Anexo A

Ejemplos de representación formal de servicios convergentes

En este anexo son presentados algunos ejemplos de servicios convergentes, los cuales fueron modelados mediante las representaciones formales seleccionadas en el capítulo 3 (grafos y CPN). Este modelado fue realizado con el fin de validar de manera práctica la selección realizada.

A continuación son presentados algunos ejemplos de servicios convergentes modelados en CPN.

A.1. Servicio: cajero automático

En la figura A.1 es presentado el ejemplo de un cajero automático, en este ejemplo la CPN facilita la especificación de la información para realizar la transacción. En la figura A.1 los datos de la cuenta, cantidad de dinero y número de contraseña, deben almacenarse temporalmente para ser procesados. Para incluir esta información es necesario definir un tipo particular de dato, representado gráficamente por un color dentro del token.

La Figura A.1 contiene en el lugar A dos token marcados con dos colores diferentes. En la práctica, cada color podría identificar diferentes cuentas del usuario de donde se podría realizar el retiro del dinero. En el primer estado, a cada token se le añadiría la información de cantidad de dinero y contraseña, posteriormente, la información será evaluada para ejecutar la lógica del lugar B, *debitando*.

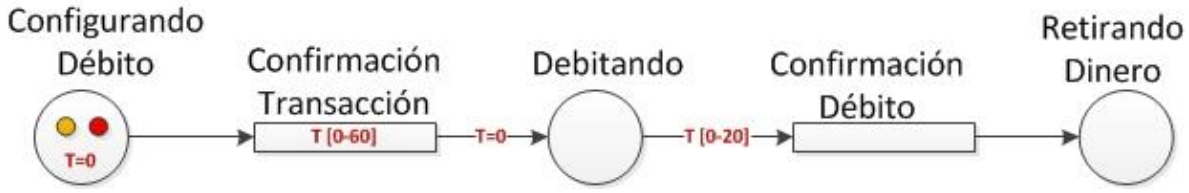

Figura A.1: CPN débito cajero automático

A.2. Servicio: Facebook Event Reminder

El servicio convergente *FacebookEventReminder* permite ejemplificar un servicio que combina la red social de Facebook (servicio Web) con la mensajería de texto SMS (servicio Telco). En este sentido, se presenta el funcionamiento general del servicio y su modelo en CPN.

Lo expuesto a continuación describe la estructura total del servicio, en consecuencia, primero se explicará el modelo de grafos el cual representa la síntesis del servicio, posteriormente se describirá la CPN que representa la orquestación.

El servicio FacebookEventRiminder consiste en el envío notificaciones de recordatorio a los asistentes de un evento publicado por un usuario en Facebook. El funcionamiento del servicio es el siguiente: Un usuario A ha inscrito el servicio de recordatorio de eventos. Para ello, el usuario del servicio ha publicado previamente un evento en facebook, el cual contiene la información detallada del lugar, hora y nombre. Para este evento existe un conjunto de personas que asistirán, a las cuales se les notificará una vez el usuario A consuma el servicio.

La figura A.2 describe el proceso abstracto en modelo de grafos del servicio *FacebookEvent Reminder*. En él, los nodos representan servicios web (en verde) o servicios Telco (en rojo) y la interacción entre los servicios se hace a través de conectores o patrones bien definidos. Inicialmente se obtiene la información del evento y luego la lista de invitados (servicios web); posteriormente se obtiene la información de cada invitado necesaria para enviarle mensajes vía SMS y

mensajería instantánea (servicios Telco). El envío de mensajes se realiza en un ciclo repetitivo para cada invitado.

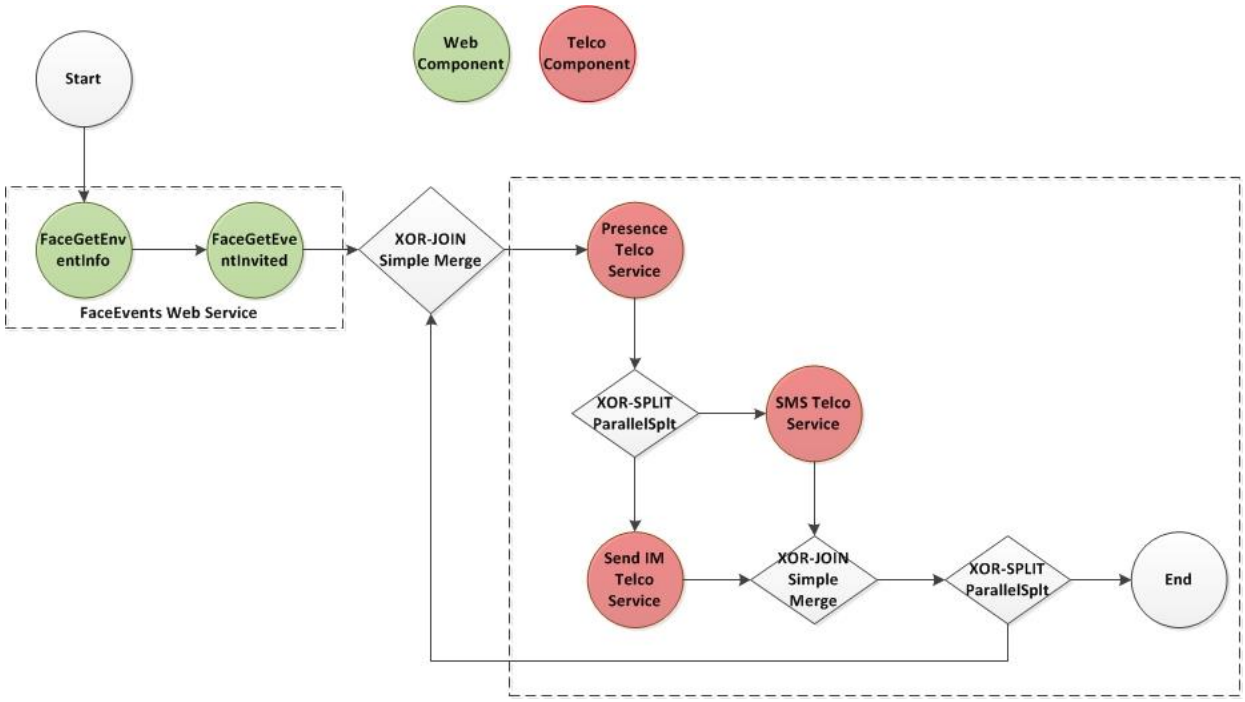

Figura A.2: síntesis del servicio FacebookEventReminder

Por otra parte la figura A.3 presenta la orquestación del servicio descrito, para realizar este modelado fue utilizada la herramienta gráfica *CPN tolos debido a que ofrece* soporte para la construcción, simulación, análisis de espacio de estados y análisis de desempeño para modelos CPN [2], además, tiene un gran soporte técnico y es usada en más 150 países con más de 10.000 licencias otorgadas (académica, comercial o investigación) [5] [18].

En la figura A.3 inicialmente el servicio se encuentra en el lugar de espera por algún evento inicial (*Wating Starting*). Un vez el usuario A consume el servicio, la solicitud de inicio, idfaceR, es procesada por el lugar *Analysing Initial Event*. Posteriormente, el servicio internamente invoca, a través del evento *call FaceGetEventGuest*, el lugar de obtención de la información de personas que asistirán al evento, *Executing FaceGet*. Cuando la información es obtenida y almacenada haaciendo uso de evento *Data Received* y Lugar *Storing Data*, el servicio invoca, a través del evento *call FaceGetEnventInfo* la operación obtención de

la información del evento, *Executing FaceGet*. La información obtenida y almacenada temporalmente por el servicio.

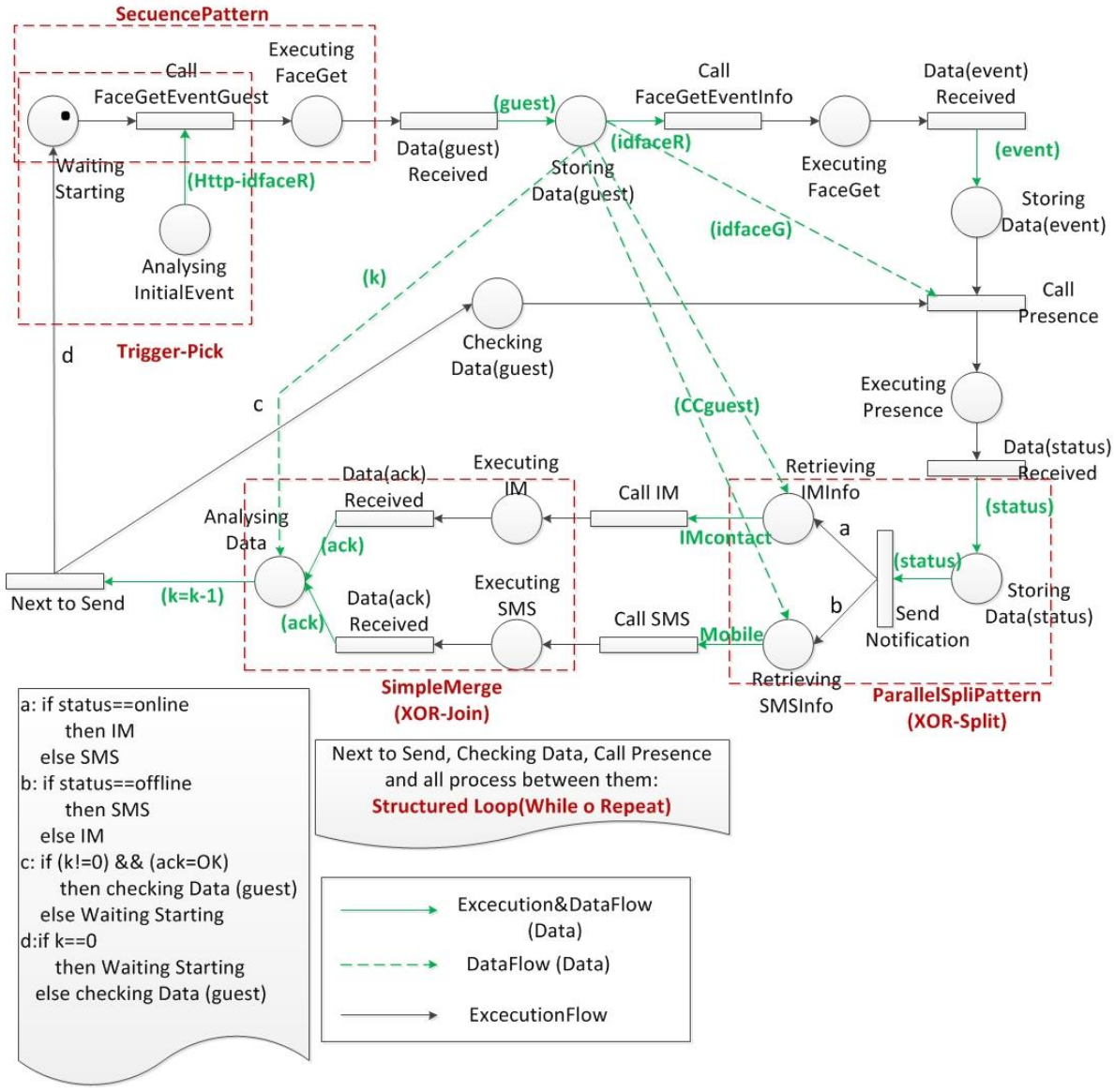

Figura A.3 FacebookEventReminder

Para proceder con las notificaciones de recordatorio, el servicio invoca, a través del evento *call presense*, el lugar de presencia, *Executing presense*. En esta tarea, el servicio reconoce qué asistente al evento está conectado o desconectado de la red a través de un dispositivo de telefonía SIP.

Las notificaciones de recordatorio son enviadas, de acuerdo a su estado de conexión, a cada uno de los participantes del evento en orden secuencial. Par ello, se evalúa el estado de conexión obtenido de la ejecución del servicio de presencia. En el caso donde un asistente al evento tenga un estado “online”, el servicio recupera la información del contacto e invoca el servicio de mensajería instantánea (lugares: *Retrieving IMInfo* y *Executing IM*; eventos: *Call IM* y *Data Received*). En el caso donde un asistente al evento tenga un estado “offline”, el servicio recupera la información de contacto e invoca el servicio de mensajería corta (lugares: *Retrieving SMSInfo Executing SMs*; eventos: *Call SMS* y *Data Received*).

Finalmente, el servicio evalúa si existen más asistentes al evento. En caso de una respuesta afirmativa, el servicio invoca nuevamente el servicio de presencia a través de evento *Next to Send* y lugar *Checking Data*. En caso de una respuesta negativa, donde no haya más asistentes por notificar, el servicio retorna a su estado inicial (*Waiting Starting*) a la espera de un evento que dé comienzo a su flujo de ejecución.

Anexo B

Revisión de servicios Web, de Telecomunicaciones, y Convergentes

En este anexo es presentada una revisión bibliográfica con un conjunto de servicios estandarizados y clasificados por Organizaciones como la 3GPP, ITU, ETSI-TISPAN, ONEAPI y TelcoML; adema se hace una revisión de los servicios que están integrados a SDP's como Mobicents, Rhino y la SDP de Ericsson; finalmente se describen los servicios ofrecidos por algunos operadores de telecomunicaciones.

B.1. Organizaciones de estandarización

Las organizaciones de estandarización son organismos encargados de establecer los diferentes estándares utilizados a nivel mundial en diferentes áreas: telecomunicaciones, redes, sistemas móviles, etc. Dichos estándares corresponden a un conjunto de normas y recomendaciones técnicas que regulan la transmisión en los sistemas de comunicaciones.

Existe una variedad muy grande de organizaciones de estandarización en el mundo, a continuación se presentan algunas de ellas.

3GPP: El Proyecto Asociación de Tercera Generación (3rd Generation Partnership Project), es una colaboración de grupos de asociaciones de telecomunicaciones para la producción de informes y especificaciones que definen las tecnologías de esta organización las cuales han estado en constante evolución a través de lo que se conoce como generaciones de sistemas comerciales móviles /celulares [47].

Las especificaciones del proyecto 3GPP se estructuran como versiones o releases, las cuales pueden tener diferentes revisiones. Cada versión incorpora

centenares de estándares individuales que proporcionan una visión completa y detallada de cómo funciona la industria de las telecomunicaciones móviles y proporcionan a los desarrolladores una plataforma de implementación que permite la adición de nuevas características requeridas por el mercado.

Dentro de las especificaciones podemos encontrar algunas relacionadas con la clasificación de servicios de telecomunicaciones y con algunos estándares, herramientas y APIs que describen su funcionamiento y dan una visión clara acerca de su implementación:

Algunas de estas especificaciones son:

- 3GPP TS 22.002: servicios Portadores soportados por una red PLMN.
- 3GPP TS 22.003: servicios Básicos soportados por una red PLMN.
- 3GPP TS 22.101: principios y aspectos generales de Servicios soportados por una red PLMN.
- 3GPP TS 22.004: aspectos generales de servicios suplementarios.
- 3GPP TS 29.199-[1-22]: servicios Web Parlay X.

ITU: La Unión Internacional de Telecomunicaciones (ITU, International Telecommunication Union) es el organismo especializado de las Naciones Unidas en el campo de las telecomunicaciones, y fue creado el 17 de mayo de 1865 como una organización intergubernamental en la cual los Estados miembros y el sector privado de las telecomunicaciones coordinan el desarrollo y operabilidad de las redes y servicios de telecomunicaciones [48].

La ITU es responsable de la regulación, normalización y desarrollo de las telecomunicaciones a nivel mundial, al tiempo que vela por la armonización de las políticas nacionales de telecomunicaciones. Forman parte de la UIT 193 Estados Miembros y varios centenares asociados.

La ITU está conformada por tres sectores:

- ITU-T: sector de normalización de las telecomunicaciones: estudia los aspectos técnicos, de explotación y tarifarios y publica normativa sobre los mismos, con vista a la normalización de las telecomunicaciones a nivel mundial.

- ITU-R: sector de normalización de las radiocomunicaciones: su función es función es regular la mayor parte del espectro radioeléctrico.

- ITU-D: sector de desarrollo de las telecomunicaciones de la ITU: creado para contribuir a difundir un acceso equitativo, sostenible y asequible a las telecomunicaciones, y de este modo, fomentar un mayor desarrollo económico y social.

En general, la normativa generada por la ITU está contenida en un amplio conjunto de documentos denominados: recomendaciones, agrupados por series. Un ejemplo de estas son las recomendaciones de la ITU-T de la Serie A hasta la Serie Z [49], algunas de las cuales son:

- **Serie F:** servicios de telecomunicaciones no telefónicos. Algunas recomendaciones se muestran en la Tabla B-1.

Tabla B-1: Recomendaciones de la serie F de la ITU

Recomendación	**Descripción**
F.160-F.399	**Servicios telemáticos.**
F.400-F.499	**Servicios de manejo de mensajes.**
F.600-F.699	**Servicios de transmisión de datos.**
F.700-F.799	**Servicios audiovisuales.**

- **Serie I:** red digital de servicios integrados. Algunas recomendaciones se muestran en la

Tabla B-2: Recomendaciones de la serie I de la ITU

Recomendación	**Descripción**
I.112	Vocabulario de términos relativos a las redes digitales de servicios integrado.
I.220	Descripción dinámica de los servicios de

	telecomunicación básicos.
I.230	Definición de las categorías de servicios portadores.
I.240	Definición de teleservicios
I.250	Definición de servicios suplementarios.

- **Serie Y:** infraestructura mundial de la información, aspectos del protocolo internet y redes de la próxima generación. Algunas recomendaciones se muestran en la Tabla B-3.

Tabla B-3: Recomendaciones de la serie Y de la ITU

Recomendación	**Descripción**
Y.2012	Requisitos y arquitectura funcional de una NGN.
Y.2013	Requisitos funcionales y arquitectura del marco de servicios convergentes.
Y.2000	Ejemplos de servicios suplementarios en la convergencia fijo-móvil.

ETSI-TISPAN: el Instituto Europeo de Estándares de Telecomunicaciones (ETSI, European Telecommunications Standards Institute) [50], es un organismo de estandarización cuyo principal logro es el aporte de diferentes estándares para redes GSM.

Cuando se dio el auge de la convergencia de las redes de telecomunicaciones, en el año de 2003 la ETSI creó el grupo TISPAN (Servicios convergentes de internet y telecomunicación & protocolos para redes avanzadas), dada la necesidad de integrar los estándares creados de la 3GPP para redes GSM, a las redes fijas. En el año 2005 el público conoció la versión 1 de la especificación para redes de nueva generación y en el año de 2008, la versión 2, que incluía IMS y capacidades IPTV.

ONEAPI: esta iniciativa define un conjunto de API's e interfaces Web, que permiten a los operadores de redes móviles y otros, exponer la información de la red y capacidades para desarrolladores [51]. Su objetivo es reducir el esfuerzo y el tiempo necesario para crear aplicaciones y contenido. Este conjunto de APIs son expuestas a través de diferentes técnicas software como REST y formatos como JSON, XML, entre otros.

OneAPI define una serie de casos de uso como:

- Servicios de Carga: permite a una tercera parte hacer cobros a un usuario a través del operador.
- Mensajería: permite a una aplicación web enviar y recibir SMS.
- Localización: permite a una aplicación web pedir la localización de uno o más dispositivos móviles que están conectados a un operador de red.
- Perfil de Usuario: permite determinar el estado de usuario y sus preferencias.
- Perfil de conexión de datos: Permite a una aplicación solicitar el tipo de canal portador que utiliza un usuario (3G, GPRS, etc.).

TelcoML: Es una iniciativa que nació desde el 2008, la cual busca representar servicios de telecomunicaciones y servicios IT, a través de UML. En el transcurso de estos años se hicieron grandes avances para que esta nueva iniciativa se convirtiera en estándar, lo cual se logró en 2011[1].

En [52] se presentan las dos principales partes de TelcoML, las cuales son una librería habilitadora y un perfil de composición. El primero de estos representa en UML las principales API Telco; y el segundo, un modelo, también en UML, de los servicios compuestos ejecutables.

Los principales servicios que soporta esta especificación son: SMS & MMS, Click to Call, localización, sincronización, reconocimiento de voz y TTS.

B.2. Plataformas para el despliegue de servicios

Mobicents: el entorno de ejecución (JAIN SLEE) del grupo Mobicents cuenta con una serie de servicios por defecto disponibles al público por medio de documentos de descripción y de su código fuente para fines de implementación.

Algunos de los servicios ofrecidos por este entorno de ejecución son [32]:

[1] La especificación Beta de TelcoML se encuentra en http://www.omg.org/spec/TelcoML/1.0/Beta1/PDF

- Control de llamada: este servicio actúa como un centro de llamadas simple, con funcionalidades como correo de voz, desvío de llamadas y funciones de bloqueo.

- Compras: este servicio consta de una tienda en línea capaz de realizar llamadas a VOIP para informar a los usuarios o administradores acerca del estado de sus cuentas. Los usuarios pueden efectuar las decisiones de compra (aceptar, rechazar, establecer fecha de envío).

- Despertador SIP: esta aplicación procesa mensajes SIP de un Agente SIP registrado para actuar como un sistema despertador. Se utiliza un formato de mensaje específico el cual contiene un temporizador T, un mensaje despertador M y el usuario de destino. Cuando T se cumple el mensaje es enviado.

Rhino Slee: Rhino de Open Cloud [31] [53] ofrece una gran cantidad de servicios los cuales pueden ser implementados con su entorno de ejecución[2]. Algunos de estos se enumeran a continuación.

- Servicios de valor agregado a la llamada básica: terminación de llamada, distribución de llamada, limite a la duración de llamada, desvío de llamada, llamada en espera, identificador de llamada entrante, transferencia de llamada

- Servicios de multimedia: servidor de chat, publicidad durante la llamada, mensajería multimedia, video conferencia, centro de mensajería virtual.

- Servicios de mensajería: texto interactivo, retardo en la entrega de mensajes, traductor, notificación de llamada perdida, texto en tiempo real, información turística.

[2] La página de Desarrolladores de Rhino Slee, presenta alrededor de 35 ejemplos para la implementación de servicios.

- Pagos y facturación: llamada patrocinada, supervisión de límite de crédito, notificación fondos insuficientes, tarjeta de llamadas virtuales, control de privacidad.

- Servicios Web: comunicaciones sociales, video conferencia Web, “Click to Call”.

Ericcson SDP: esta SDP ofrece todas las fases de aprovisionamiento de servicios (creación, despliegue, monitoreo, facturación, retiro, etc.) [29]. Los principales servicios que integran esta plataforma son:

- Un conjunto de habilitadores de servicio utilizados por la capa de ejecución como: SMS, MMS, ubicación, presencia, mensajería instantánea.

- Funciones comunes del aprovisionamiento de servicios como: manejo de dispositivos, facturación, manejo de identidad.

- Acceso a datos: provee un perfil de usuario con sus servicios y características.

- Acceso a contenido de terceros como servidores de medios (descarga de archivos, música, medios sociales, etc.).

B.3. Operadores de telecomunicaciones

Son abundantes los operadores de telecomunicaciones alrededor del mundo que prestan una gran variedad de servicios a sus usuarios dependiendo de la ubicación geográfica en la que se encuentran, algunos de ellos son nombrados a continuación.

Vodafone: Es un operador de carácter multinacional de telefonía móvil y telefonía fija. Ofrece servicios convergentes a sus usuarios entre los cuales se destacan:

- Contestador: permite al usuario convertir los mensajes que se encuentran en el buzón de voz a texto. Este servicio cobra importancia si el

usuario se encuentra ocupado o en una reunión y no puede escuchar los mensajes que se encuentran en su buzón.

- Dicta SMS: permite al usuario que realiza una llamada dejar un mensaje de voz y si este lo desea, el sistema lo convierte a texto, para que el destinatario lo reciba en forma de SMS.

- Desvío de llamadas: permite al subscriptor desviar las llamadas que llegan a su fijo o móvil, a otros destinos; por ejemplo: si el usuario está atendiendo una llamada importante y su teléfono móvil se está quedando sin batería puede desviar las llamadas a otro número fijo o móvil para continuar hablando.

- Rellamada: cuando un usuario desea ponerse en contacto con otro y éste no se encuentra disponible, el servicio de rellamada permite realizar nuevamente una llamada de forma automática, cuando la red detecta que el usuario se encuentra disponible.

- Restricción de llamadas: permite restringir llamadas salientes, entrantes o internacionales.

Tigo Colombia: Es un operador de telecomunicaciones que presta servicios de telefonía móvil, algunos de los servicios que ofrece son [54]:

- Recarga: permite realizar recargas a través de un sitio web, mediante tarjeta de crédito o débito.

- Envío de mensajes: permite establecer conversaciones desde un teléfono móvil y un usuario que cuente con una conexión a internet.

- Tono de llamada (backtones): Permite escuchar una melodía predefinida mientras el usuario espera a que el destinatario conteste; puede ser configurado desde un sitio Web.

- Transferencia de saldo: permite a un usuario enviar saldo a un número Tigo.

- Entretenimiento: el usuario puede descargar diferentes tipos de contenido como: videos juegos, música, televisión, entre otros, a su teléfono móvil.

Claro Colombia:

Ésta empresa surge de la alianza entre Comcel y Telmex Colombia el 26 de Junio de 2012. A partir de la fecha, los servicios que prestaba el operador móvil y el operador de Telecomunicaciones, son ofrecidos por una misma y unificada firma.

Los servicios ofrecidos son: Llamada, llamada en espera, llamada a elegidos, SMS, Video Conferencia, Localización y en general, todo paquete de servicios móviles antes ofrecidos por Comcel. Por otra parte, el conjunto de servicios ofrecidos por Telmex como: televisión Digital, Llamada SIP, Internet, configuración de correo electrónico, pagos, etc.

Anexo C

Patrones del flujo de control de referencia

En este anexo son presentados los 21 patrones del flujo de control representados mediante PN y Grafos, los cuales fueron objeto de estudio teniendo en cuenta [1]. En anexo solo se presentan 21 patrones ya que los demás patrones son de múltiples instancias, y no fueron detectados en el capítulo 3.

C.1. Sequence

Este patrón representa el paso del flujo de un lugar a una transición repetidamente en el caso de la representación en redes de Petri que se muestra en la Figura 1 (Izq); y de forma similar representa el paso de datos de un lugar a otro repetidamente en el caso de la representación mediante grafos (Figura 1. der). Para que se considere un patrón de secuencia debe contener al menos 3 lugares en el caso de las redes de Petri y 3 nodos en el caso de los grafos.

Figura C.1: Patrón Sequence representado mediante redes de Petri y grafos

C.2. PARALLEL SPLIT

Este patrón representa la divergencia del flujo de control de una rama entrante en 2 o más ramas salientes La compuerta ANDJ divide el flujo en cada una de las ramas para que cada una de sus actividades se realice de forma simultánea.

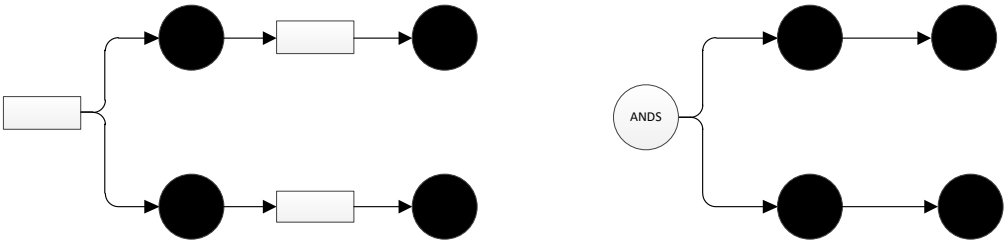

Figura C.2: Patrón Parallel Split representado mediante redes de Petri y grafos

C.3. SYNCHRONIZATION

Este patrón representa la convergencia del flujo de control de dos ramas en una sola, tal que el flujo de control de la rama saliente ocurra solo cuando el flujo de las ramas entrantes, converja después de ejecutarse cada una de sus actividades. La compuerta ANDJ, sincroniza en flujo de las ramas entrantes en un solo flujo saliente.

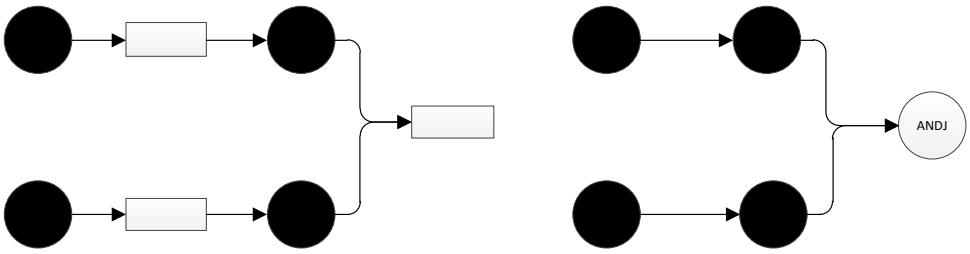

Figura C.3: Patrón Synchronization representado mediante redes de Petri y grafos

C.4. EXCLUSIVE CHOICE

Este patrón representa la divergencia del flujo de control, de una rama entrante en dos o más ramas salientes, teniendo en cuenta una condicional previa. A diferencia del patrón *Parallel Split*, el flujo solo continúa por una de las ramas salientes. La compuerta XORS garantiza la divergencia del flujo en una sola rama.

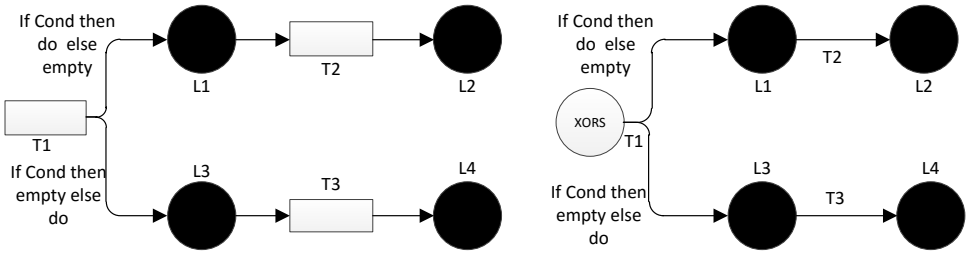

Figura C.4: Patron Exclusive Choice representado mediante redes de Petri y grafos

C.5. SIMPLE MERGE

Este patrón representa la convergencia del flujo de control, de dos o más ramas entrantes en una rama saliente de manera asíncrona. La compuerta XORJ permite que el flujo proveniente de una de las ramas entrantes, independientemente del momento en que termine su ejecución, pase a la rama saliente de forma asíncrona.

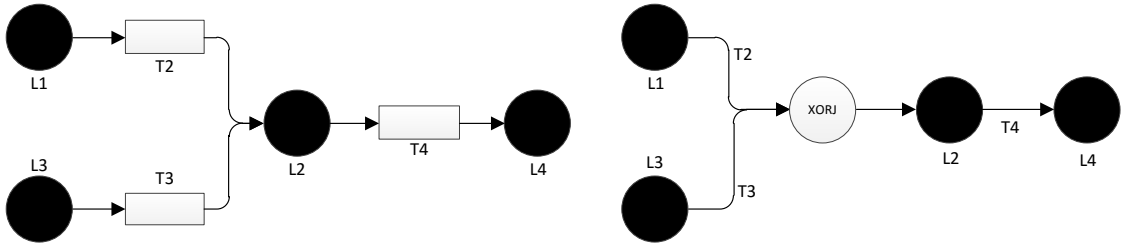

Figura C.5: Patrón Simple Merge representado mediante redes de Petri y grafos

C.6. MULTI-CHOICE

Este patrón representa la divergencia del flujo de control, de una rama entrante en dos o más ramas salientes, teniendo en cuenta una condicional previa. Este patrón se diferencia de *Exclusive Choice* ya que el flujo puede continuar por cualquier rama saliente o por todas las ramas salientes simultáneamente, la compuerta ORS representa estas características. Adicionalmente este patrón puede ser representado con la conjunción de los patrones *Parallel Split* y *Exclusive Choice*.

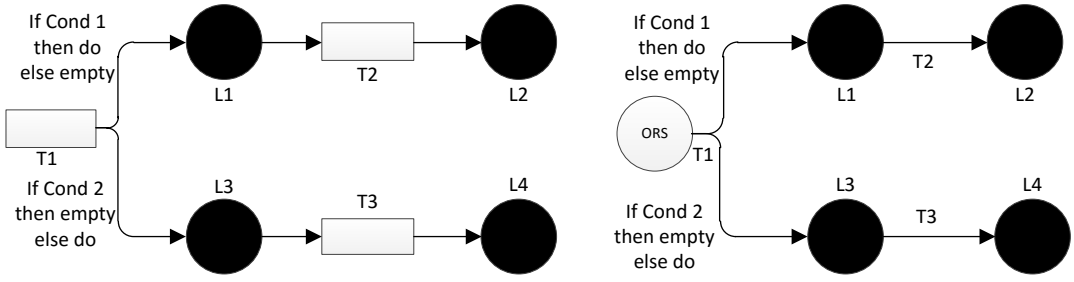

Figura C.6: Patrón Multi Choice representado mediante redes de Petri y grafos

C.7. STRUCTURED SYNCHRONIZING MERGE

Este patrón permite la sincronización del flujo proveniente de dos o más ramas entrantes en una rama saliente. A diferencia del patrón *Synchronization,* este tiene en cuenta el flujo proveniente de las ramas entrantes antes de realizar la

sincronización. La figura 8 muestra una compuerta ORS la cual diverge el flujo por las ramas superiores o inferiores, o por ambas simultáneamente. El flujo es valuado en la transición T4 (Figura 7); si las condiciones 1 y 2 se cumplieron, el flujo ingresa por ambas entradas de T4; si solo una condición se cumplió, el flujo ingresa por solo una rama de T4, mientras que la otra estará en "*empty*" por lo cual el flujo se sincroniza sin tener en cuenta la otra rama.

Para representar este patrón son necesarias dos compuertas lógicas: Una de divergencia ORJ y una de sincronización ANDJ.

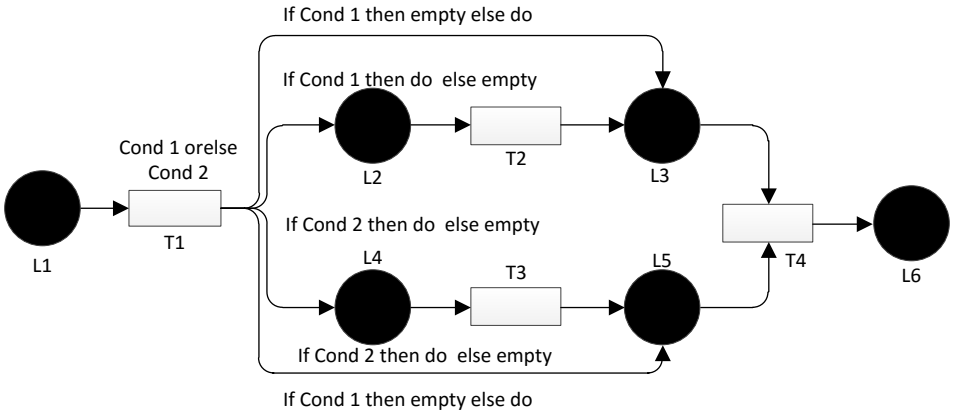

Figura C.7: Patrón Structured Synchronizing Merge representado mediante PN

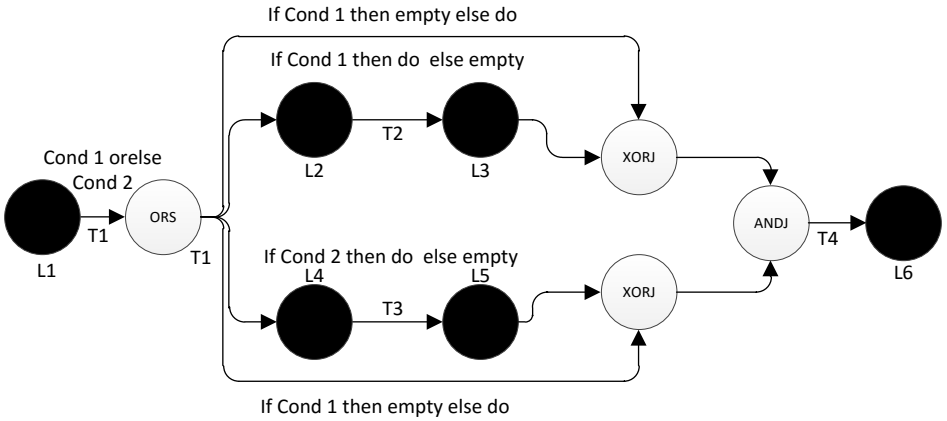

Figura C.8: Patrón Structured Synchronizing Merge representado mediante Grafos

C.8. MULTI MERGE

Este patrón representa la convergencia del flujo de control, de dos o más ramas entrantes en una rama saliente de manera asíncrona. La compuerta ORJ permite

que el flujo proveniente de una o ambas ramas entrantes, independientemente del momento en que termine su ejecución, pase a la rama saliente de forma asíncrona. La diferencia con el patrón *Simple Merge* se ve en la compuerta ORJ, la cual permite recibir el flujo por todas las ramas entrantes.

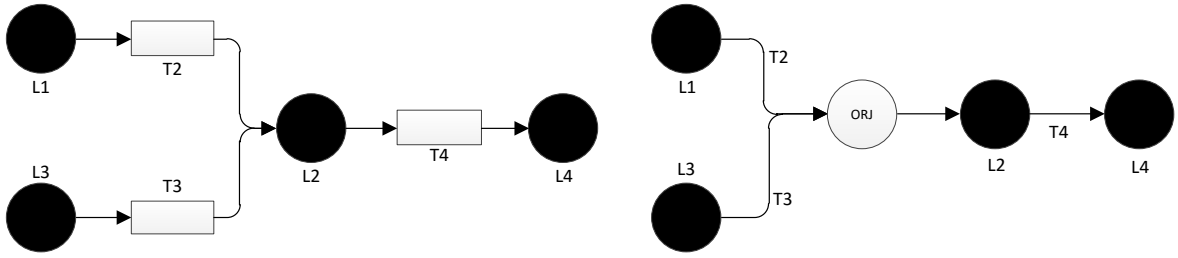

Figura C.9: Patrón Multi Merge representado mediante Grafos y Redes de Petri

C.9. GENERALIZED AND - JOIN

Este patrón permite la sincronización de dos o más ramas de entrada en una única rama de salida independientemente del tiempo de ejecución que se tome cada rama.

Como se observa en la figura 10, el flujo diverge en T1 y posteriormente se sincroniza en T4; este comportamiento se observa de forma similar en la figura 11, donde la compuerta ANDS se encarga de la divergencia del flujo y la compuerta ANDJ se encarga de la sincronización.

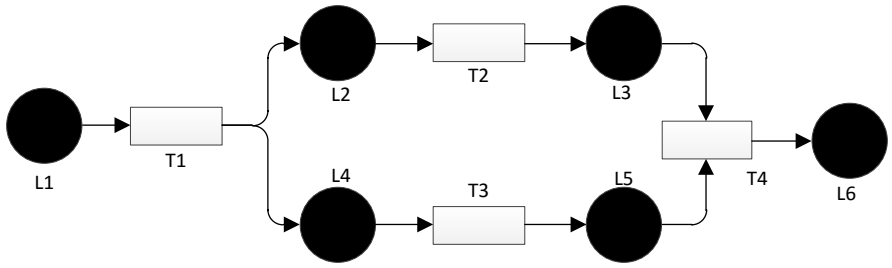

Figura C.10. Patrón Generalized AND-Join representado mediante Redes de Petri

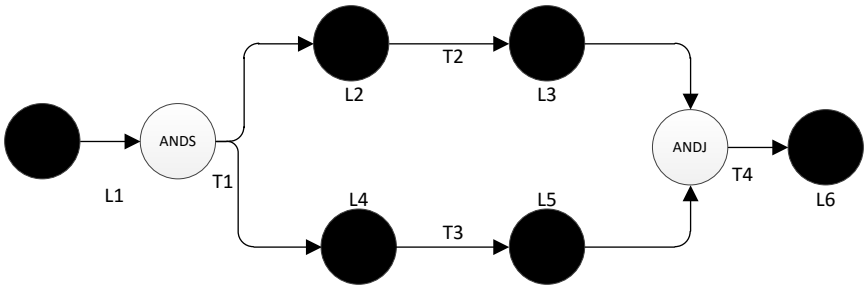

Figura C.11. Patrón Generalized AND-Join representado mediante Grafos

C.10. LOCAL SYNCHRONIZING MERGE

Este patrón es una variación de implementación del patrón *Structured Synchronizing Merge*. La diferencia está en que a lo largo de las ramas se realizan sincronizaciones parciales o locales de flujo. La figura 12 muestra como ocurren la sincronizaciones parciales o locales en los lugares L3, L4, L6 y L7; y la figura 13 muestra las compuertas necesarias para la representacion mediante grafos

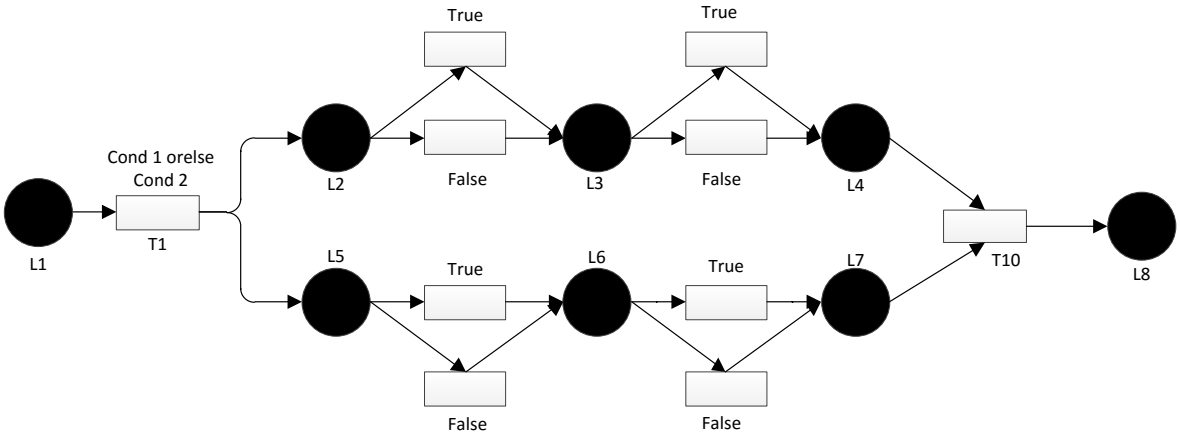

Figura C.12: Patrón Local Synchronizing Merge representado mediante Redes de Petri

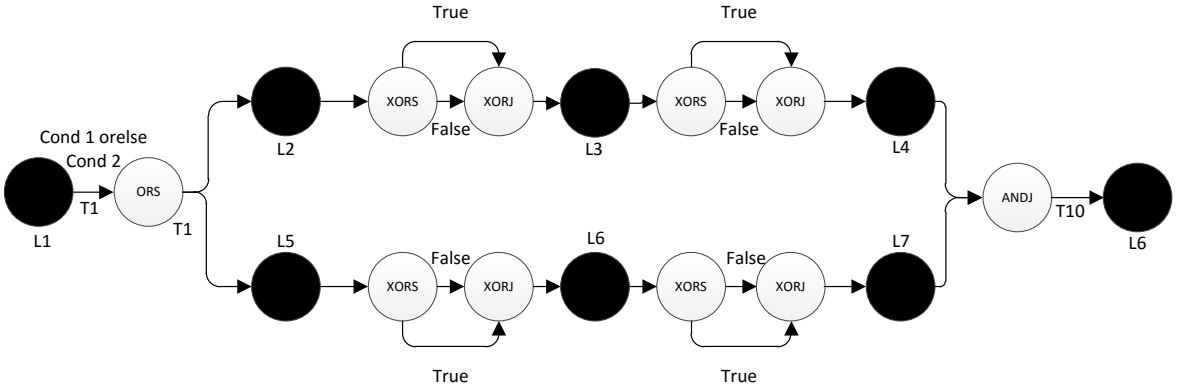

Figura C.13. Patrón Local Synchronizing Merge representado mediante Grafos

C.11. DEFERRED CHOICE

Este patrón diverge el flujo de control de una rama entrante en dos o más ramas salientes. Éste es clasificado dentro de la categoría: patrones de estado, ya que evalúa el flujo entrante para determinar por cual rama saliente continuara dicho flujo. La figura 14 muestra como en la red de Petri, L1 determina si continuara por la rama de la transición T2 o T3 y de forma similar, la compuerta XORS asegura que el flujo pase únicamente una de las dos ramas salientes

Este patrón es utilizado para modelar los casos en los cuales eventos externos como: menajes externos, datos de diferentes ambientes, temporizadores etc. Afectan el flujo de control.

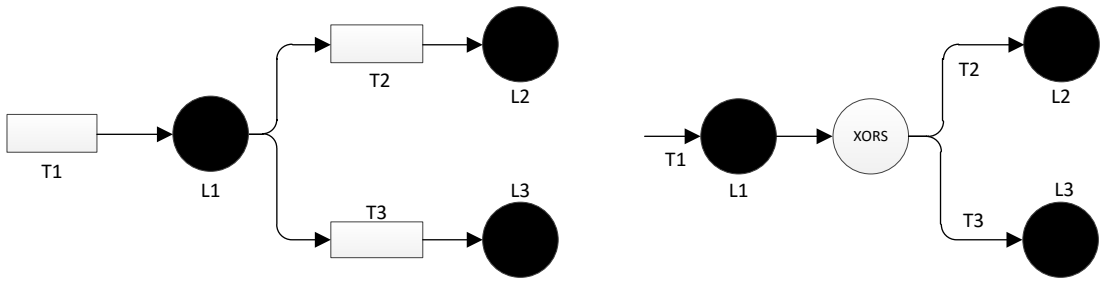

Figura C.14. Patrón Deferred Choice representado mediante Redes de Petri y Grafos

C.12. CRITICAL SECTION

Este patrón diverge y converge el flujo de una manera especial. En la figura se observa como una vez evaluada la transición T1 el flujo diverge de manera síncrona por ambas ramas, sin embargo los lugares y transiciones de la rama superior NO deben ejecutarse ni evaluarse simultáneamente a los lugares y transiciones de la rama inferior, este comportamiento se logra comunicando las secciones críticas por medio de temporizadores o multiplexores que controlen el tiempo en el cual son ejecutadas.

La figura 16 representa este patrón en grafos mediante las compuertas ANDS encargada de la divergencia del flujo de control de forma síncrona y una compuerta ANDJ encargada de sincronizar el flujo que proviene de las ramas entrantes.

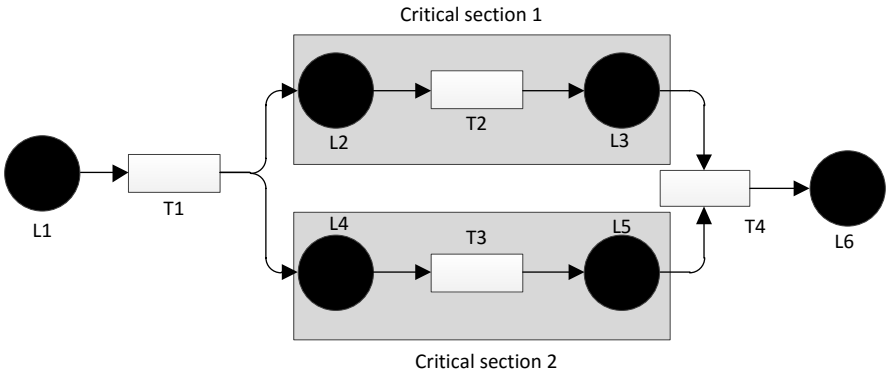

Figura C.15. Patrón Critical Section representado mediante Redes de Petri

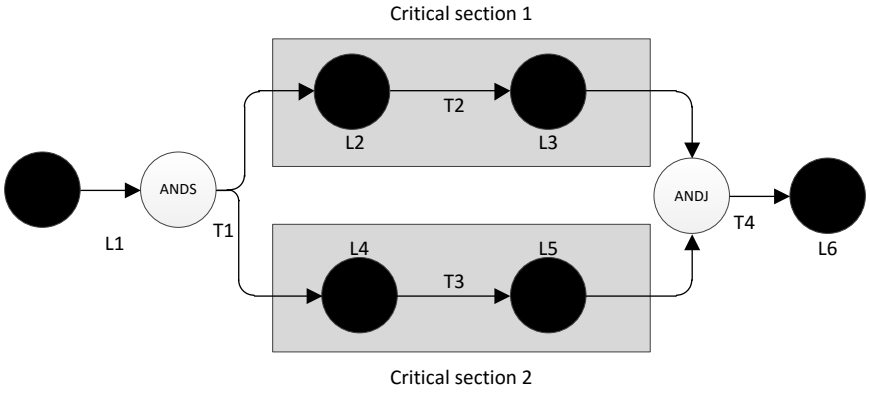

Figura C.16. Patrón Critical Section representado mediante Grafos

C.13. INTERLEAVED ROUTING

Este patrón es muy similar al patrón *Critical Section*. La única diferencia es que no se presentan secciones críticas, pero si es necesario que cada una de las transiciones de las ramas se ejecute de manera aislada al resto de transiciones de la demás ramas. En la figura 17, la transición T2 debe ejecutarse NO simultáneamente a la transición T3; en el momento en que T2 finalice su ejecución, T3 podrá iniciar. En la figura 17 se representa este patrón mediante las compuertas ANDS y ANDJ.

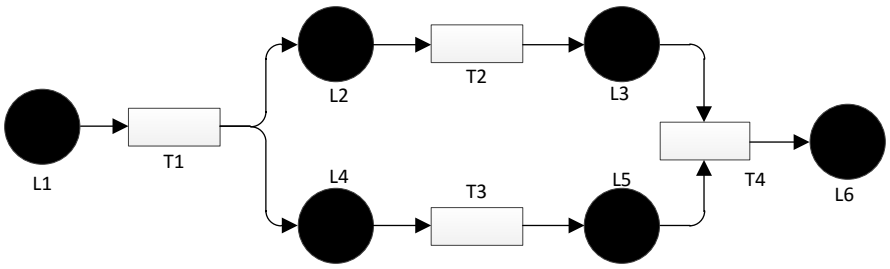

Figura C.17. Patrón Interleaved Routing representado mediante Redes de Petri

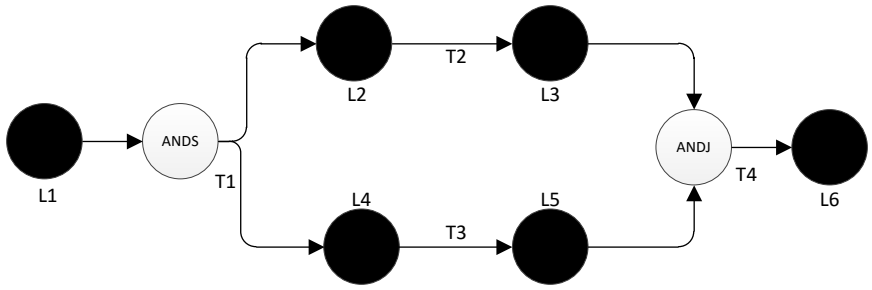

Figura C.18. Patrón Interleaved Routing representado mediante Grafos

C.14. MULTIPLE INSTANCES WITHOUT SYNCHRONIZATION

Este patrón es utilizado para la creación de instancia de una tarea específica. Es representado mediante una compuerta XORS ya que según lo que ocurra en T1 es creada o no dicha instancia.

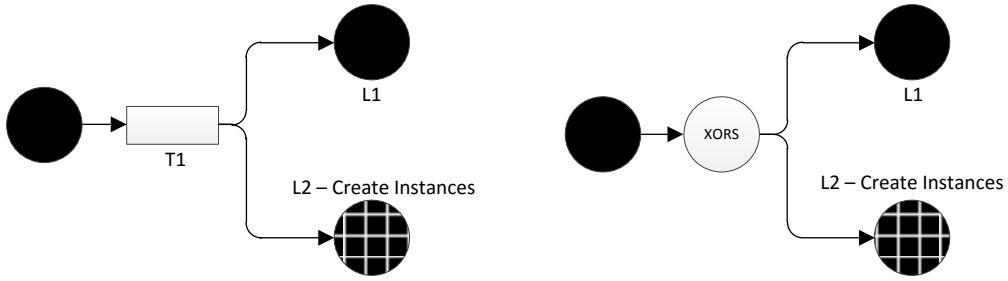

Figura C.19. Patrón Multiple Instances Without Synchronization representado mediante Redes de Petri y Grafos

C.15. CANCEL TASK

Este patrón representa la cancelación de una tarea dentro en el flujo de control. La figura 20 muestra el patrón representado mediante redes de Petri y grafos. Es importante aclarar que su representación puede realizarse mediante una compuerta XORJ, si la rama saliente es dirigida al nodo de cancelación; y mediante una compuerta XORS si una de sus ramas salientes se dirige hacia el nodo de cancelación.

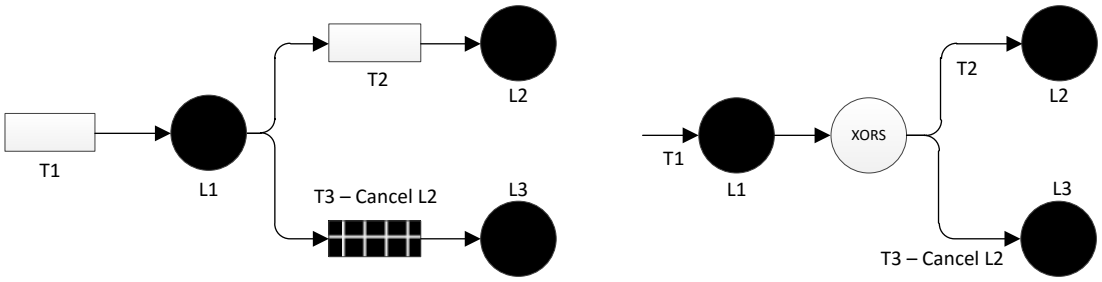

Figura C.20. Patrón Cancel Task representado mediante Redes de Petri y Grafos

C.16. CANCEL CASE

Este patrón representa la cancelación de todo el flujo de control. La figura 21 representa éste patrón mediante redes de Petri y Grafos. La compuerta XORS diverge el flujo en dos ramas, si el flujo sigue por la rama inferior absolutamente todo el flujo es cancelado sin importar que tarea se está ejecutando.

Su representación puede variar mediante una compuerta XORJ, si su rama saliente se dirige al nodo “end”.

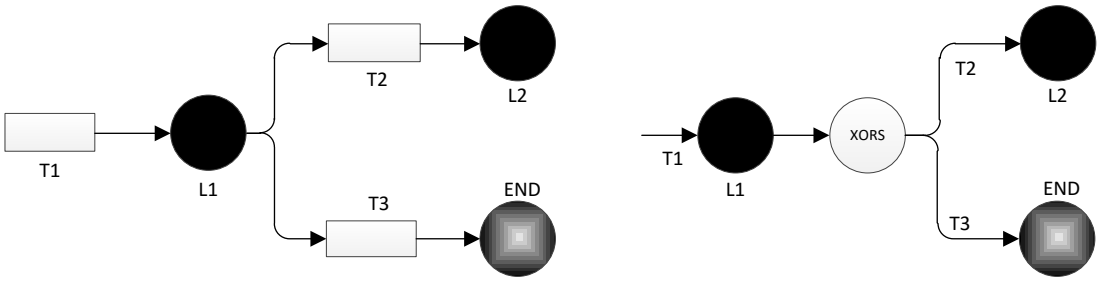

Figura C.21. Patrón Cancel Case representado mediante Redes de Petri y Grafos

C.17. ARBITRARY CYCLES

Este patrón representa ciclos repetitivos de tipo while o repeat presentes en el flujo de control. En la figura 22 se observa que L3 es el lugar que se puede ejecutar n cantidad de veces dependiendo de la condición evaluada en T3; si 1 y se cumple el flujo se devuelve nuevamente a T2 y L3 es nuevamente ejecutado, en caso de no cumplirse; no se hace el ciclo y el flujo termina en L6. La característica que diferencia este patron de “*Structured Loop*”, es la presencia de multiples entradas o salidas en el lugar que se va a ejecutar repetidamente, para el ejemplo L3.

Para su representación por medio de compuertas lógicas es necesario utilizar compuertas XORJ y XORS, cumpliendo las características nombradas, como se muestra en la Figura 23.

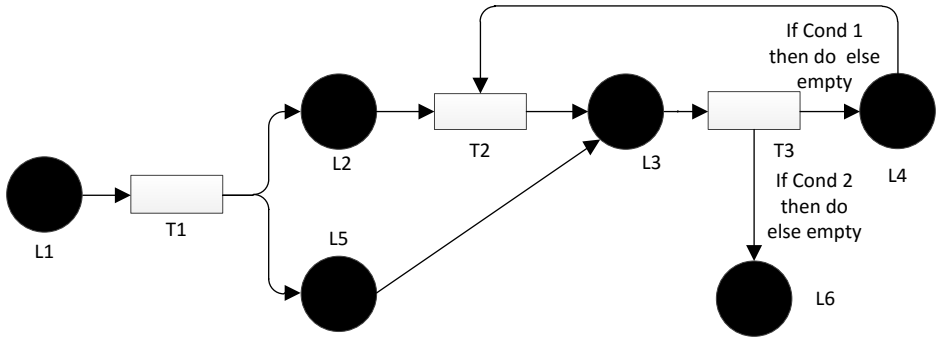

Figura C.22. Patrón Arbitrary Cicles representado mediante Redes de Petri

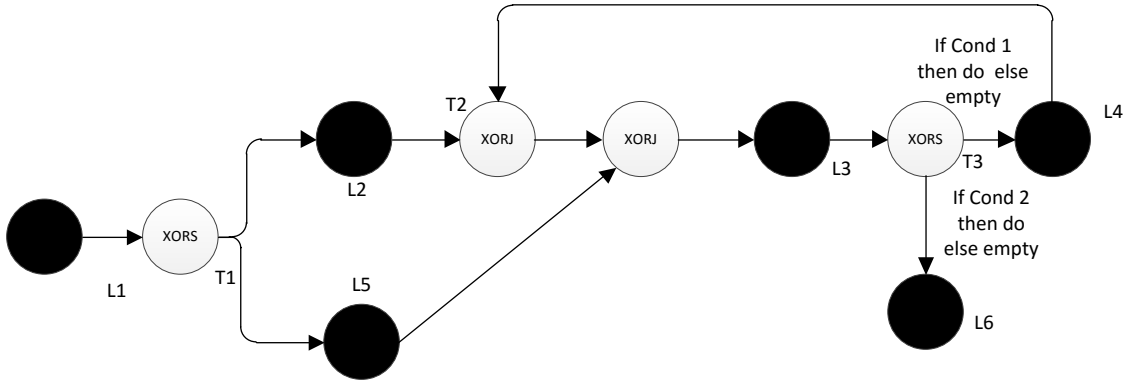

Figura C.23. Patrón Arbitrary Cicles representado mediante Grafos

C.18. STRUCTURED LOOP

Este patrón es similar a *Arbitrary Cicles*, la diferencia está en que este no soporta múltiples entradas y salidas en el lugar que se va a ejecutar repetidamente, como se muestra en la figura 24. Este patrón presenta algunas variaciones dependiendo de la transición donde se evalúan las condiciones. Si la condición se evalúa antes del lugar que se repite, en este caso L3, se denomina *Structured loop pre-test,* en el caso contrario de denomina *Structured loop post-test*

De forma similar al patrón *Arbitrary Cicles* su representación en grafos, como se muestra en la figura 25, se hace mediante compuertas XORJ y XORS

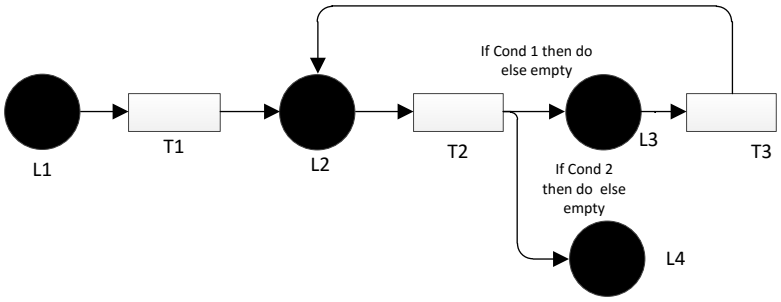

Figura C.24. Patrón Structured Loop representado mediante Redes de Petri

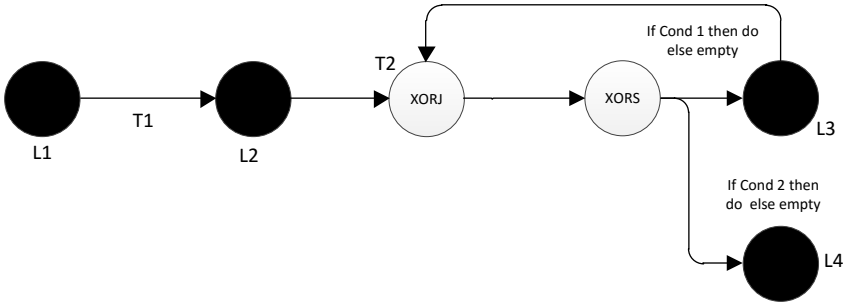

Figura C.25. Patrón Structured Loop representado mediante Grafos

C.19. IMPLICIT TERMINATION

Este patrón representa la ejecución de lugares y transiciones en un determinado en un orden determinado. Es similar *Critical Section*, la diferencia está en que permite realizar la ejecución de las tareas de las ramas superiores e inferiores en cualquier orden. Su representación mediante compuertas lógicas, como se muestra en la Figura 26, se hace a través de ANDS la cual diverge el flujo en dos o mas ramas

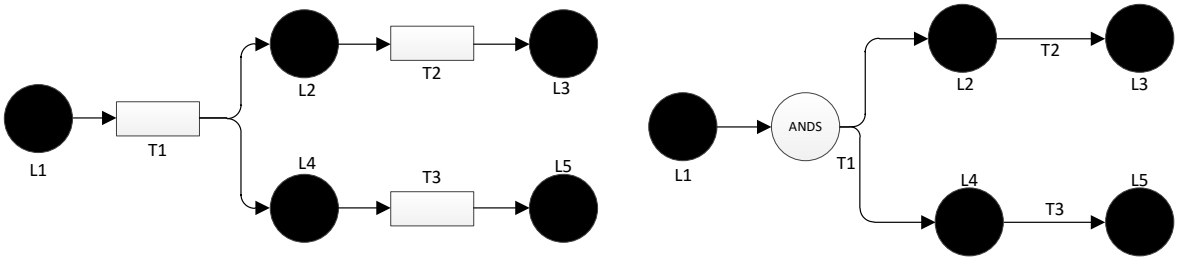

Figura C.26. Patrón Implicit Termination representado mediante redes de Petri y Grafos

C.20. TRANSIENT TRIGGER

Este patrón representa un trigger se produce de forma externa. Este lanzador (trigger) perdura un determinado tiempo. Este debe ser lanzado en un momento determinado cuando la tarea se está ejecutando. Su representación mediante compuertas lógicas, como se muestra en la figura 27, se hace a través de ANDJ, la cual converge el flujo de una rama y el flujo proveniente del lanzador.

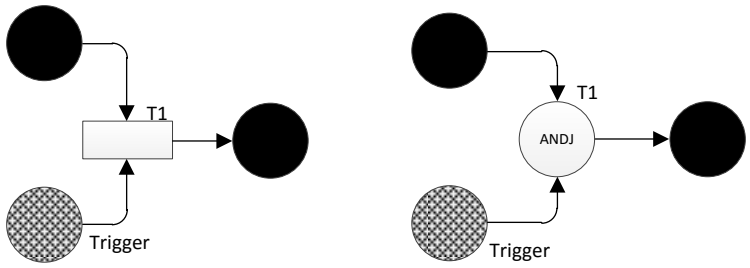

Figura C.27. Patrón Transient Trigger representado mediante Redes de Petri y Grafos

C.21. PERSISTENT TRIGGER

Es una variación de *Transient Trigger*, a diferencia de que cuando es lanzado perdura en el tiempo. En un ejemplo de ensamblaje de autos, se produjo una alarma en el sector de pintura, sin embargo el auto apenas va en ensamblaje, por lo cual no se detiene el proceso de producción hasta que el carro llegue a pintura, cuando la alarma se desactive, el automóvil puede seguir su recorrido hacia otro sector de la fábrica.

C.22. Correspondencia entre patrones y compuertas logicas.

A continuación se presenta una tabla, la cual relaciona patrones de flujo de control con las compuertas necesarias para su representación.

	Patrón	Compuertas	Comentarios
1	Sequence	-	
2	Parallel Split	ANDS	
3	Synchronization	ANDJ	
4	Exclusive Choice	XORS	

5	Simple Merge	XORJ	
6	Multi-choice	ORS	
7	Structured synchronizing merge	ORS + ANDJ	
8	Multi-merge	ORJ	
9	Generalized AND-join	ANDS + ANDJ	
10	Local syncrhonizing merge	ORS + ANDJ	Se necesita sincronizaciones parciales o locales del flujo mediante compuertas XORJ y XORS. Este patrón es una variación de representación del *Structured synchronizing merge*
11	Deferred Choice	XORS	
12	Critical Section	ANDS + ANDJ	
13	Interleaved Routing	ANDS +ANDJ	
14	Multiple instances without synchronization	XORS	
15	Cancel Task	XORS ó XORJ	Una rama, debe ir a un nodo de cancelación. La rama de destino de XORJ debe ir al nodo de cancelación
16	Cancel Case	XORS o XORJ	Una rama debe ir a un nodo de fin La rama de destino de XORJ debe ir al fin
17	Arbitrary Cycles	XORS + XORJ	Una rama de la compuerta XORS es dirigida a la compuerta XORJ.
18	Structured Loop	XORS +XORJ	Una rama de la compuerta XORS es dirigida a la compuerta XORJ.
19	Implicit Termination	ANDS	
20	Transient Trigger	ANDJ	
21	Persistent Trigger	ANDJ	

Tabla 1. Relación entre patrones de Flujo de Control y compuertas lógicas necesarias para su representación

Anexo D

Patrones del flujo de datos de referencia

En este anexo son presentados los 16 patrones del flujo de datos que fueron objeto de estudio en el capítulo 4, teniendo como referencia [1].

Los patrones de datos tienen como objetivo capturar las diversas formas en la cual, los datos son transmitidos entre tareas[3] y entornos externos, independientemente de la tecnología de implementación.

Para la representación de los 16 DP soportados por ambos lenguajes, se presenta la convención que se muestra en la Tabla .

Tabla D.1: Convención para los patrones de Datos

Grafico	Descripción
	Lugar en redes de Petri, representa un estado en el Flujo de Control.
	Transicion en redes de Petri, representa eventos que activan o disparan lugares.
	Arco en redes de Petri, representa la conexión entre lugares y transiciones.
	Flujo de datos entre transiciones y lugares
	Flujo de datos entre lugares, transiciones y aplicaciones externas.
	Flujo de datos entre lugares, transiciones

[3] Tareas: Se refiere específicamente al Workflow.

	y aplicaciones externas.
External Aplication	Aplicación externa al Flujo de Control (servicios web, repositorios de datos, etc.)
Var x = 21	Éste rectángulo indica el valor de las variables para un único lugar o transición, o para un grupo de lugares y transiciones.

D.1. Scope Data

Éste patrón representa la definición de una variable para un conjunto de lugares y transiciones. En la Figura D-1-1 se observa que solo L1 y T1 hacen uso de la variable X y ninguno de los otros lugares y transiciones pueden hacer uso de ella.

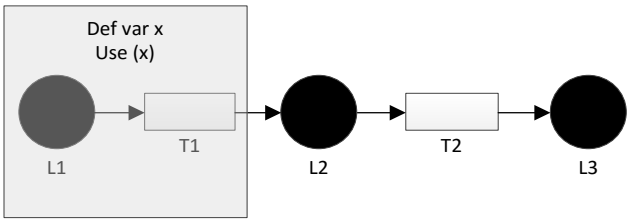

Figura D-1: Patrón Scope Data

D.2. Case Data

Este patrón representa la definición y el uso de datos para todo el flujo, es decir, todos los lugares y transiciones utilizan datos establecidos previamente. La Figura D-2 representa el patrón Case Data.

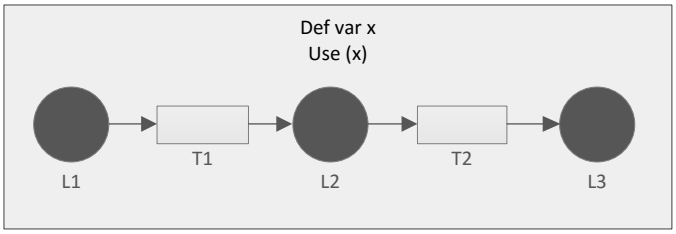

Figura D-2: Patrón Case Data

D.3. Environment Data

Los lugares y transiciones hacen uso de datos los cuales son establecidos por entornos externos. La figura Figura **D-3** representa el patrón Environment Data.

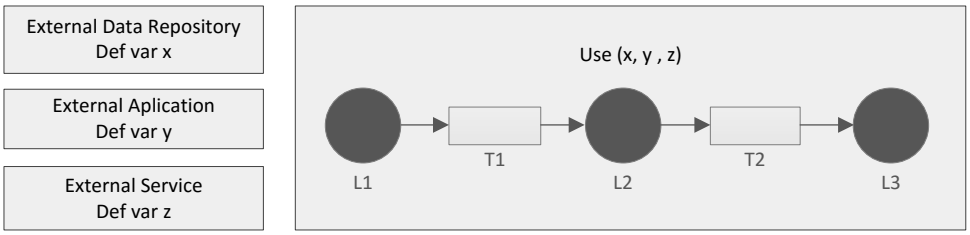

Figura D-3: Patrón Environment Data

D.4. Data Interaction between Tasks

Este patrón representa el paso de datos entre lugares y transiciones continuas como es el caso de L1 y T1; y el paso de datos entre lugares-lugares como es el caso de L2 - L3. Lo más importante es que soporta el intercambio de datos sin tener en cuenta el flujo de control. La Figura D-4 representa el patron task to task.

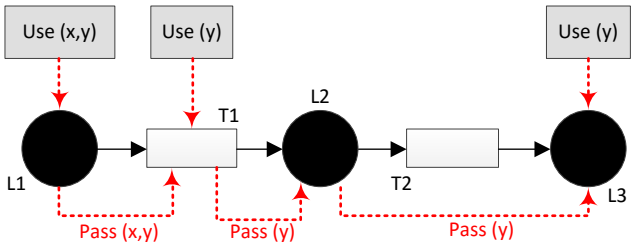

Figura D-4: Patrón Data Interacion between Tasks

D.5. Data Interaction - Task to Environment - Push-Oriented

Este patrón representa el intercambio de datos entre un proceso externo al flujo. En el ejemplo que se muestra en la Figura D-5, las transiciones y lugares al ejecutarse establecen el valor de las variables m,z y j en una aplicación externa.

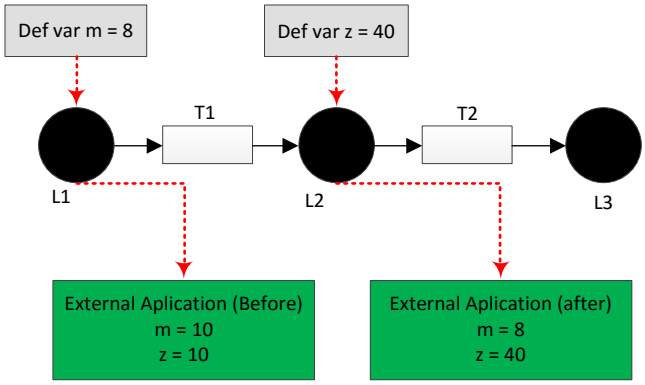

Figura D-5: Patron Data Interaction - Task to Environment – Push - Oriented

D.6. Data Interaction - Environment to Task - Pull-Oriented

Este DP es similar al patron *Task to Environment – Push Oriented,* la diferencia está en que la aplicación externa pasa los datos a los lugares y transiciones y puede cambiar las variables que ya estaban predefinidas. La Figura D-6 muestra el paso de datos entre la aplicación externa y los lugares L1 y L2 a través de peticiones y respuestas (request & response).

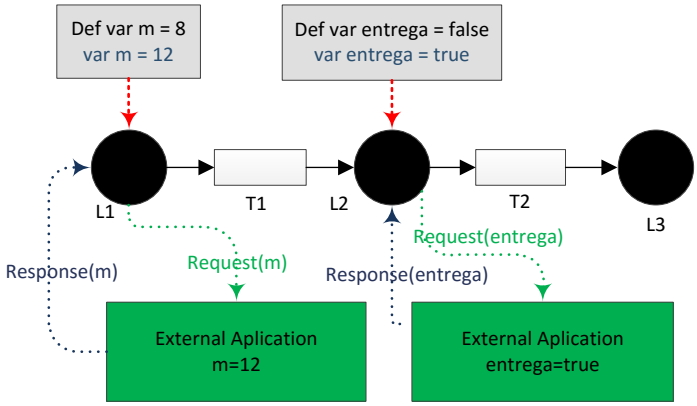

Figura D-6: Patron Data Interaction - Task to Environment – Push – Oriented

D.7. Data Interaction - Task to Environment - Pull-Oriented

Es un patrón que permite el intercambio de datos mediante peticiones y respuestas entre los lugares y transiciones de un determinado proceso, y cualquier aplicación externa. La diferencia con los patrones *Task to Environment – Push Oriented y Environment to Task – Pull Oriented* es que permite el paso de datos sin tener en cuenta la definición de las variables. La Figura D-7 muestra un ejemplo del intercambio de datos entre L1, L2 y algunas aplicación externas, para el caso de L1, la aplicación externa hace una petición a éste para que le pase el valor de la variable que está definida, en este caso m, por esto la aplicación externa establece el valor de j en con el mismo valor de m. De forma similar ocurre con L2

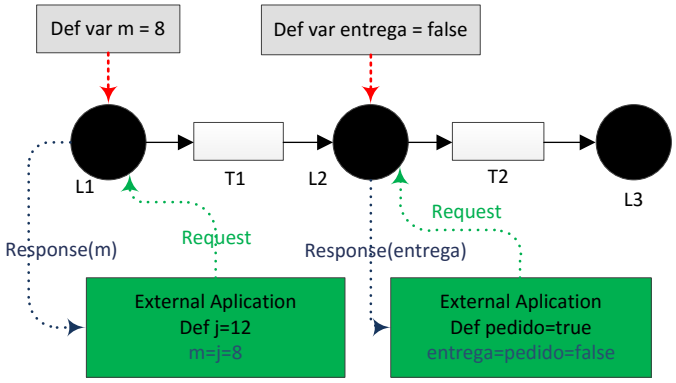

Figura D-7: Patrón Data Interaction -Task to Environment - Pull-Oriented

D.8. Data Passing by Value - Incoming & Outgoing

En este patrón se representa la transferencia de datos por valor. La Figura D-8 muestra un ejemplo en donde todas las variables (x,y,z) son establecidas en 21.

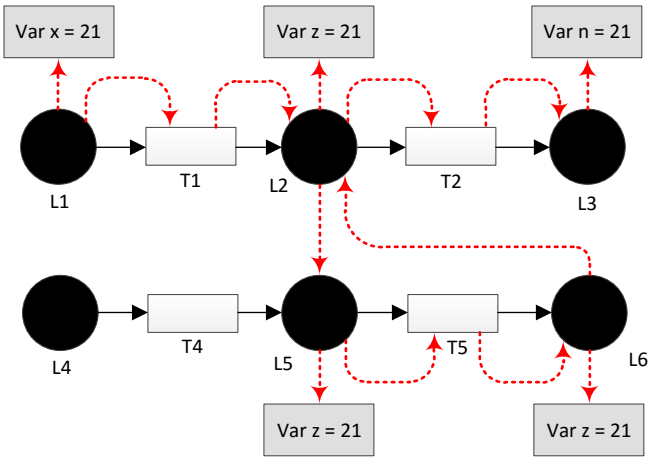

Figura D-8: Patrón Data Passing by Value – Incoming & Outgoing

D.9. Data Passing - Copy In/Copy Out

Este patrón representa el paso de datos entre nodos y transiciones y una aplicación externa con características de almacenamiento de datos, por ejemplo, bases de datos y repositorios. La figura Figura D-9 muestra como L1, L2 y L3 obtienen los datos de una base de datos.

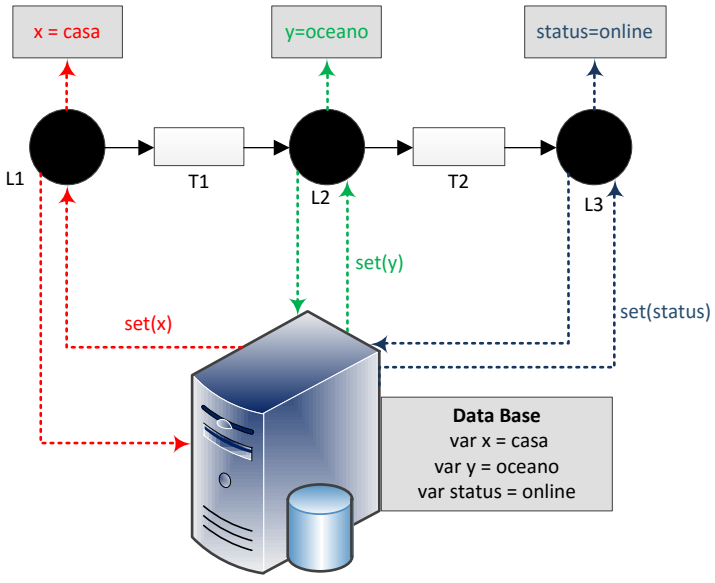

Figura D-9: Patrón Data Passing - Copy In/Copy Out

D.10. Data Passing by Reference – Unlocked

Éste patrón representa la transferencia de datos de entornos externos entre lugares y transiciones. Es similar al patrón case data, sin embargo, la diferencia está en que los datos se tienen que sincronizar, es decir, el flujo de control es tenido en cuenta para el intercambio de datos. La Figura D-10 ilustra el patrón Data Transfer by Reference.

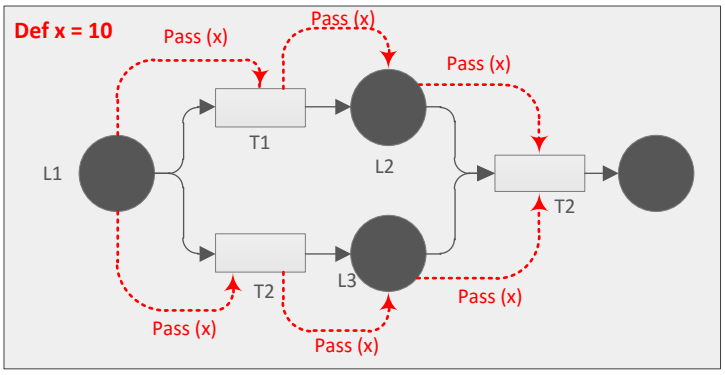

Figura D-10: Patrón Data Passing by Reference – Unlocked

D.11. Data Transformation – Input & Output

Este patrón representa la transformación que van sufriendo los datos a medida que estos son transferidos entre lugares y transiciones. La Figura D-11 muestra como la variable x sufre continuas transformaciónes. La diferencia entre el patrón *Data Transformation Input* y *Output*, simplemente se relaciona con el lugar donde se realiza la transformación; si es a la entrada de un lugar o transición, se dice que es de tipo *Input*, en caso contrario, si se realiza al finalizar la lógica de implementación de cada elemento, se dice que es una transformación de tipo *Output*.

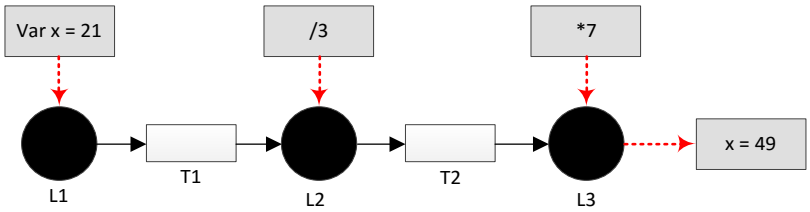

Figura D-11: Patrón Data Transformation – Input & Output

D.12. Task Precondition - Data Value

Este patrón representa un condicional previo existente el cual se tiene que cumplir para que el flujo pueda continuar. Para el ejemplo de la Figura D-12 el flujo puede continuar a L2 si se cumple la condicion x > 10.

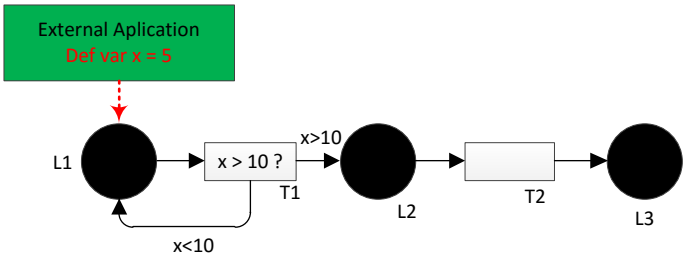

Figura D-12: Patrón Task Precondition - Data Value

D.13. Event-based Task Trigger

Este patrón representa el disparo de las actividades del flujo mediente actividades o ambientes externas, como por ejemplo: servicios web, aplicaciones de terceros, etc. La Figura D-13 representa el patron Event – based task Trigger.

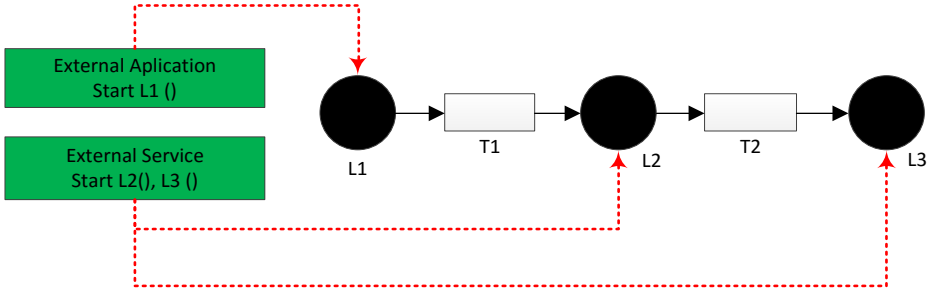

Figura D-13: Patrón Event-based Task Trigger

D.14. Data-based Routing

Este patrón tiene especial relación con los CFP representados mediante compuertas lógicas de tipo ORS y XORS, como lo son: *MultiChoice, Exclusive Choice,* etc. ya que la rama por la cual va a continuar el flujo, depende de los datos establecidos previamente. La Figura D-14, representa el patrón Data – based Routing.

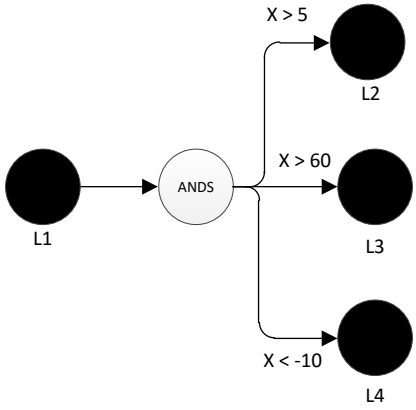

Figura D-14: Patrón Data-based Routing

Anexo E

Templates utilizados en el mecanismo

En este anexo son presentados los templates de código que fueron utilizados en el prototipo que implemento el mecanismo de orquestación automática. Dichos templates están divididos en cuatro grupos: templates básicos, templates de eventos, templates de patrones y templates XML. A continuación son presentados cada uno.

E.1. Templates básicos:

En este grupo se encuentran 11 templates, los cuales contienen los códigos básicos del SBB como las librerías, constructores, contexto, actividades, evento inicial, y la plantilla principal del SBB. A continuación son listados.

- **Template import:**

```
package sbb;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.slee.*;
import javax.slee.nullactivity.NullActivity;
import javax.slee.nullactivity.NullActivityContextInterfaceFactory;
import javax.slee.nullactivity.NullActivityFactory;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
```

```
import javax.slee.resource.ActivityAlreadyExistsException;
import javax.slee.resource.StartActivityException;
import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.io.PrintWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.ArrayList;
import net.java.slee.resource.http.HttpServletRaActivityContextInterfaceFactory;
import net.java.slee.resource.http.HttpServletRaSbbInterface;
import net.java.slee.resource.http.HttpSessionActivity;
import org.telcomp.events.StartWSInvocatorEvent;
import org.telcomp.events.EndWSInvocatorEvent;
```

- **Template servicestart:**

```
public void onServiceStartedEvent ( javax.slee.serviceactivity.ServiceStartedEvent event, ActivityContextInterface aci) {
        System.out.println("!!!!!!!!!!! Se activa el servicio Compuesto: "+this.getSbbContext().getSbb().getName());
         aci.detach(this.sbbContext.getSbbLocalObject());
    }
```

- **Template userdata:**

```
public void onGET(net.java.slee.resource.http.events.HttpServletRequestEvent event, ActivityContextInterface aci) {
        aci.detach(sbbContext.getSbbLocalObject());
        //estas 3 lineas y el IF deben ir en todo SBB que tenga HTTP Servlet
        HttpServletRequest request = event.getRequest();
        String uri = request.getRequestURI().substring(11);
        uri = uri.substring(0, uri.indexOf("/")).replaceAll("%20", " ");
         if (uri.equals(this.getSbbContext().getSbb().getName())) {
                //dentro de este If se pone lo que debe hacer con el evento
```

```
                System.out.println("Entra al SBB: "+this.getSbbContext().getSbb().getName());
                int n = sbbContext.getSbbLocalObject().hashCode();
                System.out.println("!!!!!!!!!!!!!!!! Id objeto sbb: "+n);
                System.out.println("Entra  al GET !!!!!!!!!!!!!!!!!!!!");
                response = event.getResponse();
                try {
                        w = response.getWriter();
                        w.print("<script> window.history.back(); </script>");
                        w.flush();
                        response.flushBuffer();
                        System.out.println("HttpServletRAExampleSbb: GET received and OK! response sent.");
                } catch (Exception e) {
                        System.out.println("no  se pudo");
                }
            endUser_Data = true;
            endData_Module = true;
            endSTART = true;
            this.createNullActivityACI();
          //_userData_
          this.dataMapping();
          this.controlFlow();
          this.triggerEvent();
        }
    }
```

- **Template userVar:**

```
_userVar_ = event.getRequest().getParameter("_userVar2_");
```

- **Template mapping:**

```
private void dataMapping() {
   _mapping_
}
```

- **Template setdata:**

```
_input_ = _output_;
```

- **Template httpEvent:**

```
public void onSessionPOST(net.java.slee.resource.http.events.HttpServletRequestEvent event, ActivityContextInterface aci) {
        HttpServletRequest request = event.getRequest();
         System.out.println("----------*  " + request.getRequestURI());
        String uri = request.getRequestURI().substring(11);
        uri = uri.substring(0, uri.indexOf("/")).replaceAll("%20", " ");
        if (uri.equals(this.getSbbContext().getSbb().getName())) {
                System.out.println("Entra al SBB: "+this.getSbbContext().getSbb().getName());
                int n = sbbContext.getSbbLocalObject().hashCode();
                System.out.println("!!!!!!!!!!!!!!!! Id objeto sbb: "+n);
                session = request.getSession(true);
                if (session == null) {
                        System.out.println("!!!!!!! La asesion esta nula");
                } else {
                        System.out.println("!!!!!!! La sesion es: "+session.getId());
                }
                System.out.println("Entra al POST CON session !!!!!!!!!!!!!!!!!!!");
               ByteArrayOutputStream baos = new ByteArrayOutputStream();
               try {
                  InputStream is = request.getInputStream();
                  byte[] b = new byte[1];
                  while (is.read(b) != -1) {
                baos.write(b);
                  }
               }catch (IOException ex) {
                  System.out.println("!!!!!!! No pudo leer en contenido del Request");
               }
              String datos = new String(baos.toByteArray());
```

```
                System.out.println("HTTP POST Request Content: " + datos);
                activator = datos;
                HttpServletResponse response = event.getResponse();
                try {
                        PrintWriter w = response.getWriter();
                        w.print("onPOST OK! Served by SBB = " +
this.getSbbContext().getSbb().toString());
                        w.flush();
                        response.flushBuffer();
                        System.out.println("POST Session received and OK! Response
sent.");
                } catch (Exception e) {
                        System.out.println("No se pudo enviar el Response");
                }
                this.transientTrigger();
        }
    }
    public void onPOST(net.java.slee.resource.http.events.HttpServletRequestEvent
event, ActivityContextInterface aci) {
        aci.detach(sbbContext.getSbbLocalObject());
        HttpServletRequest request = event.getRequest();
        String uri = request.getRequestURI().substring(11);
        uri = uri.substring(0, uri.indexOf("/")).replaceAll("%20", " ");
        if (uri.equals(this.getSbbContext().getSbb().getName())) {
                System.out.println("!!!!!!!! Entra al SBB:
"+this.getSbbContext().getSbb().getName());
                int n = sbbContext.getSbbLocalObject().hashCode();
                System.out.println("!!!!!!!! Id objeto Sbb: "+n);
                session = request.getSession();
                if (session == null) {
                        System.out.println("!!!!!!! La asesion esta nula");
                } else {
                        System.out.println("!!!!!!! La sesion es: "+session.getId());
                }
                System.out.println("Entra al POST SIN session !!!!!!!!!!!!!!!!!!!");
```

```
                this.createHttpSessionACI(session);
                httpSessionACI.attach(sbbContext.getSbbLocalObject());
                ByteArrayOutputStream  baos = new ByteArrayOutputStream();
                try {
                    InputStream  is = request.getInputStream();
                    byte[] b = new byte[1];
                    while (is.read(b) != -1) {
                baos.write(b);
                    }
                }catch (IOException ex) {
                    System.out.println("!!!!!!! No pudo leer en contenido del Request");
                }
                String datos = new String(baos.toByteArray());
                System.out.println("HTTP  POST Request Content: " + datos);
                activator = datos;
                HttpServletResponse  response = event.getResponse();
                try {
                        PrintWriter w = response.getWriter();
                        w.print("onPOST OK! Served by SBB = " +
this.getSbbContext().getSbb().toString());
                        w.flush();
                        response.flushBuffer();
                        System.out.println("POST Request received and OK! Response
sent.");
                } catch (Exception e) {
                        System.out.println("no  se pudo");
                }
                this.transientTrigger();
        }
    }
```

- **Template classstart:**

```
public abstract class _name_Sbb implements javax.slee.Sbb {
```

```
    private NullActivityFactory nullActivityFactory;
    private NullActivityContextInterfaceFactory nullACIFactory;
    private SbbContext sbbContext;
    _activity_
    private HttpServletRaSbbInterface httpServletRaSbbInterface;
    private HttpServletRaActivityContextInterfaceFactory
httpServletRaActivityContextInterfaceFactory;
    private ActivityContextInterface httpACI;
    private PrintWriter w;
    private HttpServletResponse response;
    private boolean finishService;
    public void setSbbContext(SbbContext context) {
        this.sbbContext = context;
       //_dataModule_
        try {
                Context ctx = (Context) new InitialContext().lookup("java:comp/env");
                nullActivityFactory = (NullActivityFactory)
ctx.lookup("slee/nullactivity/factory");
                nullACIFactory = (NullActivityContextInterfaceFactory)
ctx.lookup("slee/nullactivity/activitycontextinterfacefactory");
              httpServletRaSbbInterface = (HttpServletRaSbbInterface)
ctx.lookup("slee/resources/mobicents/httpservlet/sbbrainterface");
                httpServletRaActivityContextInterfaceFactory =
(HttpServletRaActivityContextInterfaceFactory)
ctx.lookup("slee/resources/mobicents/httpservlet/acifactory");
          } catch (NamingException e) {
                e.printStackTrace();
        }
    }
    private void createNullActivityACI() {
        _aci_
    }
    public void unsetSbbContext() {
        this.sbbContext = null;
    }
```

```
public void sbbCreate() throws javax.slee.CreateException {}
public void sbbPostCreate() throws javax.slee.CreateException {}
public void sbbActivate() {}
public void sbbPassivate() {}
public void sbbRemove() {}
public void sbbLoad() {}
public void sbbStore() {}
public void sbbExceptionThrown(Exception exception, Object event,
ActivityContextInterface activity) {}
public void sbbRolledBack(RolledBackContext context) {}
protected SbbContext getSbbContext() {
    return sbbContext;
}
public abstract void fireStartWSInvocatorEvent (StartWSInvocatorEvent event,
ActivityContextInterface aci, Address address);
public void onEndWSInvocatorEvent (EndWSInvocatorEvent event,
ActivityContextInterface aci){
    System.out.println("<<<---- Recibe evento Web Service");
    //_wsinvocator_
    this.dataMapping();
    this.controlFlow();
    this.triggerEvent();
}
private boolean endSTART = false;
private boolean endUser_Data = false;
private boolean endData_Module = false;
private boolean endEND = false;
```

- **Template detach:**

```
nullActivityACI.detach(sbbContext.getSbbLocalObject());
session.invalidate();
```

- **Template nullactivity:**

```
NullActivity null_aci_ = this.nullActivityFactory.createNullActivity();
```

```
_aci_ = this.nullACIFactory.getActivityContextInterface(null_aci_);
_aci_.attach(this.sbbContext.getSbbLocalObject());
```

- **Template classend:**

```
}
```

E.2. Templates de eventos

En este grupo se encuentran 9 templates, los cuales contienen los códigos asociados al envío y recepción de los eventos, incluyendo las actividades y datos necesarios en cada uno. A continuación son listados.

- **Template invoke:**

```
this.fireStart_name_Event(event, _aci_, null);
```

- **Template receiveevent:**

```
public void onEnd_name_Event(End_name_Event event, ActivityContextInterface
aci) {
        System.out.println("<<<----  Recibe evento _name_ !!!!! ");
      //_data_
      //_receiveState_
      this.dataMapping();
      this.controlFlow();
      this.triggerEvent();
}
```

- **Template activities:**

```
if(aci.equals(_activity_)){
   end_name_ = true;
   //_data_
}
```

- **Template activitiesWS:**

```
if(aci.equals(_activity_)){
   //_var_
   end_name_ = true;
}
```

- **Template getdata:**

```
_output_  = event.get_input_();
```

- **Template getdataWS:**

```
_output_ = event.getOperationOutputs().get("_output2_").get(0);
```

- **Template importevent:**

```
import org.telcomp.events.Start_name_Event;
import org.telcomp.events.End_name_Event;
```

- **Template controlflow:**

```
private void controlFlow(){
   if((endSTART == true) && (endUser_Data == true) && (endData_Module == true))
{
      start_name_ = true;
   }
_flow_
}
```

- **Template sequence:**

```
   if(end_lastEvent_  == true) {
      end_lastEvent_  = false;
      start_nextEvent_  = true;
   }
```

E.3. Templates de patrones

En este grupo se encuentran 5 templates, los cuales contienen los códigos correspondientes a los patrones implementados en el prototipo. A continuación son listados.

- **Template sequence:**

```
   if(end_lastEvent_  == true) {
      end_lastEvent_  = false;
      start_nextEvent_  = true;
      System.out.println("  << Patron >> paso por el Sequence");
   }
```

- **Template AND-Split:**

```
if(end_lastEvent_ == true) {
    end_lastEvent_ = false;
    start_nextEvent1_ = true;
    start_nextEvent2_ = true;
    System.out.println("<< Patron >> paso por el AND-Split");
}
```

- **Template AND-Join:**

```
if(end_lastEvent1_ == true && end_lastEvent2_ == true) {
    end_lastEvent1_ = false;
    end_lastEvent2_ = false;
    start_nextEvent_ = true;
    System.out.println("<< Patron >> paso por el AND-Join");
}
```

- **Template transientTrigger:**

```
private void transientTrigger(){
    System.out.println("!!!!!!!!!! Entra al Trigger");
    System.out.println("!!!!!!!!!! La variable Start: " + this.start);
    //_node_
}
```

- **Template transientTriggerNode:**

```
if(_lastService_ == true && activator.equals("_thisService_")){
    //_invoke_
}
```

E.4. Templates XML:

En este grupo se encuentran 8 templates, los cuales contienen los códigos xml de todos los descriptores de servicios, SBB, eventos, RA y DU. A continuación son listados.

- **Template basexmlsbb:**

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE sbb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD JAIN SLEE SBB
1.1//EN"
                        "http://java.sun.com/dtd/slee-sbb-jar_1_1.dtd">
```

```
<sbb-jar>
  <sbb>
    <description />
    <sbb-name>_name_</sbb-name>
    <sbb-vendor>David Ramirez</sbb-vendor>
    <sbb-version>1.0</sbb-version>
    <sbb-classes>
      <sbb-abstract-class reentrant="False">
        <sbb-abstract-class-name>sbb._name_Sbb</sbb-abstract-class-name>
      </sbb-abstract-class>
    </sbb-classes>
    <!-- _event_ -->
    <!-- _ra_ -->
  </sbb>
</sbb-jar>
```

- **Template fireevent:**

```
    <event event-direction="Fire" initial-event="False" mask-on-attach="False" last-in-
transaction="True">
      <event-name>Start_name_Event</event-name>
      <event-type-ref>
        <event-type-name>Start_name_Event</event-type-name>
        <event-type-vendor>Telcomp2.0</event-type-vendor>
        <event-type-version>1.0</event-type-version>
      </event-type-ref>
    </event>
    <!-- _event_ -->
```

- **Template receiveevent:**

```
    <event event-direction="Receive" initial-event="False" mask-on-attach="False"
last-in-transaction="True">
      <event-name>End_name_Event</event-name>
      <event-type-ref>
        <event-type-name>End_name_Event</event-type-name>
        <event-type-vendor>Telcomp2.0</event-type-vendor>
        <event-type-version>1.0</event-type-version>
      </event-type-ref>
    </event>
    <!-- _event_ -->
```

- **Template httpevent:**

```
    <event event-direction="Receive" initial-event="True">
```

```
      <event-name>GET</event-name>
      <event-type-ref>
         <event-type-
name>net.java.slee.resource.http.events.incoming.request.GET</event-type-name>
         <event-type-vendor>net.java.slee</event-type-vendor>
         <event-type-version>1.0</event-type-version>
      </event-type-ref>
      <initial-event-select variable="ActivityContext"/>
   </event>
```

- **Template servicestart:**

```
   <event event-direction="Receive" initial-event="True" mask-on-attach="False" last-
in-transaction="True">
     <event-name>ServiceStartedEvent</event-name>
     <event-type-ref>
       <event-type-name>javax.slee.serviceactivity.ServiceStartedEvent</event-type-
name>
       <event-type-vendor>javax.slee</event-type-vendor>
       <event-type-version>1.1</event-type-version>
     </event-type-ref>
     <initial-event-select variable="ActivityContext" />
   </event>
   <!-- _event_ -->
```

- **Template httpRA:**

```
<resource-adaptor-type-binding>
   <resource-adaptor-type-ref>
      <resource-adaptor-type-name>HttpServletResourceAdaptorType</resource-
adaptor-type-name>
      <resource-adaptor-type-vendor>org.mobicents</resource-adaptor-type-vendor>
      <resource-adaptor-type-version>1.0</resource-adaptor-type-version>
   </resource-adaptor-type-ref>
   <activity-context-interface-factory-
name>slee/resources/mobicents/httpservlet/acifactory</activity-context-interface-
factory-name>
   <resource-adaptor-entity-binding>
      <resource-adaptor-object-
name>slee/resources/mobicents/httpservlet/sbbrainterface</resource-adaptor-object-
name>
      <resource-adaptor-entity-link>HttpServletRA</resource-adaptor-entity-link>
   </resource-adaptor-entity-binding>
</resource-adaptor-type-binding>
```

- **Template service:**

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE service-xml PUBLIC "-//Sun Microsystems, Inc.//DTD JAIN SLEE
Service 1.1//EN"
                    "http://java.sun.com/dtd/slee-service_1_1.dtd">
<service-xml>
   <service>
      <description/>
      <service-name>_name_</service-name>
      <service-vendor>David Ramirez</service-vendor>
      <service-version>1.0</service-version>
      <root-sbb>
         <description/>
         <sbb-name>_name_</sbb-name>
         <sbb-vendor>David Ramirez</sbb-vendor>
         <sbb-version>1.0</sbb-version>
      </root-sbb>
      <default-priority>0</default-priority>
   </service>
</service-xml>
```

- **Template du:**

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE deployable-unit PUBLIC "-//Sun Microsystems, Inc.//DTD JAIN SLEE
Deployable Unit 1.1//EN"
                      "http://java.sun.com/dtd/slee-deployable-unit_1_1.dtd">
<deployable-unit>
   <jar>jars/_name_-sbb.jar</jar>
   <service-xml>src/service/_name_-service.xml</service-xml>
</deployable-unit>
```

Anexo F

Código fuente de un servicio

En este anexo es presentado el código fuente del servicio LinkedInJobNotificator, este código fue obtenido utilizando el mecanismo de orquestación automática propuesto. A continuación son presentados los archivos más relevantes como lo son el SBB.java, el descriptor XML del SBB, y algunos eventos utilizados.

F.1. Código SBB.java:

Este es el archivo más importante ya que contiene toda la lógica de la orquestación generada.

```
package sbb;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.slee.*;
import javax.slee.nullactivity.NullActivity;
import javax.slee.nullactivity.NullActivityContextInterfaceFactory;
import javax.slee.nullactivity.NullActivityFactory;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import javax.slee.resource.ActivityAlreadyExistsException;
import javax.slee.resource.StartActivityException;

import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.io.PrintWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.ArrayList;
```

```
import net.java.slee.resource.http.HttpServletRaActivityContextInterfaceFactory;
import net.java.slee.resource.http.HttpServletRaSbbInterface;
import net.java.slee.resource.http.HttpSessionActivity;

import org.telcomp.events.StartWSInvocatorEvent;
import org.telcomp.events.EndWSInvocatorEvent;

import org.telcomp.events.StartGetDataTelcoServiceEvent;
import org.telcomp.events.EndGetDataTelcoServiceEvent;


public abstract class LinkedInNotificator2Sbb implements javax.slee.Sbb {

private NullActivityFactory nullActivityFactory;
private NullActivityContextInterfaceFactory nullACIFactory;
private SbbContext sbbContext;
private ActivityContextInterface Activity_00;


private HttpServletRaSbbInterface httpServletRaSbbInterface;
private HttpServletRaActivityContextInterfaceFactory
httpServletRaActivityContextInterfaceFactory;
private ActivityContextInterface httpACI;

private PrintWriter w;
private HttpServletResponse response;
private boolean finishService;

public void setSbbContext(SbbContext context) {
        this.sbbContext = context;
   //_dataModule_
        try {
                Context ctx = (Context) new InitialContext().lookup("java:comp/env");
                nullActivityFactory = (NullActivityFactory)
ctx.lookup("slee/nullactivity/factory");
                nullACIFactory = (NullActivityContextInterfaceFactory)
ctx.lookup("slee/nullactivity/activitycontextinterfacefactory");
             httpServletRaSbbInterface = (HttpServletRaSbbInterface)
ctx.lookup("slee/resources/mobicents/httpservlet/sbbrainterface");
                httpServletRaActivityContextInterfaceFactory =
(HttpServletRaActivityContextInterfaceFactory)
ctx.lookup("slee/resources/mobicents/httpservlet/acifactory");
       } catch (NamingException e) {
                e.printStackTrace();
        }
```

```
}

private void createNullActivityACI() {
        NullActivity nullActivity_00 = this.nullActivityFactory.createNullActivity();
Activity_00 = this.nullACIFactory.getActivityContextInterface(nullActivity_00);
Activity_00.attach(this.sbbContext.getSbbLocalObject());

}

public void unsetSbbContext() {
        this.sbbContext = null;
}
public void sbbCreate() throws javax.slee.CreateException {}
public void sbbPostCreate() throws javax.slee.CreateException {}
public void sbbActivate() {}
public void sbbPassivate() {}
public void sbbRemove() {}
public void sbbLoad() {}
public void sbbStore() {}
public void sbbExceptionThrown(Exception exception, Object event,
ActivityContextInterface activity) {}
public void sbbRolledBack(RolledBackContext context) {}
protected SbbContext getSbbContext() {
        return sbbContext;
}

public abstract void fireStartWSInvocatorEvent (StartWSInvocatorEvent event,
ActivityContextInterface aci, Address address);

public void onEndWSInvocatorEvent (EndWSInvocatorEvent event,
ActivityContextInterface aci){
   System.out.println("<<<----  Recibe evento Web Service");

   this.dataMapping();
   this.controlFlow();
   this.triggerEvent();
}

private boolean endSTART = false;
private boolean endUser_Data = false;
private boolean endData_Module = false;
private boolean endEND = false;

private String user_4;
private String identification_1;
```

```
public abstract void fireStartGetDataTelcoServiceEvent
(StartGetDataTelcoServiceEvent event, ActivityContextInterface aci, Address
address);

private boolean startGetDataTelcoService_01 = false;
private boolean endGetDataTelcoService_01 = false;
private boolean startGetDataTelcoService_02 = false;
private boolean endGetDataTelcoService_02 = false;

private void dataMapping() {
   identification_1 = user_4;

}

public void onServiceStartedEvent ( javax.slee.serviceactivity.ServiceStartedEvent
event, ActivityContextInterface aci) {
        System.out.println("!!!!!!!!!!! Se activa el servicio Compuesto:
"+this.getSbbContext().getSbb().getName());
      aci.detach(this.sbbContext.getSbbLocalObject());
}

public void onGET(net.java.slee.resource.http.events.HttpServletRequestEvent event,
ActivityContextInterface aci) {
        aci.detach(sbbContext.getSbbLocalObject());
        //estas 3 lineas y el IF deben ir en todo SBB que tenga HTTP Servlet
        HttpServletRequest request = event.getRequest();
        String uri = request.getRequestURI().substring(11);
        uri = uri.substring(0, uri.indexOf("/")).replaceAll("%20", " ");

      if (uri.equals(this.getSbbContext().getSbb().getName())) {
              //dentro de este If se pone lo que debe hacer con el evento

              System.out.println("Entra al SBB:
"+this.getSbbContext().getSbb().getName());
              int n = sbbContext.getSbbLocalObject().hashCode();
              System.out.println("!!!!!!!!!!!!!!!! Id objeto sbb: "+n);

              System.out.println("Entra al GET !!!!!!!!!!!!!!!!!!!");

              response = event.getResponse();
              try {
                      w = response.getWriter();
                      w.print("<script> window.history.back(); </script>");
                      w.flush();
                      response.flushBuffer();
```

```
                    System.out.println("HttpServletRAExampleSbb: GET received
and OK! response sent.");
                } catch (Exception e) {
                    System.out.println("no se pudo");
                }
          endUser_Data = true;
          endData_Module = true;
          endSTART = true;
          this.createNullActivityACI();
       user_4 = event.getRequest().getParameter("user");

       this.dataMapping();
       this.controlFlow();
       this.triggerEvent();
         }
}

public void onEndGetDataTelcoServiceEvent(EndGetDataTelcoServiceEvent event,
ActivityContextInterface aci) {
        System.out.println("<<<---- Recibe evento GetDataTelcoService !!!!! ");

       //_data_
       if(aci.equals(Activity_00)){
   endGetDataTelcoService_01 = true;

}
if(aci.equals(Activity_00)){
   endGetDataTelcoService_02 = true;
}
       this.dataMapping();
       this.controlFlow();
       this.triggerEvent();
}

private void triggerEvent(){
if ((endSTART == true) && (endUser_Data == true) && (endData_Module == true)) {
endSTART = false;
endUser_Data = false;
endData_Module = false;
}
if (startGetDataTelcoService_01 == true) {
startGetDataTelcoService_01 = false;
HashMap<String, Object> operationInputs = new HashMap<String, Object>();
operationInputs.put("identification", (String) identification_1);
```

```
StartGetDataTelcoServiceEvent event = new
StartGetDataTelcoServiceEvent(operationInputs);

this.fireStartGetDataTelcoServiceEvent(event, Activity_00, null);

System.out.println("---->>> Dispara el evento startGetDataTelcoService_01");
}
if (startGetDataTelcoService_02 == true) {
startGetDataTelcoService_02 = false;
HashMap<String, Object> operationInputs = new HashMap<String, Object>();

StartGetDataTelcoServiceEvent event = new
StartGetDataTelcoServiceEvent(operationInputs);

this.fireStartGetDataTelcoServiceEvent(event, Activity_00, null);

System.out.println("---->>> Dispara el evento startGetDataTelcoService_02");
}

if (endEND){
endEND = false;
Activity_00.detach(this.sbbContext.getSbbLocalObject());

System.out.println("<< EXIT >> detach the activities....");
}
}

private void controlFlow(){
   if((endSTART == true) && (endUser_Data == true) && (endData_Module == true))
{
      startGetDataTelcoService_01 = true;
   }
   if(endGetDataTelcoService_01 == true) {
      endGetDataTelcoService_01 = false;
      startGetDataTelcoService_02 = true;
      System.out.println(" << Patron >> paso por el Sequence");
   }
   if(endGetDataTelcoService_02 == true) {
      endGetDataTelcoService_02 = false;
      endEND = true;
      System.out.println(" << Patron >> paso por el Sequence");
   }
}
}
```

F.2. Código SBB.xml:

Este es el archivo que describe el contenido del SBB, como sus eventos de entrada y salida, RA, y librerías.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE sbb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD JAIN SLEE SBB 1.1//EN"
                  "http://java.sun.com/dtd/slee-sbb-jar_1_1.dtd">
<sbb-jar>
 <sbb>
   <description />
   <sbb-name>LinkedInNotificator2</sbb-name>
   <sbb-vendor>David Ramirez</sbb-vendor>
   <sbb-version>1.0</sbb-version>
   <sbb-classes>
     <sbb-abstract-class reentrant="False">
      <sbb-abstract-class-name>sbb.LinkedInNotificator2Sbb</sbb-abstract-class-name>
     </sbb-abstract-class>
   </sbb-classes>
      <event event-direction="Receive" initial-event="True" mask-on-attach="False" last-in-transaction="True">
     <event-name>ServiceStartedEvent</event-name>
     <event-type-ref>
      <event-type-name>javax.slee.serviceactivity.ServiceStartedEvent</event-type-name>
       <event-type-vendor>javax.slee</event-type-vendor>
       <event-type-version>1.1</event-type-version>
     </event-type-ref>
     <initial-event-select variable="ActivityContext" />
   </event>
      <event event-direction="Fire" initial-event="False" mask-on-attach="False" last-in-transaction="True">
     <event-name>StartGetDataTelcoServiceEvent</event-name>
     <event-type-ref>
       <event-type-name>StartGetDataTelcoServiceEvent</event-type-name>
       <event-type-vendor>Telcomp2.0</event-type-vendor>
       <event-type-version>1.0</event-type-version>
     </event-type-ref>
   </event>
      <event event-direction="Receive" initial-event="False" mask-on-attach="False" last-in-transaction="True">
     <event-name>EndGetDataTelcoServiceEvent</event-name>
```

```
      <event-type-ref>
        <event-type-name>EndGetDataTelcoServiceEvent</event-type-name>
        <event-type-vendor>Telcomp2.0</event-type-vendor>
        <event-type-version>1.0</event-type-version>
      </event-type-ref>
    </event>
      <event event-direction="Fire" initial-event="False" mask-on-attach="False" last-
in-transaction="True">
      <event-name>StartWSInvocatorEvent</event-name>
      <event-type-ref>
        <event-type-name>StartWSInvocatorEvent</event-type-name>
        <event-type-vendor>Telcomp2.0</event-type-vendor>
        <event-type-version>1.0</event-type-version>
      </event-type-ref>
    </event>
      <event event-direction="Receive" initial-event="False" mask-on-attach="False"
last-in-transaction="True">
      <event-name>EndWSInvocatorEvent</event-name>
      <event-type-ref>
        <event-type-name>EndWSInvocatorEvent</event-type-name>
        <event-type-vendor>Telcomp2.0</event-type-vendor>
        <event-type-version>1.0</event-type-version>
      </event-type-ref>
    </event>
      <event event-direction="Receive" initial-event="True">
      <event-name>GET</event-name>
      <event-type-ref>
        <event-type-
name>net.java.slee.resource.http.events.incoming.request.GET</event-type-name>
          <event-type-vendor>net.java.slee</event-type-vendor>
          <event-type-version>1.0</event-type-version>
      </event-type-ref>
      <initial-event-select variable="ActivityContext"/>
    </event>
    <resource-adaptor-type-binding>
    <resource-adaptor-type-ref>
      <resource-adaptor-type-name>HttpServletResourceAdaptorType</resource-
adaptor-type-name>
        <resource-adaptor-type-vendor>org.mobicents</resource-adaptor-type-vendor>
        <resource-adaptor-type-version>1.0</resource-adaptor-type-version>
    </resource-adaptor-type-ref>
   <activity-context-interface-factory-
name>slee/resources/mobicents/httpservlet/acifactory</activity-context-interface-
factory-name>
    <resource-adaptor-entity-binding>
```

```
      <resource-adaptor-object-
name>slee/resources/mobicents/httpservlet/sbbrainterface</resource-adaptor-object-
name>
       <resource-adaptor-entity-link>HttpServletRA</resource-adaptor-entity-link>
   </resource-adaptor-entity-binding>
</resource-adaptor-type-binding>
  </sbb>
</sbb-jar>
```

F.3. Código evento.java:

Este es el archivo que describe uno de los eventos asociados al SBB. Si bien son bastantes eventos, todos tienen un formato genérico y lo que cambia son los datos, por lo tanto solo se presenta un evento.

```
package event;

import java.util.Random;
import java.io.Serializable;

public final class StartSendSMSEvent implements Serializable {
        /**
         *
         */
        private static final long serialVersionUID = 1L;
        private final long id;
        private String message;
        private String number;

        public StartSendSMSEvent(String  message, String number) {
                id = new Random().nextLong()  ^ System.currentTimeMillis();
                this.message = message;
                this.number  = number;

        }
        public String getMessage(){
                return this.message;
        }
        public String getNumber(){
                return this.number;
        }
        public boolean equals(Object o) {
                if (o == this) return true;
```

```
                if (o == null) return false;
                return (o instanceof StartSendSMSEvent) &&
((StartSendSMSEvent)o).id == id;
        }
        public int hashCode() {
                return (int) id;
        }
        public String toString() {
                return "StartSendSMSEvent[" + hashCode() + "]";
        }
}
```