}_2(o) = \emptyset)$$
$$\mathcal{T}_1 \between_e \mathcal{T}_2 = \forall o.\mathcal{T}_1(o) = \emptyset \vee \mathcal{T}_2(o) = \emptyset$$

La definición para $\between_c$ requiere que las operaciones que son comunes entre ambos tipos, es decir $\mathcal{T}_1$ y $\mathcal{T}_2$, tengan diferentes parámetros de entrada. Mostramos luego que esta condición es suficiente para evitar excepciones del tipo *conflicting receive*. La condición de compatibilidad definida como $\between_a$ requiere que o bien no

haya operaciones en común entre ambos tipos, o si hay una operación en común, deben tener los mismos parámetros. Finalmente $\oslash_e$ prohíbe que existan dos o más acciones concurrentes con el mismo nombre de operación(es la conjunción de las restricciones).

Consideremos algunos ejemplos para explicar la condición de compatibilidad.

1. Sea P el siguiente flujo de actividades: $o_1(x)|o_1(x)$, el tipo definido sobre o_1 para la rama de la izquierda es $\mathcal{T}_1(o_1) = \{\langle x\rangle\}$ y para la rama derecha es $\mathcal{T}_2(o_1) = \{\langle x\rangle\}$. Este ejemplo muestra que se viola la condición de compatibilidad para $\oslash_c$, ya que se encuentran en paralelo dos actividades con el mismo nombre de operación por lo que la intersección es distinta de vacío. Ahora chequeemos que ocurre con la condición de compatibilidad de $\oslash_a$. La operación en común es o_1 y el cardinal de la unión de los tipos es 1, por lo tanto no se violó la condición de compatibilidad de $\oslash_a$ ya que el antecedente es falso.

2. Sea P el siguiente flujo de actividades: $o_1(x); o_1(y)|o_1(z)$, el tipo definido sobre o_1 para la rama de la izquierda es $\mathcal{T}_1(o_1) = \{\langle x\rangle, \langle y\rangle\}$ y para la rama derecha es $\mathcal{T}_2(o_1) = \{\langle z\rangle\}$. La operación en común es o_1 por lo tanto violamos $\oslash_c$. Por otra parte, la unión de los tipos es $\{\langle x\rangle, \langle y\rangle, \langle z\rangle\}$, por lo que el cardinal es 3. Dado que tenemos una misma operación en ramas distintas y las variables son distintas nos encontramos violando $\oslash_a$.

3. Sea P el siguiente flujo de actividades: $o_1(x); o_1(y)|o_2(z)$, el tipo definido sobre o_1 para la rama de la izquierda es $\mathcal{T}_1(o_1) = \{\langle x\rangle, \langle y\rangle\}$ y para la rama derecha es $\mathcal{T}_2(o_1) = \emptyset$. Dado que no tienen operación en común no violamos $\oslash_c$. Por otra parte, la unión de los tipos es $\{\langle x\rangle, \langle y\rangle\}$, por lo que el cardinal es 2. Dado que tenemos una operación definida en una única rama tampoco violamos la condición de compatibilidad $\oslash_a$.

5.3.1. Propiedades

A continuación presentamos una serie de resultados auxiliares que serán utilizados en la demostración de los principales resultados de esta sección. Los primeros tres establecen que el tipo de un proceso captura los parámetros formales de las operaciones de entrada que están activas en un proceso.

Proposición 5.3.1. *Sea P un proceso. Si $\Gamma \vdash P : \mathcal{T}$ y $P \xmapsto{o(v_1,\ldots,v_n),c'} P'$, entonces existe $\langle y_1, \ldots, y_n\rangle \in \mathcal{T}(o)$ tal que $dom(c') = \langle y_1, \ldots, y_n\rangle$.*

Demostración. La prueba sigue por inducción en la derivación $P \xmapsto{o(v_1,\ldots,v_n),c'} P'$.

- (IN). $P = \Sigma_i o_i(\vec{x_i}); P_i$. En este caso, $P' = P_i$ para algún i. Por la regla de tipado, sabemos que $\mathcal{T}(o) = \bigoplus_i \mathcal{T}_i(o)$ donde $\Gamma_i \vdash o_i(\vec{x_i}); P_i : \mathcal{T}_i$. Por otra parte, $\Gamma_i \vdash o_i(\vec{x_i}); P_i : \mathcal{T}_i' \oplus \{o \mapsto \vec{x_i}\}$. Es fácil ver que $\vec{x_i} \in \mathcal{T}(o)$ porque $\vec{x_i} \in \mathcal{T}_i(o)$.

- (OUT). No puede ser aplicada.
- Los casos restantes se demuestran simplemente usando hipótesis inductiva.

□

Proposición 5.3.2. *Sea P un proceso. Si $\Gamma \vdash P : \mathcal{T}$ y $P \xrightarrow{o(v_1,\ldots,v_n),c'} P'$, entonces una de las siguientes condiciones vale:*

1. $\exists\ \langle y_1, \ldots, y_n\rangle \in \mathcal{T}(o)$ *tal que* $dom(c') = \langle y_1, \ldots, y_n\rangle$.
2. $P' = \dagger$.

Demostración. La prueba sigue inmediatamente por análisis de la regla utilizada para derivar $P \xrightarrow{o(v_1,\ldots,v_n),c'} P'$ y uso de la Proposición 5.3.1. □

Proposición 5.3.3. *Sea P un proceso. Si $\Gamma \vdash P : \mathcal{T}$ y $c \triangleright [P] \xrightarrow{o(v_1,\ldots,v_n),c'} c[c'] \triangleright [P']$, entonces una de las siguientes condiciones vale:*

1. $\exists\ \langle y_1, \ldots, y_n\rangle \in \mathcal{T}(o)$ *tal que* $dom(c') = \langle y_1, \ldots, y_n\rangle$.
2. $P' = \dagger$.

Demostración. La prueba sigue inmediatamente por Proposición 5.3.2. □

Proposición 5.3.4. *Sea $\mathcal{T}_1$, $\mathcal{T}_2$ y $\mathcal{T}_1'$ tipos de procesos. Si $\mathcal{T}_1 \between \mathcal{T}_2$ y $\mathcal{T}_1'(o) \subseteq \mathcal{T}_1(o)$ para toda o implica $\mathcal{T}_1' \between \mathcal{T}_2$.*

Demostración. La prueba se hace para cada operador de compatibilidad:

- Para $\between_c$. Por definición, $\mathcal{T}_1 \between_c \mathcal{T}_2 = \forall o. \mathcal{T}_1(o) \cap \mathcal{T}_2(o) = \emptyset$. Como $\mathcal{T}_1'(o) \subseteq \mathcal{T}_1(o)$ es inmediato que $\forall o. \mathcal{T}_1'(o) \cap \mathcal{T}_2(o) = \emptyset = \mathcal{T}_1' \between_c \mathcal{T}_2$.
- Para $\between_a$, $\between_e$ es análogo.

□

Proposición 5.3.5. *Sea P un proceso bien tipado y c una instancia de correlación. Si $\Gamma \vdash P : \mathcal{T}$ entonces $\Gamma \vdash Pc : \mathcal{T}$.*

Demostración. La prueba es por inducción en la estructura de P

- (SUM). $P = \Sigma_i o_i(\vec{x_i}); P_i$, luego $Pc = (\Sigma_i o_i(\vec{x_i}); P_i)c$ que por definición de sustitución $Pc = (\Sigma_i o_i(\vec{x_i})); P_i c$. Entonces aplicando hipótesis inductiva sabemos que el tipo de P_i es el mismo que el de $P_i c$ por lo tanto el tipo de P es el mismo al de Pc.
- Los otros casos son inmediatos usando hipótesis inductiva.

□

El siguiente resultado establece la Propiedad de *subject reduction* respecto de la relación de reducción $\overset{\alpha,c}{\longmapsto}$ y se utilizará en la demostración de la propiedad de *subject reduction* respecto de $\overset{\alpha,c}{\longrightarrow}$.

Proposición 5.3.6. *Sea P un proceso bien tipado. Si $\Gamma \vdash P : \mathcal{T}$ y $P \overset{\alpha,c'}{\longmapsto} P'$, entonces existe $\mathcal{T}'$ tal que:*

- *$\Gamma \vdash P' : \mathcal{T}'$, es decir, el término es bien tipado.*
- *$\forall o. \mathcal{T}'(o) \subseteq \mathcal{T}(o)$.*

Demostración. La prueba es por inducción en la derivación $P \overset{\alpha,c'}{\longmapsto} P'$. Procedemos analizando la última regla aplicada

- (IN). Luego, $P = \Sigma_i o_i(\vec{x_i}); P_i$. Como P es bien tipado, todo P_i es bien tipado. Por lo tanto, para todo i existe $\mathcal{T}_{P_i}$ tal que $\Gamma \vdash P_i : \mathcal{T}_{P_i}$. Luego, $\mathcal{T} = \bigoplus_i \mathcal{T}_i = \bigcup_i \{o_i \mapsto \vec{x_i}\} \cup \mathcal{T}_{P_i}$. Además, α tiene que ser algún $o_i(\vec{v_i})$. Por lo tanto, $P' = P_i$, que por hipótesis es un término bien tipado. Esto significa que $\mathcal{T}' = \mathcal{T}_{P_i}$ para algún i. Es fácil ver que para toda operación o vale que $\mathcal{T}'(o) = \mathcal{T}_{P_i}(o) \subseteq \bigcup_i \{o_i \mapsto \vec{x_i}\} \cup \mathcal{T}_{P_i}(o) \subseteq \bigoplus_i \mathcal{T}_i(o)$.
- (OUT). Inmediato, ya que $P' = 0$ tiene tipo $\emptyset$.
- (PAR). $P = Q_1|Q_2$, con $\Gamma \vdash Q_1 : \mathcal{T}_1$, $\Gamma \vdash Q_2 : \mathcal{T}_2$, $\mathcal{T}_1 \between \mathcal{T}_2$ y $\mathcal{T} = \mathcal{T}_1 \oplus \mathcal{T}_2$. Luego, $P' = Q_1'|Q_2$ con $Q_1 \overset{\alpha,c'}{\longmapsto} Q_1'$. Entonces por hipótesis inductiva, sabemos que existe $\mathcal{T}_1'$ tal que $\Gamma \vdash Q_1' : \mathcal{T}_1'$ y $\mathcal{T}_1'(o) \subseteq \mathcal{T}_1(o)$ para toda o. Por Proposición 5.3.4, para las tres definiciones del operador de compatibilidad($\between_c$, $\between_a$ y $\between_e$) se cumple que: $\mathcal{T}_1 \between \mathcal{T}_2$ y $\mathcal{T}_1'(o) \subseteq \mathcal{T}_1(o)$ para toda o implica $\mathcal{T}_1' \between \mathcal{T}_2$, es decir $\Gamma \vdash Q_1'|Q_2 : \mathcal{T}_1' \oplus \mathcal{T}_2$. De esta manera, $\mathcal{T}_1'(o) \subseteq \mathcal{T}_1(o)$ implica que $\mathcal{T}_1'(o) \cup \mathcal{T}_2(o) \subseteq \mathcal{T}_1(o) \cup \mathcal{T}_2(o)$, por lo tanto $\mathcal{T}'(o) \subseteq \mathcal{T}(o)$.
- (SEQ). $P = Q_1; Q_2$, $\Gamma \vdash Q_1 : \mathcal{T}_1$, $\Gamma \vdash Q_2 : \mathcal{T}_2$ y $\mathcal{T} = \mathcal{T}_1 \oplus \mathcal{T}_2$. Luego, $P' = Q_1'; Q_2(c')$ con $Q_1 \overset{\alpha,c'}{\longmapsto} Q_1'$. Entonces por hipótesis inductiva, sabemos que existe $\mathcal{T}_1'$ tal que $\Gamma \vdash Q_1' : \mathcal{T}_1'$ y $\mathcal{T}_1'(o) \subseteq \mathcal{T}_1(o)$ para toda o. Por Proposición 5.3.5 sabemos que el tipo de Q_2 luego de la sustitución es el mismo, además, $\mathcal{T}_1'(o) \subseteq \mathcal{T}_1(o)$ implica que $\mathcal{T}_1'(o) \oplus \mathcal{T}_2(o) \subseteq \mathcal{T}_1(o) \oplus \mathcal{T}_2(o)$, por lo tanto $\mathcal{T}'(o) \subseteq \mathcal{T}(o)$.
- (REC). $P = \mathbf{rec}_X\ Q$, $\Gamma \vdash Q : \mathcal{T}$ y $\Gamma(X) = \mathcal{T}$. Por Proposición 5.3.5, $\Gamma \vdash Q[^{\mathbf{rec}_X\ P}/_X] : \mathcal{T}$. Luego, por hipótesis inductiva sobre $Q[^{\mathbf{rec}_X\ Q}/_X] \overset{\alpha,c'}{\longmapsto} P'$ concluímos que existe $\mathcal{T}'$ tal que $\Gamma \vdash P' : \mathcal{T}'$ y $\mathcal{T}'(o) \subseteq \mathcal{T}(o)$ para todo o.

- (IF). $P = \textbf{if } v = v' \textbf{ then } Q_1 \textbf{ else } Q_2$, $\Gamma \vdash Q_1 : \mathcal{T}_1$, $\Gamma \vdash Q_2 : \mathcal{T}_2$ y $\mathcal{T} = \mathcal{T}_1 \oplus \mathcal{T}_2$. Además, $P = Q_1'$ y $Q_1 \xmapsto{\alpha, c'} Q_1'$. Por hipótesis inductiva, sabemos que existe $\mathcal{T}_1'$ tal que $\Gamma \vdash Q_1' : \mathcal{T}_1'$ y $\mathcal{T}_1'(o) \subseteq \mathcal{T}_1(o)$ para toda o. De esta manera, $\mathcal{T}_1'(o) \subseteq \mathcal{T}_1(o)$ implica que $\mathcal{T}_1'(o) \oplus \mathcal{T}_2(o) \subseteq \mathcal{T}_1(o) \oplus \mathcal{T}_2(o)$, por lo tanto $\mathcal{T}'(o) \subseteq \mathcal{T}(o)$.
- (THEN). Demostración análoga al caso anterior

□

El siguiente resultado establece que el tipo de un proceso captura todos los posibles parámetros formales de todas las operaciones de entrada que podría ejecutar un proceso.

Lema 5.3.1. *Sea P un proceso. Si $\Gamma \vdash P : \mathcal{T}$ y $c \triangleright [P] \Rightarrow^* c_n \triangleright [P_n] \xrightarrow{o(v_1,\ldots,v_n),c'} c[c'] \triangleright [P']$, entonces una de las siguientes condiciones vale:*

1. *$\exists\, \langle y_1, \ldots, y_n \rangle \in \mathcal{T}(o)$ tal que $dom(c') = \langle y_1, \ldots, y_n \rangle$.*
2. *$P' = \dagger$.*

Demostración. La demostración se realiza haciendo inducción en la cantidad de derivaciones $\Rightarrow^*$.

- **n=0**. Este caso es inmediato por la proposición 5.3.3.
- **n=k+1**. $c \triangleright [P] \xrightarrow{\alpha, c_0} c'' \triangleright [P''] \Rightarrow^k c_{k+1} \triangleright [P_{k+1}] \xrightarrow{o(v_1,\ldots,v_n),c_{k+1}} c' \triangleright [P']$. Analicemos la estructura de P.
  - $P = \Sigma_i o_i(\vec{x_i}); P_i$. Entonces, $c \triangleright [P] \xrightarrow{o_i(\vec{v}), \vec{x_i} \mapsto \vec{v}} c[\vec{x_i} \mapsto \vec{v}] \triangleright [P_i']$. Por la Proposición 5.3.3 sabemos que existe $\mathcal{T}_1$ tal que $\Gamma \vdash P_i' : \mathcal{T}_1$ y $\forall o. \mathcal{T}_1(o) \subseteq \mathcal{T}(o)$ ó $P_i = \dagger$, pero este segundo caso no vale porque luego no se podría hacer otro paso de reducción. Por hipótesis inductiva (aplicada sobre $c'' \triangleright [P''] \Rightarrow^k c_{k+1} \triangleright [P_{k+1}] \xrightarrow{o(v_1,\ldots,v_n),c_{k+1}} c' \triangleright [P']$) sabemos que existe $\langle y_1, \ldots, y_m \rangle \in \mathcal{T}_1(o)$ tal que $dom(c') = \langle y_1, \ldots, y_n \rangle$ ó $P_i' = \dagger$. Dado que $\forall o. \mathcal{T}_1(o) \subseteq \mathcal{T}(o)$ entonces concluímos que $\langle y_1, \ldots, y_m \rangle \in \mathcal{T}_1(o)$.
  - La prueba para el resto de los casos es análoga.

□

Lema 5.3.2 (Subject congruence)**.** *Sea P y Q procesos bien tipados. Si $\Gamma \vdash P : \mathcal{T}$ y $P \equiv Q$ entonces $\Gamma \vdash Q : \mathcal{T}$.*

Demostración. La prueba se realiza por inducción en la derivación de $P \equiv Q$ mostrando que:

1. $\Gamma \vdash P : \mathcal{T}$ implica $\Gamma \vdash Q : \mathcal{T}$.

2. $\Gamma \vdash Q : \mathcal{T}$ implica $\Gamma \vdash P : \mathcal{T}$.

- (IN). Luego, $P = \Sigma_i o_i(\vec{x_i}); P_i$, $Q = \Sigma_i o_i(\vec{x_i}); Q_i$ y $P_i \equiv Q_i$. Como P y Q son bien tipados, todo P_i y Q_i son bien tipado. Por lo tanto, para todo i existe $\mathcal{T}_{P_i}$ tal que $\Gamma \vdash P_i : \mathcal{T}_{P_i}$. Por (1) asumiremos que $\Gamma \vdash P : \mathcal{T}_P$, luego, $\mathcal{T}_P = \bigoplus_i \mathcal{T}_i = \bigcup_i \{o_i \mapsto \vec{x_i}\} \cup \mathcal{T}_{P_i}$. Por hipótesis inductiva $\Gamma \vdash Q_i : \mathcal{T}_{P_i}$. Esto significa que $\mathcal{T}_Q = \bigoplus_i \mathcal{T}_i = \bigcup_i \{o_i \mapsto \vec{x_i}\} \cup \mathcal{T}_{P_i}$. El caso para (2) es simétrico.
- (PAR). Luego, $P = P_1|R$, $Q = Q_1|R$ y $P_1 \equiv Q_1$. Como P y Q son bien tipados, entonces $\Gamma \vdash P : \mathcal{T}_P$ y $\Gamma \vdash Q : \mathcal{T}_Q$. Por (1) asumiremos que $\Gamma \vdash P_1 : \mathcal{T}_1$, $\Gamma \vdash R : \mathcal{T}_2$, $\mathcal{T}_1 \between \mathcal{T}_2$ y $\mathcal{T}_P = \mathcal{T}_1 \oplus \mathcal{T}_2$. Por hipótesis inductiva, $\Gamma \vdash Q_1 : \mathcal{T}_1$. Luego, $\Gamma \vdash Q_1 : \mathcal{T}_1$ por lo que $\mathcal{T}_Q = \mathcal{T}_1 \oplus \mathcal{T}_2$. El caso para (2) es simétrico. La operacion sobre los tipos es unión de conjuntos que es asociativa y comutativa, además para el 0, el tipo es vacío que es la identidad para la unión.
- La prueba para el resto es análoga.

□

Los siguientes tres resultados establecen la propiedad de subject reduction respecto de $\xrightarrow{\alpha,c'}$ cuando se consideran respectivamente las nociones de compatibilidad de tipos $\between_e$, $\between_a$ y $\between_c$.

Lema 5.3.3 (Subject reduction para $\between_e$)**.** *Sea P un proceso. Si $\Gamma \vdash P : \mathcal{T}$ (usando la noción de compatibilidad $\between_e$) y $P \xrightarrow{\alpha,c'} P'$, entonces existe $\mathcal{T}'$ tal que:*

- $\Gamma \vdash P' : \mathcal{T}'$*, es decir, el término es bien tipado.*
- $\forall o. \mathcal{T}'(o) \subseteq \mathcal{T}(o)$.

Demostración. La prueba se realiza analizando la regla aplicada para derivar $P \xrightarrow{\alpha,c'} P'$.

- (NO-EXCP). Inmediato por Proposición 5.3.6.
- (AMB-REC-EXCP). Mostraremos que esta regla no puede ser utilizada. $P = P_1|P_2$. Como P es bien tipado, se debe cumplir que $\Gamma \vdash P_1 : \mathcal{T}_1$, $\Gamma \vdash P_1 : \mathcal{T}_2$ tal que $\mathcal{T}_1 \between_e \mathcal{T}_2$. Además, $P_1 \xmapsto{o(\vec{v}),c_1} P_1'$ y $P_2 \xmapsto{o(\vec{v}),c_2} P_2'$. Por Proposición 5.3.1 podemos concluir que $\mathcal{T}_1(o) \neq \emptyset$ y $\mathcal{T}_2(o) \neq \emptyset$, lo que implica que P no es bien tipado.
- (CONF-REC-EXCP). Mostraremos que esta regla no puede ser utilizada. Dado que $P \xmapsto{\alpha,c} P'$, podemos concluir usando Proposición 5.3.6 que P' es bien tipado. Razonando en modo análogo al caso previo, podemos concluir que $P' \equiv P_1|P_2$ no puede habilitar concurrentemente dos acciones de input sobre la misma operación y por lo tanto la aplicación de esta regla implica que P no es bien tipado.

□

Lema 5.3.4 (Subject reduction para $\owedge_c$)**.** *Sea P un proceso. Si $\Gamma \vdash P : \mathcal{T}$ (usando la noción de compatibilidad $\owedge_c$) y $P \xrightarrow{\alpha,c'} P'$, entonces $P' = \dagger$ o bien existe $\mathcal{T}'$ tal que:*

- *$\Gamma \vdash P' : \mathcal{T}'$, es decir, el término es bien tipado.*
- *$\forall o.\mathcal{T}'(o) \subseteq \mathcal{T}(o)$.*

Demostración. La prueba se realiza analizando la regla aplicada para derivar $P \xrightarrow{\alpha,c'} P'$.

- (NO-EXCP). Inmediato por Proposición 5.3.6.
- (AMB-REC-EXCP). Inmediato, ya que $P' = \dagger$.
- (CONF-REC-EXCP). Mostraremos que esta regla no puede ser utilizada. Dado que $P \xmapsto{\alpha,c} P'$, podemos concluir usando Proposición 5.3.6 que P' es bien tipado. Luego, $P' \equiv P_1|P_2$. Como P' es bien tipado, se debe cumplir Subject Congruence(5.3.2) y que $\Gamma \vdash P_1 : \mathcal{T}_1$, $\Gamma \vdash P_2 : \mathcal{T}_2$ tal que $\mathcal{T}_1 \owedge_c \mathcal{T}_2$. Además, $P_1 \xmapsto{o(\vec{v}),c_1} P_1'$ y $P_2 \xmapsto{o(\vec{v}),c_1} P_2'$. Por Proposición 5.3.1 podemos concluir que $\mathcal{T}_1(o) \neq \emptyset$ y $\mathcal{T}_2(o) \neq \emptyset$ y, en consecuencia $\mathcal{T}_1(o) \cup \mathcal{T}_2(o) \neq \emptyset$. Por lo tanto P' no es bien tipado, lo que contradice el hecho que P es bien tipado(Proposición 5.3.6).

□

Lema 5.3.5 (Subject reduction para $\owedge_a$)**.** *Sea P un proceso. Si $\Gamma \vdash P : \mathcal{T}$ (usando la noción de compatibilidad $\owedge_a$) y $P \xrightarrow{\alpha,c'} P'$, entonces $P' = \ddagger$ o bien existe $\mathcal{T}'$ tal que:*

- *$\Gamma \vdash P' : \mathcal{T}'$, es decir, el término es bien tipado.*
- *$\forall o.\mathcal{T}'(o) \subseteq \mathcal{T}(o)$.*

Demostración. La prueba se realiza analizando la regla aplicada para derivar $P \xrightarrow{\alpha,c'} P'$.

- (NO-EXCP). Inmediato por Proposición 5.3.6.
- (AMB-REC-EXCP). Mostraremos que esta regla no puede ser utilizada. $P = P_1|P_2$. Como P es bien tipado, se debe cumplir que $\Gamma \vdash P_1 : \mathcal{T}_1$, $\Gamma \vdash P_1 : \mathcal{T}_2$ tal que $\mathcal{T}_1 \owedge_a \mathcal{T}_2$. Además, $P_1 \xmapsto{o(\vec{v}),c_1} P_1'$ y $P_2 \xmapsto{o(\vec{v}),c_2} P_2'$ y $c_1 \neq c_2$. Proposición 5.3.1 podemos concluir que $\mathcal{T}_1(o) \neq \emptyset$ y $\mathcal{T}_2(o) \neq \emptyset$. Notar que $c_1 \neq c_2$ implica $dom(c_1) \neq dom(c_2)$ (necesariamente la imagen de la correlación debe coincidir porque las dos correlaciones deben corresponderse con la misma acción de entrada), por lo tanto $\#(\mathcal{T}_1(o) \cup \mathcal{T}_2(o)) > 1$ lo que implica que P no es bien tipado.

- (CONF-REC-EXCP). Inmediato, ya que $P' = \ddagger$.

□

El siguiente teorema es el principal resultado de nuestro trabajo y dice que un servicio bien tipado es libre de excepciones.

Teorema 5.3.1. *Sea P un proceso tal que $\Gamma \vdash P : \mathcal{T}$, entonces un servicio bien formado $s_C^O\{P, 0, \emptyset\}$ es*

1. *libre de excepciones ambiguous-receive si el tipo define como operador de compatibilidad a $\oslash_a$.*
2. *libre de excepciones conflict-receive si el tipo define como operador de compatibilidad a $\oslash_c$.*
3. *libre de excepciones de correlación si el tipo define como operador de compatibilidad a $\oslash_e$.*

Demostración. 1. Probamos por inducción en la cantidad de reducciones $\Rightarrow^n$ que $s_C^O\{P, 0, \emptyset\} \Rightarrow^n s_C^O\{P, I, M\}$ implica (i) $I \neq c \triangleright [\dagger]||I'$ y (ii) $I \equiv c \triangleright [Q]||I'$ implica Q es bien tipado o $Q = \ddagger$.

- **n=0**. Es inmediato ya que $I \equiv 0$.
- **n=k+1**. Entonces $s_C^O\{P, 0, \emptyset\} \Rightarrow^k s_C^O\{P, I_k, M_k\} \Rightarrow s_C^O\{P, I, M\}$. Por hipótesis inductiva sabemos que $I_k \not\equiv c \triangleright [\dagger]||I'_k$. Por lo tanto, queda por demostrar que la excepción no es arrojada en el último paso. Para eso, analicemos el caso de la última reducción $s_C^O\{P, I_k, M_k\} \Rightarrow s_C^O\{P, I, M\}$.
  - $s_C^O\{P, I_k, M_k\} \xrightarrow{\tau} s_C^O\{P, I, M\}$. Hay tres posibilidades:
    - **regla (dispatch)**. $I' \equiv c \triangleright [Q]||I_2 \xrightarrow{o_1(v_1),c'} I = c[c'] \triangleright [Q']||I_2$ con $Q \xrightarrow{o_1(v_1),c'} Q'$. Lo probaremos por absurdo. Supongamos $Q' = \dagger$. Luego, $Q \xrightarrow{o_1(v_1),c'} \dagger$ lo que contradice Lemma 5.3.4 ya que $\dagger$ no es bien tipado. Por lo tanto podemos concluir que $Q' \neq \dagger$ y Q' es bien tipado.
    - **rule (new)**. $I \equiv c \triangleright [Q]||I_2$ con $C_\perp \triangleright [P] \xrightarrow{o_1(v_1),c'} Q$. Como en el caso anterior, la prueba es por absurdo, suponiendo que $Q = C_\perp \triangleright [\dagger]$ lo que contradice Lemma 5.3.4. Por lo tanto podemos concluir que $Q \neq C_\perp \triangleright [\dagger]$ y Q es bien tipado.
    - **rule (svc-non-in)**. Es inmediato puesto que las acciones τ no arrojan $\dagger$.
  - $s_C^O\{P, I_k, M_k\} \xrightarrow{\overline{o}\langle v\rangle} s_C^O\{P, I, M\}$ ó $s_C^O\{P, I_k, M_k\} \xrightarrow{o(v)} s_C^O\{P, I, M\}$. Estos casos no realizan reducciones que arrojan excepciones.

Casos 2. y 3. son análogos.

□

En el capítulo anterior presentamos a modo de ejemplo diferentes sistemas descriptos en CAB (Sección 4.4). En particular, los ejemplos 4 y 5 describieron escenarios donde los sistemas alcanzaban excepciones. A continuación analizaremos estos ejemplos mostrando que las excepciones son alcanzadas dado que no tipan.

El ejemplo 4 presenta el siguiente sistema:

$$N = s^{\{o_1,o_2\}}_{\{x,y\}}\{o_1(x,y); (o_2(x)|o_2(y)), 0, \emptyset\}$$

Para inferir el tipo de N aplicaremos las reglas de tipado. La regla de precedencia establece que la composición secuencial tiene mayor precedencia que la composición en paralelo, por lo tanto apliquemos la regla SEQ. El juicio de tipado para el término de más a la izquierda se define como: $\Gamma \vdash o_1(x,y) : \mathcal{T}_1$, que según la regla de INPUT, $\mathcal{T}_1 = \{o_1 \mapsto \langle x,y\rangle\}$. Para inferir el tipo del término de la derecha de la composición secuencial aplicaremos la regla PAR. Esta regla establece que el tipo $(\mathcal{T}_2)$, es la composición de los tipos asignados a cada uno de los subtérminos. Entonces, el juicio de tipado para el término de la izquierda de la composición paralela es: $\Gamma \vdash o_2(x) : \{o_2 \mapsto \langle x\rangle\}$ y para el término de más a la derecha es: $\Gamma \vdash o_2(x) : \{o_2 \mapsto \langle y\rangle\}$. Por último, evaluémos el operador de compatibilidad entre estos términos. La operación en común entre ambos términos es o_2, que en particular tiene parámetros distintos, por lo tanto violamos la condición de compatibilidad $\oslash_a$. Violar la condición de compatibilidad $\oslash_a$ nos alcanza para mostrar que N no tipa.

El ejemplo 5 presenta el siguiente sistema:

$$N = s^{\{o_1,o_2\}}_{\{x\}}\{o_1(x); (o_2(x)|o_2(x)), 0, \emptyset\}$$

De la misma manera, para inferir el tipo de N aplicaremos las reglas de tipado. La primer regla a aplicar es SEQ. El juicio de tipado para el término de más a la izquierda se define como: $\Gamma \vdash o_1(x) : \mathcal{T}_1$, que según la regla de INPUT, $\mathcal{T}_1 = \{o_1 \mapsto \langle x\rangle\}$. Para inferir el tipo del término de la derecha de la composición secuencial aplicaremos la regla PAR. Entonces, el juicio de tipado para el término de la izquierda de la composición paralela es: $\Gamma \vdash o_2(x) : \{o_2 \mapsto \langle x\rangle\}$ y para el término de más a la derecha es: $\Gamma \vdash o_2(x) : \{o_2 \mapsto \langle x\rangle\}$. Finalmente evaluémos el operador de compatibilidad entre estos términos. La operación en común entre ambos términos es o_2, y en particular, tienen los mismos parámetros de entrada. Por lo tanto, violamos la condición de compatibilidad $\oslash_c$ que nos alcanza para mostrar que N no tipa.

Capítulo 6

Conclusiones y Trabajo futuro

Este trabajo presenta un cálculo de procesos, llamado Cab, que describe el uso de primitivas de correlación. La definición formal del mecanismo de correlación en Cab se inspira fuertemente en el cálculo de procesos Sock [7]. Sin embargo, elegimos no incluir en nuestro modelo aspectos tales como operaciones de tipo two-way o manipulación de estado para obtener un lenguaje minimal que describa de forma simple el uso de correlaciones y excepciones de correlación. De todas maneras, el modelo que presentamos podría extenderse a un cálculo que contenga características como las propuestas en Sock, como por ejemplo, manejo de compensaciones o links, ya que forman parte de la especificación de BPEL. Explorar este tipo de extensiones se deja como trabajos futuros de investigación.

Como trabajos futuros, el cálculo puede extenderse de forma tal que incluya variables de correlación frescas. El uso de variables frescas, permitirá garantizar que las instancias se creen de forma unívoca, evitando así que existan instancias con el mismo valor de correlación.

A continuación mencionamos las principales diferencias de Cab con respecto a otras propuestas en la literatura. Hacemos notar que Sock no permite generar intancias que colisionen, mientras que Cab no impone ningún tipo de restricción al respecto. Además, Cab tiene un mecanismo que arroja excepciones automáticamente cuando la instancia activa concurrentemente dos o más acciones de entrada que pueden manejar una misma solicitud. Aunque algunas extensiones de Sock (tal como la propuesta en [6]) ofrecen primitivas para manejar excepciones, las excepciones son lanzadas por la ejecución de una primitiva específica y no como consecuencia de violar alguna restricción en el uso de las correlaciones. Blite [10] es otro cálculo de procesos que formaliza un gran número de primitivas de Bpel. Como consecuencia, contiene varias primitivas que decidimos no incluir en nuestro cálculo. En lo referido al mecanismo de correlación, hacemos notar que los servicios en Blite entregan los mensajes usando "el principio de la instancia más específica", es decir, si varias instancias pueden responder a un mensaje de entrada, entonces el mensaje es dirigido a la instancia que para procesar el mensaje recibido necesita inicializar la menor cantidad de variables. Sí hay más de una instancia, entonces Blite elige

de manera no determinística alguna de ellas. Este comportamiento contrasta con CAB, donde el mecanismo de correlación no realiza ningún tipo de control entre distintas instancias y la elección siempre es realizada de manera no determinística. A diferencia de BLITE y SOCK, CAB no modela explícitamente a los socios de una comunicación. Tomamos esta decisión de diseño porque los mecanismos de correlación son usados generalmente en combinación con identificadores de puntos de acceso ó endpoints (son los puntos donde se dirigen las solicitudes) estáticos. Esto hace que los socios de un servicio no puedan cambiar dinámicamente y, en consecuencia, no vemos razones para modelar los socios explícitamente en el modelo. Con respecto a COWS [10] mencionamos que este modelo se basa en un mecanismo de pattern matching para modelar conjuntos de correlación. En ese sentido, COWS describe a los conjuntos de correlación en un nivel más bajo de abstracción. Asimismo, se basa en "el principio de la instancia más específica" para resolver posibles conflictos.

Como aspecto novedoso de CAB con respecto a modelos propuestos previamente en la literatura remarcamos que CAB describe la interacción entre el mecanismo de correlación y de excepciones que se lanzan cuando se violan ciertas restricciones en el uso de correlaciones. En particular, nuestro modelo considera las excepciones ambigous-receive y conflicting-receive de la especificación BPEL.

Nuestro trabajo además caracteriza servicios que están definidos de manera tal que garantizan que sus instancias no arrojarán excepciones de correlación durante la ejecución. Tales servicios están caracterizados estáticamente mediante un sistema de tipos que es paramétrico respecto a una noción de compatibilidad de tipos asociados a flujos concurrentes. En particular, hemos propuesto tres nociones de compatibilidad y mostrado que cada una de ellas se caracteriza la ausencia de distinto tipo de excepciones.

Bibliografía

[1] World Wide Web Consortium Web Services Architecture. Available at `http://www.w3.org/TR/ws-arch/wsa.pdf`, September 2011.

[2] Eduardo Bonelli and Adriana Compagnoni. Multisession session types for a distributed calculus. In *Proc. of Trustworthy Global Computing 2007*, volume 4912 of *Lecture Notes in Computer Science*, pages 240–256. Springer, 2007.

[3] M. Boreale, R. Bruni, L. Caires, R. De Nicola, I. Lanese, M. Loreti, F. Martins, U. Montanari, A. Ravara, D. Sangiorgi, V. Vasconcelos, and G. Zavattaro. SCC: a service centered calculus. In *Proc. of WS-FM 2006*, volume 4184 of *Lect. Notes in Comput. Sci.*, pages 38–57. Springer, 2006.

[4] Web services addressing (ws-addressing). Available at `http://www.w3.org/Submission/ws-addressing/`, August 2004.

[5] Web services Business Process Execution Language (BPEL). version 2.0. Available at `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf`, Abril 2007.

[6] C. Guidi, I. Lanese, F. Montesi, and G. Zavattaro. On the interplay between fault handling and request-response service invocations. In *Proceedings of 8th International Conference on Application of Concurrency to System Design (ACSD 2008)*, pages 190–198. IEEE, 2008.

[7] C. Guidi, R. Lucchi, R. Gorrieri, N. Busi, and G. Zavattaro. SOCK: A calculus for service oriented computing. In *Service-Oriented Computing - ICSOC 2006*, volume 4294 of *Lecture Notes in Computer Science*, pages 327–338. Springer, 2006.

[8] K. Honda, V. Vasconcelos, and M. Kubo. Language primitives and type disciplines for structured communication-based programming. In *Proc. of ESOP'98*, volume 1381 of *Lect. Notes in Comput. Sci.*, pages 22–138. Springer, 1998.

[9] I. Lanese, V.T. Vasconcelos, F. Martins, and A. Ravara. Disciplining orchestration and conversation in service-oriented computing. In *Proc. of SEFM'07*, pages 305–314. IEEE Computer Society Press, 2007.

[10] A. Lapadula, R. Pugliese, and F. Tiezzi. A calculus for orchestration of web services. In *Proc. of ESOP'07*, volume 4421 of *Lect. Notes in Comput. Sci.*, pages 33–47. Springer, 2007.

[11] A. Lapadula, R. Pugliese, and F. Tiezzi. A formal account of ws-bpel. In *COORDINATION*, volume 5052 of *Lecture Notes in Computer Science*, pages 199–215. Springer, 2008.

[12] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lect. Notes in Comput. Sci.* Springer, 1980.

[13] G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, Computer Science Department, 1981.

[14] M. Viroli. A core calculus for correlation in orchestration languages. *J. Log. Algebr. Program.*, 70(1):74–95, 2007.

[15] ActiveBPEL 4.1. `http://www.active-endpoints.com`, April 2010.

[16] Apache ODE 1.1.1 `http://ode.apache.org/`, April 2010.

[17] Oracle BPEL Process Manager 10.1.3 `http://www.oracle.com/technology/bpel`, April 2010.

[18] Web Services Business Process Execution Language Technical Committee. WS-BPEL issues list, `http://http://www.oasis-open.org/committees/download.php/20228/WS_BPEL_issues_list.html`, Marzo 2010.

[19] Franck van Breugel and Maria Koshkina. Models and Verification of BPEL

[20] C. Stahl. Transformation von BPEL4WS in Petrinetze. Master's thesis Humboldt-Universitat zu Berlin, Berlin, Germany, April 2004.

[21] H.M.W. Verbeek and W.M.P. van der Aalst. Analyzing BPEL processes using Petri nets. In D. Marinescu, editor, Proceedings of the Second International Workshop on Applications of Petri Nets to Coordination, Work ow and Business Process Management, pages 59-78, Miami, FL, USA, October 2005.

[22] Y. Yang, Q. Tan, and Y. Xiao. Verifying web services composition based on hierarchical colored Petri nets. In Proceedings of the 1st International Workshop on Interoperability of Heterogeneous Information Systems, pages 47-54, Bremen, Germany, November 2005. ACM.

[23] X. Fu, T. Bultan, and J. Su. Model checking XML manipulating software. In G.S. Avrunin and G. Rothermel, editors, Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis, pages 252-262, Boston, MA, USA, July 2004. ACM.

[24] S. Nakajima. On verifying web service. In Proceedings of the Symposium on Applications and the Internet Workshops, pages 223-224, Nara City, Japan, January/February 2002. IEEE.

[25] X. Fu, T. Bultan, and J. Su. Model checking XML manipulating software. In G.S. Avrunin and G. Rothermel, editors, Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis, pages 252-262, Boston, MA, USA, July 2004. ACM.

[26] R. Faranhbod, U. Glässer, and M. Vajihollahi. Abstract operational semantics of the business process execution language for web services. Technical Report SFUCMPT-TR-2004-03, Simon Frasier University, Burnaby, Canada, April 2004.

[27] R. Faranhbod, U. Glässer, and M. Vajihollahi. Specifcation and validation of the business process execution language for web services. In W. Zimmermann and B. Thalheim, editors, Proceedings of the 11th International Workshop on Abstract State Machines, volume 3052 of Lecture Notes in Computer Science, pages 78-94, Lutherstadt Wittenberg, Germany, May 2004. Springer-Verlag.

[28] R. Faranhbod, U. Glässer, and M. Vajihollahi. An abstract machine architecture for web service based business process management. In C. Bussler and A. Haller,editors, Proceedings of the Business Process Management Workshops, volume 3812 of Lecture Notes in Computer Science, pages 144-157, Nancy, France, September 2005. Springer-Verlag.

[29] R. Faranhbod, U. Glässer, and M. Vajihollahi. A formal semantics for the business process execution language for web services. In S. Bevinakoppa, L.F. Pires, and S. Hammoudi, editors, Proceedings of the Joint Workshop on Web Services and Model-Driven Enterprise Information Services, pages 122-133, Miami, FL, USA,May 2005. INSTICC Press.

[30] A. Wombacher, P. Fankhauser, and E. Neuhold. Transforming BPEL into annotated deterministic finite state automata for service discovery. In Proceedings of the 2nd IEEE International Conference on Web Services, pages 316-323, San Diego, CA, USA, July 2004. IEEE.

[31] S. Haddad, T. Melliti, P. Moreaux, and S. Rampacek. Modelling web services interoperability. In Proceedings of the 6th International Conference on Enterprise Information Systems, volume 4, pages 287-295, Porto, Portugal, April 2004.

[32] R. Kazhamiakin and M. Pistore. Parametric communication model for the verification of BPEL4WS compositions. In M. Bravetti, L. Kloul, and G. Zavattaro, editors, Proceedings of the 2nd International Workshop on Web Services and

Formal Methods, volume 3670 of Lecture Notes in Computer Science, pages 318-332, Versailles, France, September 2005. Springer-Verlag.