UNIVERSIDAD TECNOLÓGICA NACIONAL

Facultad Regional Santa Fe

DOCTORADO EN INGENIERÍA

MENCIÓN EN INGENIERÍA EN SISTEMAS DE INFORMACIÓN

Tesis Doctoral

"VERIFICACIÓN Y ALINEACIÓN DE PROCESOS DE NEGOCIO COLABORATIVOS"

Tesista: Ing. Jorge Roa

Director: Dr. Pablo Villarreal

Co-Director: Dr. Omar Chiotti

Santa Fe, Argentina Febrero 2014

UNIVERSIDAD TECNOLÓGICA NACIONAL

Facultad Regional Santa Fe

Comisión de Posgrado

Se presenta esta Tesis en cumplimiento de los requisitos exigidos por la Universidad Tecnológica Nacional para la obtención del grado académico de Doctor en Ingeniería, mención Sistemas de Información

"VERIFICACIÓN Y ALINEACIÓN DE PROCESOS DE NEGOCIO COLABORATIVOS"

POR

ING. JORGE ROA

DIRECTOR: DR. PABLO VILLARREAL

CO-DIRECTOR: DR. OMAR CHIOTTI

JURADOS DE TESIS

DR. ALDO VECCHIETTI

DR. ALEJANDRO ZUNINO

DR. NAZARENO AGUIRRE

SANTA FE, ARGENTINA FEBRERO 2014

Índice General

Índice de Tablas

Índice de Figuras

Índice de Algoritmos

Resumen

Las colaboraciones inter-organizacionales permiten nuevas formas de gestión basadas en la cooperación. Los sistemas de información que brindan soporte a la gestión de estas acciones de colaboración requieren definir modelos y especificaciones de procesos de negocio colaborativos (PNCs) que representan el comportamiento explícito de la colaboración.

En base a técnicas formales y a los conceptos del desarrollo dirigido por modelos, en esta tesis se proponen métodos y herramientas que permiten determinar el correcto comportamiento de los modelos de PNCs y, a partir de dichos modelos, generar especificaciones de PNCs cuyo comportamiento está alineado con el de los modelos.

Se propone el lenguaje formal Redes de Interacción Global (GI-Nets) y un método de transformación para definir modelos formales de PNCs con GI-Nets a partir de modelos conceptuales de PNCs.

Se definen dos métodos de verificación que permiten determinar si un modelo de PNC satisface un conjunto de propiedades que determinan su correcto comportamiento. El primer método, basado en GI-Nets, propone la propiedad de *Solidez de Interacción Global* de una GI-Net como principal criterio de verificación. Este método permite detectar el lugar específico donde existe un bloqueo en un modelo de PNC. El segundo método, basado en anti-patrones de comportamiento, provee un enfoque para especificar en forma sistemática los anti-patrones de cualquier lenguaje de PNCs. Este método detecta el lugar de un bloqueo y determina el conjunto de elementos que lo produce. Ambos métodos pueden ser utilizados con cualquier lenguaje de PNCs y dan soporte a la verificación de PNCs con constructores complejos.

Se define un método formal de transformación de modelos que permite generar en forma automática una especificación de PNC cuyo comportamiento esté alineado con el definido en el modelo conceptual del PNC a partir del cual fue generada. Esto implica que el comportamiento de la especificación será correcto si el comportamiento del modelo del PNC es correcto, y viceversa. El método se utiliza para generar especificaciones de PNCs basadas en tecnologías de servicios Web a partir de modelos conceptuales de PNCs.

Finalmente, se presentan las herramientas desarrolladas para la formalización, verificación y transformación de modelos y especificaciones de PNCs, y se utilizan las mismas para evaluar y validar los métodos propuestos en la tesis.

Prologo

Los modelos de gestión y de negocio basados en cooperación y colaboración han despertado el interés de las organizaciones. Estos modelos implican el desarrollo de colaboraciones inter-organizacionales gestionadas en forma electrónica a través de sistemas de información. Las interacciones entre las organizaciones y sus roles, desde un punto de vista global, dan lugar a *procesos de negocio colaborativos* (PNCs).

El desarrollo de sistemas de información para dar soporte a colaboraciones inter-organizacionales requiere *modelar* los PNCs a nivel de negocio, y luego *especificarlos a* nivel tecnológico. Estas especificaciones son interpretadas por sistemas de información inter-organizacionales orientados a procesos, que permiten automatizar y ejecutar los PNCs.

El comportamiento de un PNC se expresa en un modelo que define el flujo de control de las interacciones entre las organizaciones. Una definición incorrecta del mismo puede afectar los procesos internos de las organizaciones y propagar errores más allá de sus fronteras, generar desconfianza entre las organizaciones, afectar la eficiencia de la colaboración, impedir el logro de las metas comunes fijadas en la colaboración, e incrementar los costos y el tiempo de desarrollo. Por lo cual, es necesario poder determinar que el comportamiento de estos modelos sea correcto.

La verificación del comportamiento de un modelo de PNC permite determinar la presencia de problemas, tales como bloqueos o fallas de sincronización. Las propuestas de verificación existentes, no permiten verificar modelos de PNC que presentan constructores complejos de flujo de control. Algunas propuestas afectan la autonomía de las organizaciones, mientras que otras sólo permiten verificar especificaciones de PNCs dependientes de la tecnología. La mayoría de las propuestas de verificación están ligadas a los lenguajes para los que fueron diseñadas, lo que impide o dificulta su reuso con otros lenguajes y a lo largo del proceso de desarrollo. Si bien se comprobó que el uso de modelos estructurados de procesos mejora el rendimiento de la verificación, las técnicas actúales de verificación basadas en modelos estructurados no dan soporte a constructores complejos. Los métodos de verificación actuales también se ven afectados por la heterogeneidad y

complejidad de los lenguajes de PNCs, ya que no existe un formalismo que permita formalizar todos los constructores de estos lenguajes.

Los métodos de verificación de modelos de PNCs deben satisfacer los criterios de completitud y rendimiento. La *completitud* refiere a que el método debería poder verificar todo proceso definido con un dado lenguaje, lo que implica dar soporte a cada constructor de dicho lenguaje. El *rendimiento* refiere al tiempo que tarda en realizar la verificación.

Cuando se utiliza un enfoque de desarrollo dirigido por modelos, el modelo de un PNC evoluciona hacia una especificación a través de diferentes fases de desarrollo. En cada fase se utilizan diversos lenguajes que representan el modelo del PNC con distintos niveles de abstracción. Mediante la aplicación de técnicas de transformación de modelos, es posible derivar automáticamente una especificación tecnología del PNC a partir de un modelo del PNC independiente de la tecnología. No obstante, esto no es suficiente para garantizar que el comportamiento de la especificación tecnológica sea el correcto.

Los métodos y herramientas que utiliza un enfoque de desarrollo dirigido por modelos deben garantizar que el comportamiento de la especificación generada automáticamente esté alineado con el del modelo, es decir, que ambos tengan el mismo comportamiento.

Si bien existen distintas propuestas para determinar alineación de comportamiento entre modelos de procesos, éstas se basan en técnicas que pueden ser computacionalmente exponenciales, y no se enfocan en generar especificaciones de procesos a partir de modelos. Además, si bien la verificación y la alineación de comportamiento están relacionadas, los enfoques actuales están orientados a resolver estos problemas de manera separada.

El objetivo principal de esta tesis es desarrollar métodos y herramientas que permitan a los analistas de negocios y diseñadores/desarrolladores de sistemas de información verificar el comportamiento del modelo de un PNC y generar en forma automática una especificación tecnológica cuyo comportamiento esté alineado con el del modelo.

Para alcanzar estos objetivos se proponen métodos y herramientas basados en técnicas formales de procesos y sistemas concurrentes, junto con la aplicación de los conceptos del desarrollo dirigido por modelos.

Los principales aportes de la tesis son:

Un lenguaje formal redes de Interacción Global (GI-Nets), el cual es una extensión de las redes de Petri Jerárquicas y Coloreadas, que permite formalizar PNCs definidos con diferentes lenguajes de modelado o especificación.

Un método de transformación para definir modelos formales con GI-Nets a partir de modelos conceptuales de PNCs.

Un método de verificación basado en GI-Nets que da soporte a la verificación de procesos con constructores de sincronización avanzada y manejo de excepciones. Se define la propiedad de Solidez de Interacción Global de una GI-Net, que permite determinar si un modelo GI-Net que representa un PNC está libre de bloqueos. A diferencia de métodos de verificación tradicionales basados en redes de Petri, este método permite detectar el lugar específico en el modelo donde existe un bloqueo, lo que facilita la corrección del mismo.

Un método de verificación basado en anti-patrones de comportamiento para verificar modelos y especificaciones de PNCs. Se presenta un enfoque para especificar en forma sistemática los anti-patrones para un dado lenguaje de PNCs, los cuales son utilizados por el método para detectar errores en el comportamiento de los modelos. El método indica el lugar específico donde se produce un bloqueo, y el conjunto exacto de elementos que lo producen, lo que facilita la corrección de este tipo de errores.

La principal ventaja de los métodos de verificación propuestos es que logran alcanzar el criterio de completitud, ya que posibilitan verificar modelos con constructores avanzados y definidos con cualquier lenguaje de modelado o especificación de PNCs. El método basado en anti-patrones mejora el criterio de rendimiento.

Un método formal de transformación de modelos que permite generar especificaciones de PNCs a partir de modelos conceptuales de PNCs. El uso de modelos y especificaciones de PNCs formalizados con GI-Nets permite garantizar que el comportamiento de la especificación generada esté alineado con el comportamiento definido en el modelo. El método se utiliza para generar especificaciones de PNCs basadas en tecnologías de servicios Web a partir de modelos conceptuales de PNCs.

Dos herramientas para la formalización, verificación y transformación de modelos y especificaciones de PNCs. Las mismas permitieron evaluar y validar los métodos propuestos en la tesis.

Los resultados de la tesis dieron lugar a las siguientes publicaciones:

- Roa, J., Chiotti, O., Villarreal, P. Behavior Alignment and Control Flow Verification of Process and Service Choreographies. Journal of Universal Computer Science, vol. 18, no. 17, pp. 2383-2406, 2012. ISI Impact Factor 2012=0,762.
- Roa, J., Chiotti, O., Villarreal, P. A Survey of Verification Methods for the Behavior of Cross-Organizational Business Processes. ACM Computing Surveys. ISI Impact Factor 2012= 3,543. En revisión.
- Roa, J., Chiotti, O., Villarreal, P. Verification of Structured Processes: A Method Based on an Unsoundness Profile. 42 Jornadas Argentinas de Informática e Investigación Operativa (JAIIO). Simposio Argentino de Ingeniería de Software (ASSE) 2013.
- Roa, J., Chiotti, O. Villarreal, P. A Verification Method for Collaborative Business Processes. Lecture Notes in Business Information Processing (LNBIP), vol. 099, pp. 293-305, ISSN: 1865-1348, Springer-Verlag Berlin Heidelberg, 2012.
- Lazarte, I., Tello-Leal, E., Roa, J., Chiotti, O., Villarreal, P. Model-Driven Development Methodology for B2B Collaborations. En Proceedings of the 2010 14th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW 2010). ISBN: 978-0-7695-4164-8.
- Villarreal, P., Lazarte, I., Roa, J., Chiotti, O. A Modeling Approach for Collaborative Business Processes based on the UP-ColBPIP Language. Lecture Notes in Business Information Processing (LNBIP), vol. 43, pp. 318-329, ISSN: 1865-1348, Springer-Verlag Berlin Heidelberg, 2010.
- Roa, J. A Methodology for the Design, Verification, and Validation of Business Processes in B2B Collaborations. Proceedings of the ER 2009 PhD Colloquium (ERPhD-2009) affiliated to the 28th International Conference on Conceptual Modeling (ER-2009), Gramado, Brazil, November 9, 2009, ISSN: 1613-0073, vol. 597, José Palazzo Moreira de Oliveira.

Reconocimientos

En primer lugar, quiero agradecer al Dr. Pablo Villarreal, mi director, por su confianza, apoyo, y esfuerzo durante todos estos años, y por permitirme desarrollar libremente y guiarme constantemente en mis actividades de investigación, primero como becario de grado, y luego, como becario doctoral.

También, quiero expresar mi total gratitud al Dr. Omar Chiotti, mi co-director, por las discusiones enriquecedoras y su guía y apoyo permanente en mis actividades de investigación.

A la Universidad Tecnológica Nacional y al Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET) por brindarme el soporte económico necesario para llevar a cabo la realización de esta tesis, y a la Facultad Regional Santa Fe por el espacio y recursos brindados.

A todos los integrantes del CIDISI, por permitirme formar parte de un excelente grupo de personas que día a día conforman un cálido y placentero ambiente de trabajo.

A Geor y Milagros por su ayuda incondicional e invaluables consejos.

A Esteban, Neri, y Maxi por colaborar con las implementaciones fruto de esta tesis.

A Edgar, Ivanna, y Mariano por tantos recuerdos imborrables, especialmente esas reuniones tan "productivas" en Laguna Picada.

A Hernán, Sergio, Gustavo, Lucas, Poroto, y Gonzalo por tantas noches "perdidas".

A Hernán y Pame por tantos momentos inolvidables, especialmente esas escapadas los fines de semana a Carlos Paz.

También quiero agradecer a Chili, Titi, Sol, Benito, Lili, Javi, Tebi, Pame y Mami Cuca: Gracias familia! A mi hermanita Titi, compañera incondicional y vía de escape para ver al Tate. Y en especial a mis padres, que gracias al trabajo y esfuerzo diario siempre dieron todo para que yo pueda cumplir mis sueños. Mil Gracias!

Finalmente, quiero agradecer muy especialmente a Vero, mi amor, *sin vos nada tendría sentido*.

1 Introducción

En este capítulo se describe el contexto en el que se enmarca la tesis (Sección 1.1), detallando los conceptos referidos al dominio de las colaboraciones inter-organizacionales y los procesos colaborativos. Se presentan las motivaciones y los problemas a resolver (Sección 1.2). Se definen las hipótesis y los objetivos (Sección 1.3) y se presentan las principales contribuciones (Sección 1.4). Por último, se describe la organización de la tesis (Sección 1.5).

1.1. Contexto

Con el propósito de adaptarse a la dinámica de los mercados, las organizaciones están integrando sus procesos de negocio mediante acciones de colaboración con sus socios de negocio (Liu, Li & Zhao 2009). La habilidad de una organización para describir, estandarizar y adaptar la forma de reaccionar a ciertos tipos de eventos, e interactuar con proveedores, socios, competidores y clientes, es un aspecto clave para mantener la competitividad (Bauer, Roser & Müller 2005). Esto ha tenido impacto positivo en distintas áreas, tales como: la gestión integrada de una cadena de suministros, la cual se basa en acciones de colaboración entre socios de negocio que forman parte de la misma (Sanders 2007), o la gestión integrada de servicios electrónicos de salud (e-healthcare), la cual se basa en acciones de colaboración entre prestatarios de servicios complementarios de salud (e-healthcare) (Tello-Leal, Chiotti & Villarreal 2012).

En los últimos años han surgido modelos de negocio y de gestión organizacional denominados *modelos de colaboración*, los cuales proponen nuevas formas de gestión basadas en acciones de cooperación y colaboración entre organizaciones (Villarreal, Salomone & Chiotti 2007). Estos modelos son implementados por medio de las denominadas colaboraciones inter-organizacionales, las cuales son conducidas en forma electrónica mediante sistemas de información.

La *colaboración inter-organizacional* implica llevar a cabo acciones de colaboración y cooperación durante un período de tiempo para alcanzar objetivos en

común, coordinar acciones, intercambiar información, y definir y ejecutar procesos de negocio inter-organizacionales (Villarreal, Salomone & Chiotti 2006; Camarinha-Matos, Nathalie Galeano & Molina 2009; Li et al. 2010). Esta forma de gestión produce importantes beneficios para las organizaciones participantes tales como: disminución de costos, flexibilidad para adaptarse a los cambios y manejo eficiente de información en tiempo real, lo cual permite mantener o mejorar la competitividad en el mercado global (Sanders 2007; Li et al. 2010; Tello-Leal 2012).

Implementar una colaboración inter-organizacional implica que organizaciones heterogéneas y autónomas puedan integrar sus procesos de negocio tanto a nivel de negocio (u organizacional) como a nivel tecnológico (Villarreal, Salomone & Chiotti 2007; Weske 2007; Lippe, Greiner & Barros 2005). A *nivel de negocio*, las organizaciones definen y acuerdan el comportamiento de la colaboración, para lo cual se enfocan en el diseño de los procesos de negocio inter-organizacionales. A través de dichos procesos acuerdan tomar decisiones en conjunto, coordinar acciones, e intercambiar información mediante los denominados sistemas de información inter-organizacionales (Villarreal, Salomone & Chiotti 2007; Roser & Bauer 2005). El comportamiento explícito de la colaboración se define en los modelos de estos procesos.

A *nivel tecnológico*, las organizaciones se enfocan en la implementación, integración e interoperabilidad de sus sistemas de información inter-organizacionales. Esto requiere definir las especificaciones de las interfaces de los sistemas de información de las organizaciones y las especificaciones de los procesos de negocio en base a una plataforma tecnológica. Estas especificaciones permiten a las organizaciones automatizar y ejecutar los procesos de negocio inter-organizacionales a través de sistemas de información orientados a procesos (Lazarte et al. 2010; Tello-Leal, Chiotti & Villarreal 2012). Un sistema de información orientado a procesos ejecuta y gestiona instancias de procesos a partir de la interpretación de modelos o especificaciones de procesos (Dumas, van der Aalst & ter Hofstede 2005)

El *comportamiento* de un proceso de negocio inter-organizacional refiere a la forma en que las organizaciones van a interactuar en el marco de la colaboración, y puede ser definido desde una perspectiva global o una perspectiva local. En la *perspectiva global* se define el comportamiento público y las responsabilidades de cada organización en términos

de las secuencias de interacciones y las dependencias entre las mismas, sin considerar las actividades internas de las organizaciones (Villarreal 2005; Villarreal, Salomone & Chiotti 2007; Decker, Kopp & Barros 2008; Issarny et al. 2011). La perspectiva global es definida en los denominados procesos de negocio colaborativos. Un *proceso de negocio colaborativo (PNC)*, también llamado coreografía de proceso (Decker, Kopp & Barros 2008; Issarny et al. 2011; Weske 2007)), es un proceso de negocio inter-organizacional que establece una secuencia de interacciones entre organizaciones desde una perspectiva global, para alcanzar un objetivo en común (Villarreal, Salomone & Chiotti 2007; Roser & Bauer 2005).

En la *perspectiva local* se define el comportamiento del proceso de negocio inter-organizacional desde el punto de vista de las actividades que una organización debe ejecutar para cumplir el rol que desempeña en un PNC (Su et al. 2008). En esta perspectiva, el comportamiento puede ser público o privado. El *comportamiento público* define el comportamiento visible de una organización en términos de las actividades que dan soporte al envío y recepción de los mensajes intercambiados con otras organizaciones. Puede ser definido por medio de los denominados procesos de interfaz (Villarreal, Salomone & Chiotti 2007; Weske 2007; Lazarte et al. 2010), también conocidos como procesos públicos (OMG-BPMN 2011), workflows públicos (van der Aalst et al. 2010; Van Der Aalst 2003), o procesos abstractos (OASIS 2007). El *comportamiento privado* define la lógica de negocio interna requerida para dar soporte al rol que desempeña una organización en un PNC. Este comportamiento se define por medio de los denominados procesos de integración, también conocidos como procesos privados (OMG-BPMN 2011), procesos ejecutables (OASIS 2007), o modelos de interacción (Decker 2009).

El desarrollo de sistemas de información para colaboraciones inter-organizacionales siguiendo un enfoque "Top-Down" requiere definir los PNCs de dos maneras. En primer lugar, requiere diseñar el *modelo conceptual* de los PNCs. Dicho modelo describe los aspectos de la colaboración en un alto nivel de abstracción y forma parte de la *solución de negocio* (Villarreal, Salomone & Chiotti 2007; Lazarte et al. 2010). En segundo lugar, requiere definir las *especificaciones* de los procesos y las interfaces de los sistemas inter-organizacionales. Dichas especificaciones conforman la *solución tecnológica* (Villarreal 2005; Bauer, Roser & Müller 2005; Villarreal, Salomone & Chiotti 2007; Stuit & Szirbik

2009; Lazarte et al. 2010). Las especificaciones de los PNCs y/o de los procesos de integración son interpretadas por los sistemas de información de las organizaciones para gestionar la colaboración inter-organizacional y ejecutar los PNCs.

En (Lazarte et al. 2010), se propuso una metodología "Top-down", basada en el desarrollo dirigido por modelos (Model-Driven Development) (Selic 2003), para el desarrollo de sistemas de información orientados a procesos para colaboraciones inter-organizacionales. El desarrollo dirigido por modelos se sustenta en dos premisas básicas: los modelos son los principales productos del desarrollo de software, en lugar del código; y el código (ej: especificaciones de procesos) puede ser generado automáticamente a partir de sus correspondientes modelos, aplicando transformaciones de modelos. En base a estas premisas, la metodología propuesta por (Lazarte et al. 2010) define a los modelos conceptuales de PNCs como uno de los principales artefactos de desarrollo para generar automáticamente las especificaciones de sistemas y procesos requeridas por las organizaciones para ejecutar PNCs. De este modo, los modelos no sólo son utilizados como medio de documentación sino también para construir el producto final. La metodología define etapas de diseño del modelo conceptual, seguida de otras enfocadas en generar automáticamente las especificaciones de los PNCs, posibilitando que analistas de negocio y diseñadores de sistemas definan dichos modelos utilizando conceptos propios del dominio de la colaboración inter-organizacional con independencia de las tecnologías de implementación.

Existen distintos lenguajes propuestos por consorcios de organizaciones y empresas que pueden ser utilizados para definir modelos o especificaciones de los PNCs, tales como BPMN (OMG-BPMN 2011), WS-CDL (W3C-WSCDL 2005), BPEL (OASIS 2007), y ebXML (ebXML 2002). En (Mili et al. 2010) se presenta un resumen de los mismos. También existen lenguajes orientados a dar soporte a requerimientos específicos del dominio de las colaboraciones inter-organizacionales, tales como UP-ColBPIP (Villarreal 2005; Villarreal et al. 2010), UMM (Huemer et al. 2008), y Let's Dance (Weske 2007). Los principales requerimientos específicos de este dominio son: representación de la vista global o coreografía de interacciones entre las organizaciones, autonomía de las organizaciones, gestión descentralizada, representación de compromisos, soporte a negociaciones complejas, modelado del contexto de negocio, y gestión de cancelaciones y

excepciones (Medjahed et al. 2003; Goldkuhl 2004; Villarreal, Salomone & Chiotti 2007; Bauer, Roser & Müller 2005; Lippe, Greiner & Barros 2005; Zaha et al. 2006; Huemer et al. 2008; Villarreal et al. 2010; Decker 2009).

Si bien la definición del comportamiento de un PNC implica considerar distintos aspectos, tales como de flujo de control, de datos, y de recursos (Russell et al. 2006), esta tesis se enfoca exclusivamente en el aspecto de flujo de control. El *flujo de control* define el comportamiento de un proceso de negocio mediante constructores (o primitivas) que permiten modelar, en el caso de los PNCs, el orden en que se llevan a cabo las interacciones entre las organizaciones.

Los lenguajes de modelado y especificación de PNCs disponen de distintos tipos de constructores que varían de acuerdo a sus semánticas específicas de comportamiento. En general, disponen de un conjunto de constructores para modelar flujos de control simples, tales como secuencia, bifurcación y sincronización, y exclusión mutua, y para modelar flujos de control complejos, tales como sincronización avanzada, instancias múltiples, cancelaciones y manejo de excepciones. En esta tesis, a los primeros se los denomina *constructores de flujo de control simples (o constructores simples)*, y a los segundos se los denomina *constructores de flujo de control complejos (o constructores complejos).*

1.2. Motivación

La metodología de desarrollo de sistemas para colaboraciones inter-organizacionales propuesta por (Lazarte et al. 2010) expresa que además de métodos de modelado y especificación de PNCs, se requieren métodos y herramientas que den soporte a la verificación del comportamiento de los PNCs. La definición incorrecta del comportamiento de los PNCs puede causar problemas, tales como: (1) afectar los procesos internos de las organizaciones y propagar errores más allá de sus fronteras, (2) generar desconfianza entre las organizaciones, (3) afectar la eficiencia de la colaboración, (4) o impedir el logro de las metas comunes fijadas en la colaboración. Esto puede implicar además un incremento de los costos y del tiempo de desarrollo. Los problemas señalados realzan la importancia de verificar que el comportamiento definido en los modelos y especificaciones de PNCs sea correcto.

La tarea de chequear que un modelo sea correcto se denomina verificación, e implica determinar que el modelo se ajuste a ciertas propiedades (Girault & Valk 2001). En esta tesis, el término *verificación* refiere a la tarea de verificar el comportamiento de un PNC desde el aspecto de flujo de control, lo cual implica chequear que el modelo o la especificación del PNC satisfacen un conjunto de propiedades de comportamiento en relación al flujo de control; por ejemplo, ausencia de bloqueos y/o bucles infinitos y/o caminos inalcanzables.

Existen diversos métodos y lenguajes formales sustentados en modelos matemáticos que permiten verificar modelos en distintas etapas del desarrollo de software (Clarke & Kurshan 1996; Woodcock et al. 2009). Redes de Petri (Murata 1989), Workflow Nets (van der Aalst 1998) (basadas en redes de Petri), y álgebra de procesos(Bergstra & Klop 1984) han sido utilizados para formalizar y verificar modelos de procesos de negocio privados e inter-organizacionales. Los métodos de verificación basados en lenguajes formales son referidos en esta tesis como *métodos formales de verificación*.

Se han propuesto diferentes criterios para evaluar un método de verificación (Clarke & Kurshan 1996). En particular, un método de verificación de PNCs debería satisfacer el criterio de completitud. *Completitud* refiere a que el método debería permitir verificar todos los posibles PNCs que pueden ser definidos con un dado lenguaje, lo que implica dar soporte a cada constructor de dicho lenguaje. Cuanto mayor sea el conjunto de constructores soportados por un método de verificación, más completo es el mismo. Los métodos existentes para verificar PNCs, como los propuestos por (Diaz et al. 2005; Yang et al. 2006; Breugel & Koshkina 2006; Stuit & Szirbik 2009), no dan soporte a constructores complejos. *Por lo tanto, se requieren métodos formales y herramientas de verificación de PNCs que incluyan constructores complejos.*

Otro criterio a considerar en la evaluación de un método de verificación es su rendimiento. El *rendimiento* refiere al tiempo de respuesta, esto es, el tiempo que tarda en llevar a cabo la verificación. Los criterios de completitud y rendimiento se contraponen, ya que, cuanto mayor es el conjunto de constructores complejos a dar soporte, mayor es el tiempo de respuesta, lo cual afecta negativamente el rendimiento, y viceversa. Por ejemplo, si bien un método de verificación de un proceso de negocio basado en el uso de Workflow Nets puede realizar la verificación en tiempo lineal (Fahland et al. 2009), no da soporte a la

verificación de modelos con constructores complejos. Existen otros métodos de verificación de procesos (Dijkman, Dumas & Ouyang 2008; Wynn et al. 2009) que dan soporte a algunos constructores complejos y que podrían ser adaptados al dominio de las colaboraciones inter-organizacionales para verificar PNCs. Sin embargo, estos métodos están basados en el análisis del espacio de estados de los modelos a verificar, lo que puede producir el problema de la explosión del espacio de estados (Valmari 1998), afectando negativamente su rendimiento.

El uso de modelos estructurados de procesos de negocio tiene un impacto positivo en el rendimiento de los métodos de verificación (Vanhatalo, Völzer & Leymann 2007; Polyvyanyy, Weidlich & Weske 2011; Hauser et al. 2008; Vanhatalo, Völzer & Koehler 2009). En un *modelo estructurado de proceso de negocio,* para cada punto de bifurcación/decisión existe un punto de sincronización/unión, formando un bloque que tiene una única entrada y una única salida (Vanhatalo, Völzer & Leymann 2007). El uso de modelos estructurados en un método formal de verificación de PNCs, llevaría a analizar un espacio de estado menor comparado con un modelo no estructurado, lo que podría mejorar el rendimiento del método. Si bien los métodos de verificación existentes que se basan en estructurar los modelos de procesos tienen buen rendimiento, no brindan soporte a la verificación de modelos que presentan constructores complejos.

La aplicación de un método formal de verificación de PNCs implica realizar varios ciclos de (re)diseño y verificación, hasta que se determina que el PNC satisface un conjunto de propiedades de comportamiento. Esto implica generar modelos de PNCs utilizando lenguajes formales. Dichos modelos serán referidos en esta tesis como *modelos formales de PNCs*, los cuales son utilizados para la verificación del PNC. Este ciclo de diseño, generación de modelos formales y verificación, no considera la reutilización de los resultados de la verificación, lo cual implica que deba ser ejecutado cada vez que se desea verificar un PNC. Por lo cual, si la verificación toma más tiempo del esperado, se dificulta completar el ciclo entre diseño y verificación. Incluso, para verificar dos modelos de PNCs iguales, este ciclo debe repetirse para cada uno.

El uso de anti-patrones podría resultar en una posible solución al conflicto entre completitud y rendimiento. Un *anti-patrón de comportamiento* de PNCs es una situación predefinida y bien conocida de una definición deficiente del flujo de control. Puede ser

vista como una combinación de constructores de flujo de control que debería ser evitada durante el modelado de PNCs. Una vez que se descubren y definen los anti-patrones, el proceso de verificación se limita a una técnica de búsqueda de coincidencias de patrones (Hoffmann & O'Donnell 1982). El uso de anti-patrones permitiría reutilizar resultados de los métodos formales de verificación, lo que podría mejorar el rendimiento de los métodos de verificación de PNCs y dar soporte a constructores complejos, mejorando de este modo el balance entre rendimiento y completitud.

El uso de anti-patrones de comportamiento requiere afrontar dos desafíos importantes. En primer lugar, los enfoques que hacen uso de anti-patrones (Koehler & Vanhatalo 2007; Gruhn & Laue 2009; Kühne et al. 2010) si bien pueden mejorar el rendimiento de los métodos formales de verificación, no satisfacen el requerimiento de completitud, ya que actualmente no existen procedimientos ni algoritmos que permitan definir todos los posibles anti-patrones de comportamiento que pueden ser generados con un dado lenguaje de modelado o de especificación de PNCs. En segundo lugar, algunos resultados de verificación de los enfoques mencionados son imprecisos, ya que existen modelos para los cuales presentan resultados erróneos (Laue & Awad 2010). *Por lo tanto, se requieren métodos y herramientas de verificación de PNCs que permitan reutilizar los resultados obtenidos con métodos formales de verificación para mejorar el rendimiento de los mismos sin afectar su completitud.*

Además de los criterios de completitud y rendimiento, otro criterio a considerar es la *reusabilidad* de un método de verificación. Los métodos de verificación de PNCs existentes (Diaz et al. 2005; Yang et al. 2006; Breugel & Koshkina 2006; Stuit & Szirbik 2009) están muy ligados a los lenguajes de modelado o de especificación de procesos para los cuales fueron diseñados, lo que impide o dificulta su reutilización en otros lenguajes. La reutilización de un método de verificación es un aspecto importante en una metodología de desarrollo de sistemas, como la propuesta en (Lazarte et al. 2010) debido a que intervienen distintos lenguajes que modelan el comportamiento de un mismo PNC en diferentes niveles de abstracción. *Por lo tanto, se requieren métodos y herramientas de verificación de PNCs que puedan ser reutilizados en distintos lenguajes y en diferentes etapas de desarrollo de los PNCs.*

Considerando que las principales decisiones de diseño se toman en las primeras etapas del desarrollo, cuando se definen los modelos conceptuales de los PNCs en la solución de negocio, es posible concluir que es en estas etapas donde los errores en la lógica de los procesos deberían detectarse. Esto permitiría minimizar el tiempo y los costos de desarrollo. La detección de errores de comportamiento en especificaciones de PNCs dependientes de la tecnología, tales como especificaciones WS-CDL, puede implicar un incremento sustancial de los costos y del tiempo de desarrollo, debido a que se detectan en etapas tardías del desarrollo, cuando la solución tecnológica ya fue generada.

Si bien las especificaciones de PNCs pueden ser derivadas automáticamente a partir de modelos de PNCs que satisfacen un conjunto de propiedades de comportamiento, la verificación de un modelo de PNC no es suficiente para garantizar que el comportamiento de su correspondiente especificación tecnológica sea correcta. Uno de los problemas más importantes del desarrollo dirigido por modelos es cómo garantizar que el comportamiento del código (en el caso de esta tesis, la especificación de un PNC) está alineado con el comportamiento del modelo a partir del cual fue generado. En esta tesis, *alineación de comportamiento* o simplemente *alineación*, refiere a que la especificación debe tener el mismo comportamiento que el modelo a partir del cual fue generada. Esto implica que el modelo y la especificación satisfacen el mismo comportamiento. En este caso se dice que la solución tecnológica está alineada con la solución de negocio desde el punto de vista del comportamiento.

Los diferentes enfoques propuestos para determinar alineación de comportamiento (también referida como consistencia) entre procesos (Varró & Pataricza 2003; Dijkman et al. 2004; Danylevych, Karastoyanova & Leymann 2010; Weidlich, Dijkman & Weske 2010) están basados en técnicas de "bisimulación", equivalencia de trazado, o chequeo de modelos, las cuales pueden ser computacionalmente costosas, debido a que se basan en la exploración del espacio de estados del comportamiento de los procesos. Por otra parte, si bien verificación y alineación están relacionadas con el comportamiento de los PNCs, los enfoques actuales están orientados a resolver estos problemas de manera separada. *Por lo tanto, se requieren métodos y herramientas para generar especificaciones de PNCs alineadas con los correspondientes modelos a partir de los cuales fueron generadas. Esto*

implica determinar que ambos, modelo y especificación, satisfacen el mismo comportamiento.

1.3. Hipótesis y Objetivos de la Tesis

1.3.1. Hipótesis

Hipótesis principal:

H0. Mediante la aplicación de enfoques de formalización y verificación de modelos estructurados de PNCs, junto con la aplicación de los principios del desarrollo dirigido por modelos, es posible definir un modelo de PNC que satisfaga un conjunto de propiedades de comportamiento y generar, a partir del mismo, una especificación alineada con el modelo.

Hipótesis específicas:

H1: Utilizando un formalismo apropiado, tales como redes de Petri, es posible definir un lenguaje formal de PNCs que de soporte a la formalización de constructores de flujo de control simples y complejos de cualquier lenguaje de modelado o especificación de PNCs.

H2. Si se verifica H1, es posible definir un método de transformación de modelos que permita generar automáticamente modelos formales estructurados de PNCs a partir de modelos o de especificaciones definidos en diferentes lenguajes de PNCs

H3. Si se verifican H1 y H2, es posible definir un método formal de verificación de PNCs estructurados tal que: (1) de soporte a la verificación de PNCs cuyo flujo de control esté definido con constructores complejos; (2) pueda ser utilizado en distintos lenguajes de PNCs y en diferentes etapas de procesos de desarrollo dirigido por modelos.

H4. Si se verifica H3, es posible definir un método de verificación de PNCs estructurados basado en anti-patrones de comportamiento, que reutilice los resultados de verificación y

mejore el rendimiento del método formal de verificación, sin afectar el criterio de completitud.

H5. Si se verifican H1 y H2, también es posible definir un método de transformación de modelos basado en modelos formales que permita generar una especificación de un PNC a partir de un modelo del mismo, tal que el comportamiento de la especificación esté alineado con el del modelo.

A continuación se detallan los objetivos necesarios para verificar las hipótesis formuladas.

1.3.2. Objetivos

Esta tesis tiene como objetivo principal el desarrollo de métodos y herramientas para la verificación de PNCs y la transformación de modelos de PNCs a especificaciones cuyo comportamiento esté alineado con el de los modelos. Para ello se proponen métodos y herramientas basados en técnicas formales junto con la aplicación de los conceptos del desarrollo dirigido por modelos. El principal propósito es proveer a los analistas de negocio y diseñadores/desarrolladores de sistemas el soporte necesario para:

- Determinar si los modelos de PNCs satisfacen un conjunto de propiedades de comportamiento.
- Generar especificaciones de PNCs a partir de modelos estructurados, tal que el comportamiento de la solución tecnológica esté alineado con el de la solución de negocio.

Para alcanzar el objetivo principal se plantean los siguientes objetivos específicos:

- Definir un lenguaje formal que de soporte a la formalización de lenguajes de PNCs a efectos de formalizar modelos y especificaciones de PNCs estructurados, cuyos flujos de control estén definidos con constructores simples y complejos.
- Definir un método de transformación de modelos formales de PNCs, que permita la generación automática de un modelo formal a partir de un modelo o

una especificación de PNC estructurado. El propósito es dar soporte a la formalización automática de PNCs.

- Definir un método de verificación de PNCs basado en modelos formales y en los conceptos del desarrollo dirigido por modelo. El propósito es determinar si un modelo de PNC satisface un conjunto de propiedades de comportamiento, transformando el mismo en un modelo formal y verificando si el modelo formal satisface dicho conjunto de propiedades de comportamiento. Esto implica definir formalmente propiedades de comportamiento que den soporte a la detección de bloqueos en el flujo de control de PNCs. Las propiedades definidas deben considerar constructores simples y complejos.
- Definir un método de verificación de PNCs basado en anti-patrones de comportamiento. El propósito es proveer un método de verificación con buen rendimiento que satisfaga el criterio de completitud, reusando resultados del método formal de verificación propuesto.
- Definir un método de transformación para la generación automática de especificaciones a partir de modelos de PNCs, de manera tal que el comportamiento de la especificación esté alineado con el del modelo. Para ello, se propone el uso y generación de modelos formales de PNCs como modelos intermedios del proceso de transformación. Esto requiere utilizar el lenguaje formal y el método de transformación propuestos en los objetivos anteriores.
- Diseñar e implementar herramientas prototipo que implementen los métodos propuestos, con el propósito de realizar una validación empírica de los mismos.

1.4. Principales Contribuciones

Las principales contribuciones de la tesis son las siguientes:

1. *Lenguaje formal para procesos de negocio colaborativos GI-Nets*

   Se define el lenguaje formal redes de Interacción Global (GI-Nets), el cual es una extensión de las redes de Petri Jerárquicas y Coloreadas (Jensen & Kristensen 2009), que posibilita definir modelos formales de PNCs. Este lenguaje es independiente de la semántica de cualquier lenguaje

de PNCs, lo que lo hace flexible y adaptable para ser utilizado con diferentes lenguajes de PNCs.

El lenguaje GI-Nets permite expresar una semántica formal del comportamiento de los constructores de un dado lenguaje de PNCs, y de modelos y especificaciones definidos con dichos constructores. Los constructores pueden ser simples o complejos y deben poder ser definidos de manera estructurada.

2. *Método de transformación para generar modelos formales de PNCs.*

   Se define un patrón de transformación que permite generar automáticamente modelos formales basados en GI-Nets a partir de modelos o especificaciones de PNCs estructurados. Estos modelos formales pueden ser utilizados por métodos de verificación y transformación de modelos.

3. *Métodos de verificación del comportamiento de PNCs.*

   Se proponen dos métodos de verificación que permiten determinar si un PNC satisface un conjunto de propiedades de comportamiento.

   a. *Método de Verificación basado en GI-Nets.*

      El método da soporte a la verificación de procesos con constructores simples y complejos, incluyendo sincronización avanzada, manejo de excepciones y cancelaciones, e instancias múltiples, en donde los PNCs son expresados en modelos formales de GI-Nets. Para ello se presenta la propiedad *Solidez de Interacción Global* de una GI-Net, la cual está basada en el análisis de espacio de estados y permite determinar si un modelo de PNC está libre de bloqueos. Este método permite también detectar el lugar específico donde existe un bloqueo, lo que facilita la corrección del modelo o de la especificación de un PNC. La principal ventaja de este método es que satisface el criterio de completitud, ya que permite verificar modelos con constructores complejos y definidos en cualquier lenguaje de modelado o especificación de PNCs.

   b. *Método de Verificación basado en Anti-patrones de comportamiento.*

El método se basa en el uso de anti-patrones de comportamiento para llevar a cabo la verificación de modelos y especificaciones de PNCs. Provee un enfoque que permite especificar en forma sistemática los anti-patrones para un dado lenguaje de PNC. El principal beneficio de este método reside en que la verificación formal de PNCs se realiza una sola vez para un dado lenguaje de PNCs. Luego, sólo se reutilizan los resultados de verificación obtenidos por los métodos formales. Este método además de detectar el lugar específico donde se produce un problema, permite detectar el conjunto exacto de elementos que produce dicho problema, lo que facilita aún más la corrección de errores por parte de analistas y diseñadores. Debido a que reutiliza resultados de métodos formales de verificación y el proceso de verificación se limita a una técnica de búsqueda de coincidencias de anti-patrones, el método satisface los criterios de completitud y rendimiento.

4. *Método de transformación de PNCs que garantiza alineación de comportamiento.*

   Se define un método formal de transformación de modelos, que permite generar especificaciones de PNCs a partir de modelos conceptuales de PNCs. El método garantiza que el comportamiento de una especificación de PNC generada esté alineado con el comportamiento definido en el modelo conceptual de PNC. Para lograr esta alineación, se utilizan modelos formales de PNCs con GI-Nets como representaciones intermedias del proceso de transformación, donde las GI-Nets formalizan tanto al modelo como a la especificación del PNC. De esta manera, si el comportamiento del modelo de PNC está bien definido, entonces la especificación de dicho proceso generada automáticamente por el método de transformación también lo estará, y viceversa. Se aplica este método a la generación de especificaciones de PNCs basadas en tecnologías de servicios Web a partir de modelos conceptuales de PNCs.

5. *Herramientas de formalización, verificación y transformación de modelos y especificaciones de PNCs*

   Se presentan dos herramientas para la formalización, verificación y transformación de modelos y especificaciones de PNCs. La primera de ellas implementa:

   - un editor de GI-Nets que permite llevar a cabo la formalización de constructores de PNCs mediante GI-Nets,
   - una máquina de transformación para la generación de modelos formales basados en GI-Nets a partir de modelos de PNCs definidos con UP-ColBPIP,
   - un módulo para la verificación automática de PNCs definidos con el lenguaje UP-ColBPIP, la cual hace uso de la máquina de transformación para generar GI-Nets. La herramienta genera una GI-Net a partir de un modelo de PNC definido con UP-ColBPIP, determina las propiedades que cumple dicha GI-Net, y realiza una interpretación de las propiedades para indicar si existen errores en el comportamiento de un PNC,
   - una máquina de transformación para la generación de especificaciones definidas en WS-CDL a partir de modelos de PNCs definidos con UP-ColBPIP.

   La segunda herramienta implementa el método de verificación basado en anti-patrones de comportamiento para modelos definidos con UP-ColBPIP.

1.5. Organización de la Tesis

En el Capítulo 2 se introduce el marco teórico en el que se basa la tesis. Presenta una revisión de los conceptos principales del desarrollo dirigido por modelos de sistemas de información para colaboraciones inter-organizacionales, y de lenguajes formales basados en redes de Petri. Finalmente se presenta una discusión de los trabajos relacionados en el área de verificación y generación de procesos inter-organizacionales.

En el Capítulo 3 se describe el lenguaje formal redes de Interacción Global (GI-Nets) que permite llevar a cabo la formalización de los constructores de lenguajes de PNCs, y la formalización de los modelos o especificaciones que se pueden definir con dichos lenguajes. Se describe un patrón de transformación que permite generar modelos formales basados en GI-Nets a partir de modelos o especificaciones de PNCs. Por último, se presenta la formalización de los lenguajes UP-ColBPIP y BPMN con GI-Nets, y se presenta un caso de estudio de formalización de modelos. Los contenidos de este capítulo fueron derivados de (Roa, Chiotti & Villarreal 2012a; Roa, Chiotti & Villarreal 2012b).

En el Capítulo 4 se describe un método de verificación de PNCs estructurados basado en el lenguaje formal GI-Nets. Primero, se introduce la propiedad *Solidez de Interacción Global* que está basada en el análisis de espacio de estados para detectar bloqueos en el flujo de control de PNCs. Finalmente se presentan dos casos de estudio en los cuales se aplica el método de verificación. Los contenidos de este capítulo fueron derivados de (Roa, Chiotti & Villarreal 2012a; Roa, Chiotti & Villarreal 2012b).

En el Capítulo 5 se introduce un método de verificación de PNCs basado en anti-patrones de comportamiento. Primero, se presenta un enfoque sistemático para especificar anti-patrones de comportamiento. Luego, en base a este enfoque, se lleva a cabo la especificación de anti-patrones de comportamiento del lenguaje UP-ColBPIP, y se describe el método de verificación correspondiente basado en antipatrones. Los contenidos de este capítulo fueron parcialmente derivados de (Roa, Chiotti & Villarreal 2013).

En el Capítulo 6 se introduce un método de transformación que genera especificaciones de PNCs cuyos comportamientos están alineados con el de sus correspondientes modelos de PNCs. Se formaliza el concepto de alineación entre modelo y especificación de PNCs, se presenta un patrón de transformaciones, y se define la condición suficiente para alineación entre modelo y especificación. Los contenidos de este capítulo fueron derivados de (Roa, Chiotti & Villarreal 2012b).

En el Capítulo 7 primero se describen las herramientas implementadas que permiten formalizar, verificar, y transformar PNCs, y luego, se presentan la evaluación de los métodos propuestos y los resultados obtenidos. Los contenidos de este capítulo fueron parcialmente derivados de (Roa, Chiotti & Villarreal 2012b).

En el Capítulo 8 se presentan las conclusiones del trabajo de tesis, una síntesis de las principales contribuciones y se identifican los trabajos futuros relacionados con el enfoque de investigación de la tesis.

2 Marco teórico y Trabajos Relacionados

En este capítulo se describen el marco teórico de la esta tesis y los trabajos relacionados. Se introduce la metodología de desarrollo dirigido por modelos para colaboraciones inter-organizacionales (Sección 2.1), y se destaca la importancia de la verificación de PNCs y la generación de especificaciones alineadas. Se presentan conceptos generales del modelado y especificación de PNCs y se describen los lenguajes de modelado UP-ColBPIP y BPMN, y el lenguaje de especificación WS-CDL (Sección 2.2). Se presentan conceptos sobre la estructura y el comportamiento de modelos estructurados de PNCs, y las implicancias de su uso (Sección 2.3). Se describen lenguajes que permiten formalizar PNCs y propiedades de comportamiento para la verificación de los mismos (Sección 2.4). Finalmente, se presentan los trabajos relacionados (Sección 2.5).

2.1. Desarrollo Dirigido por Modelos de Colaboraciones Inter-Organizacionales

Los métodos de verificación y transformación que se proponen en esta tesis siguen un enfoque "Top-Down" basado en los principios del desarrollo dirigido por modelos y se enmarcan en la metodología para dar soporte al diseño e implementación de colaboraciones inter-organizacionales propuesta en (Lazarte et al. 2010). La metodología define diferentes niveles de abstracción en los que se debe hacer foco durante el desarrollo de colaboraciones inter-organizacionales. Esto ayuda a los analistas de negocio y a los diseñadores y desarrolladores de sistemas a analizar y diseñar la solución desde diferentes niveles de abstracción mediante la separación de intereses, y la identificación de los productos o artefactos requeridos en el desarrollo de colaboraciones inter-organizacionales.

La metodología provee un proceso basado en los principios del desarrollo dirigido por modelos para solucionar problemas de interoperabilidad, heterogeneidad, autonomía de las organizaciones, y de alineación de la solución de negocio con la solución tecnológica. Propone hacer foco en los modelos independientes de la plataforma que definen la solución

de negocio y de sistemas en un nivel alto de abstracción, y generar a partir de los mismos las especificaciones dependientes de la tecnología, que definen la solución tecnológica. Para ello, propone definir modelos conceptuales de procesos de negocio y de sistemas para derivar los artefactos de la solución tecnológica de una colaboración inter-organizacional. Estos artefactos generados son las especificaciones de los PNC y las interfaces de sistemas inter-organizacionales de las organizaciones que participan de la colaboración.

La metodología define cuatro fases de desarrollo (Figura **0.1**): *Análisis de Negocio*, *Diseño de la Solución de Negocio*, *Diseño de la Solución de Arquitectura TI*, y *Diseño de la Solución Tecnológica*.

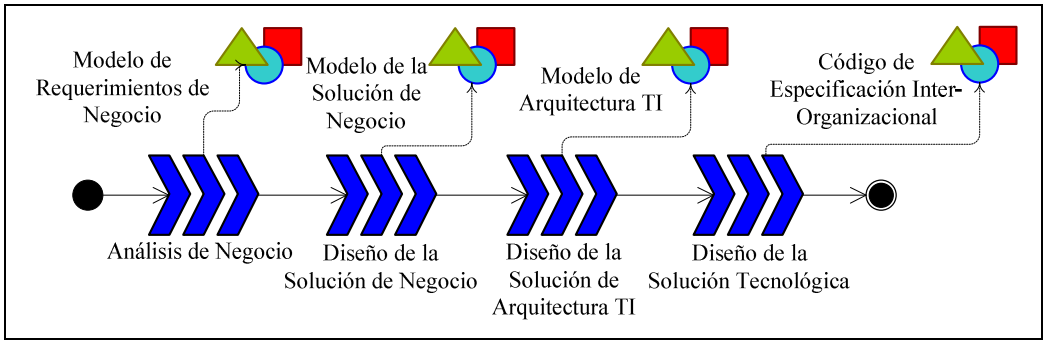


Fig. 0.1. Fases de la metodología para el desarrollo de colaboraciones inter-organizacionales

La fase *Análisis de Negocio* se enfoca en el análisis del dominio del problema y la identificación de los requerimientos de negocio. Esta fase implica un trabajo conjunto entre las organizaciones que participan de la colaboración. Para definir el *modelo de requerimientos de negocio* se utiliza el lenguaje UP-ColBPIP (Villarreal, Salomone & Chiotti 2007; Villarreal et al. 2010) que da soporte al modelado del contexto de negocios de PNCs. Este lenguaje permite definir la *Vista de Colaboración* y la *Vista de Procesos de Negocio Colaborativos* de un modelo de colaboración inter-organizacional. La salida de esta fase es el *modelo de requerimientos de negocio*, el cual es representado por ambas vistas. Este modelo ayuda a los analistas de negocio a comprender el problema e identificar los PNCs a ser diseñados en las siguientes fases.

La fase *Diseño de la Solución de Negocio* se enfoca en el diseño de los PNCs, los cuales describen el comportamiento de las interacciones entre las organizaciones desde una perspectiva global, indicando los roles que las organizaciones desempeñan en los PNCs, el

intercambio de mensajes e información, y la coordinación de sus actividades. Los analistas de negocio y diseñadores de sistemas de las organizaciones modelan los PNCs mediante el lenguaje UP-ColBPIP, definiendo el comportamiento de los PNCs en términos de protocolos de interacción. Un protocolo de interacción define los mensajes que representan las interacciones inter-organizacionales, junto con la lógica del flujo de control de los mensajes.

En esta fase, una tarea importante a considerar es la verificación de modelos de PNCs, con el propósito de detectar errores en el comportamiento de los mismos en etapas tempranas del desarrollo, previo a la generación de la solución de arquitectura TI. La salida de esta fase son *modelos de PNCs* que satisfacen un conjunto de propiedades de comportamiento.

La fase *Diseño de la Solución de Arquitectura TI* se enfoca en definir el *modelo de arquitectura de Tecnología de Información, con independencia* de la plataforma de implementación, a efectos de separar la solución lógica de TI de su implementación técnica, permitiendo su reproducción en diferentes plataformas. El modelo de arquitectura TI se deriva de los *modelos de PNCs* definidos en la fase anterior, y es diseñado por los diseñadores TI para representar la parte de la solución que se va a mantener sin cambios en cualquier plataforma, considerando el paradigma de arquitectura de software que las organizaciones acuerdan. La arquitectura Orientada a Servicios o la arquitectura de Sistemas Multi-Agentes son candidatas a ser utilizadas en la definición de un modelo de arquitectura TI desde un punto de vista conceptual.

La fase *Diseño de la Solución Tecnológica* se enfoca en la definición de la *solución tecnológica* para dar soporte a la ejecución descentralizada de PNCs mediante los sistemas de información inter-organizacionales. Esta solución es definida por los desarrolladores TI utilizando conceptos específicos de la plataforma de implementación de modo de convertir dicha solución en código ejecutable. En esta fase se acuerdan los estándares y las tecnologías a utilizar en la implementación de las especificaciones inter-organizacionales. En base a la tecnología seleccionada se generan las especificaciones de PNCs, por ejemplo, en el caso de usar plataformas basadas en servicios Web, se utilizará WS-CDL (W3C-WSCDL 2005). Las especificaciones de los PNCs deben estar alineadas con los modelos de los PNCs.

Esta metodología destaca dos aspectos importantes del desarrollo de colaboraciones inter-organizacionales que aún no están completamente cubiertos: disponer de herramientas de verificación de modelos conceptuales de PNCs y de especificaciones de PNCs; y garantizar la alineación de las especificaciones de PNCs con los modelos de PNCs a partir de los cuales son generadas.

2.2. Modelado y Especificación de Procesos de Negocio Colaborativos

Existen diferentes lenguajes para el modelado e implementación de PNCs, tales como BPMN (OMG-BPMN 2011), UP-ColBPIP (Villarreal et al. 2010), WS-CDL (W3C-WSCDL 2005), entre otros. Un lenguaje de PNCs puede ser visto como un lenguaje específico de dominio, cuya semántica está relacionada al dominio de las colaboraciones inter-organizacionales. Tiene una semántica estructural, la cual describe el significado de un modelo en términos de la estructura de las instancias de dicho modelo y es consistente con las reglas y restricciones definidas en el metamodelo del lenguaje.

La *semántica estructural* de un lenguaje de PNCs puede ser definida por su sintaxis abstracta y concreta. La *sintaxis abstracta* refiere a un metamodelo (o esquema XML) que describe los conceptos del lenguaje, sus relaciones, y las reglas de estructuración que restringen los elementos de un modelo, y cómo los mismos pueden ser combinados respetando las reglas del dominio semántico (Vallecillo 2008). La *sintaxis concret*a refiere a la notación (textual o gráfica) utilizada para representar dichos conceptos. Ambas sintaxis se relacionan mediante una función de mapeo desde la sintaxis concreta al dominio semántico de la sintaxis abstracta. La semántica de un lenguaje debe proveer el significado de cada expresión, y ese significado debe ser un elemento en algún dominio que debe estar bien definido y comprendido (Harel & Rumpe 2004).

Los elementos de la sintaxis concreta son conocidos usualmente como los constructores de un lenguaje. Un *constructor* puede ser visto como un elemento abstracto definido a nivel del metamodelo, mientras que una *instancia de constructor* es un elemento concreto definido a nivel del modelo. Un constructor puede tener infinitas instancias, y las mismas pueden ser utilizadas para generar un conjunto infinito de modelos desde un dado

metamodelo. Los constructores de un lenguaje son definidos por un conjunto no vacío de elementos del metamodelo del lenguaje, y las relaciones entre las instancias de los constructores son determinadas y restringidas por el metamodelo del lenguaje. Debido a que un modelo de PNC está siempre compuesto de instancias de constructores, en esta tesis se utiliza el término *elemento de PNC* o simplemente *elemento* para referir a una instancia de un constructor dentro de un modelo de PNC. En las siguientes secciones se describen los lenguajes para modelado de PNCs UP-ColBPIP, BPMN, y WS-CDL.

2.2.1. UP-ColBPIP

El lenguaje UP-ColBPIP extiende la semántica de UML2 para modelar PNCs independientes de la tecnología (Villarreal 2005; Villarreal, Salomone & Chiotti 2007; Villarreal et al. 2010). Fue definido como un perfil de UML para proveer una notación gráfica conocida, sencilla de comprender por analistas de negocio y diseñadores de sistemas. Propone un enfoque "Top-Down" para modelar PNCs y provee los elementos conceptuales para dar soporte a cinco vistas: (1) *Vista Colaboración Inter-Organizacional*, que define los participantes de una colaboración inter-organizacional y sus roles, junto con sus relaciones de comunicación y los objetivos de la colaboración; (2) *Vista de Procesos de Negocio Colaborativos*, que permite identificar y representar los PNCs requeridos para alcanzar los objetivos acordados por las organizaciones; (3) *Vista de Protocolos de Interacción*, que se enfoca en la representación del comportamiento de los PNCs mediante la definición de protocolos de interacción; (4) *Vista de Documentos de Negocio*, que se enfoca en la representación de los documentos de negocio que van a ser intercambiados en los PNCs; y (5) *Vista de Interfaces de Negocio*, que permite describir las interfaces de cada rol ejecutado por una determinada organización, las cuales deben ser vistas como servicios que contienen operaciones de negocio que soportan el intercambio asíncrono de mensajes de los protocolos de interacción.

Esta tesis se enfoca en la *Vista de Protocolos de Interacción* debido a que está relacionada con la descripción del flujo de control de los PNCs. La misma se describe con mayor detalle en la siguiente sección.

Vista de Protocolos de Interacción

Los principales requerimientos para el modelado conceptual de PNCs y colaboraciones inter-organizacionales son: vista global de las interacciones entre las organizaciones, autonomía de las organizaciones, gestión descentralizada, interacciones punto-a-punto, y representación de negociaciones complejas (Villarreal, Salomone & Chiotti 2007; Weske 2007). Para dar soporte a estos requerimientos, UP-ColBPIP aplica el concepto de protocolo de interacción para definir el comportamiento de los PNCs. Un *protocolo de interacción* describe un patrón de comunicaciones de alto nivel a través de una coreografía de mensajes de negocio entre organizaciones que participan de una colaboración desempeñando diferentes roles.

El modelado de protocolos de interacción se enfoca en representar la vista global de las interacciones entre las organizaciones. La coreografía de mensajes describe el flujo de control global de interacciones punto-a-punto entre organizaciones y las responsabilidades de los roles que desempeñan. Permite también representar la gestión descentralizada de las interacciones.

Los protocolos de interacción se enfocan en el intercambio de mensajes de negocio que representan interacciones entre las organizaciones. Las actividades internas de las organizaciones no pueden ser definidas en los protocolos, lo que permite preservar la autonomía de las organizaciones.

Las interacciones entre organizaciones deberían representar además del intercambio de información, la comunicación de acciones entre las organizaciones. Los aspectos de coordinación y comunicación de las interacciones se representan en los protocolos de interacción mediante el uso de *actos de comunicación* (Searle 1976). En un protocolo de interacción, un mensaje de negocio tiene asociado un acto de comunicación que representa la intención del emisor respecto al documento de negocio enviado en el mensaje. Por lo tanto, a partir de los actos de comunicación se pueden conocer las decisiones y compromisos entre las organizaciones. Esto permite definir negociaciones complejas y evita la ambigüedad en la semántica y el entendimiento de los mensajes de negocio de los PNCs.

UP-ColBPIP extiende la semántica de las Interacciones de UML2 para modelar protocolos de interacción, con lo cual los protocolos son definidos utilizando Diagramas de

Secuencia de UML2. A continuación se describen los principales constructores y elementos conceptuales utilizados para definir protocolos de interacción (Figura **0.2**).

Las organizaciones y los roles que éstas desempeñan en la colaboración se representan a través de "lifelines" (Figura **0.2** a)). La semántica de ejecución del protocolo está dada por el orden en que los elementos están definidos en las "lifelines" y por ende en el protocolo. Cuando un elemento finaliza su ejecución se habilita la ejecución del siguiente elemento en el orden en que aparecen en las "lifelines".

El constructor básico de un protocolo de interacción es el mensaje de negocio. Un *Mensaje de Negocio (Business Message)* define una interacción asíncrona en una dirección entre dos roles, emisor y receptor. Contiene un *documento de negocio* que representa la información a intercambiar entre las organizaciones, cuya semántica se define por un *acto de comunicación*. De esta manera, un mensaje de negocio permite que el emisor exprese mediante un acto de comunicación su intención con respecto al documento de negocio intercambiado en el mensaje, y la expectativa que tiene respecto de cómo debe responder el receptor. La Figura **0.2** a) muestra el mensaje de negocio *request(ForecastRequest)* que se envía desde el rol A al rol B. Este mensaje indica a través del acto de comunicación *request* que mediante el documento de negocio *ForecastRequest* se está solicitando un pronóstico.

Una *Referencia de Protocolo (Protocol Reference)* representa a un subprotocolo o protocolo anidado. Cuando se invoca al subprotocolo, el protocolo espera hasta que el subprotocolo finalice. Los protocolos pueden tener una *Terminación explícita*, la cual puede ser de tipo *Éxito (Success)*, que representa que el protocolo finaliza de manera exitosa o *Falla (Failure)*, que representa que el protocolo finaliza con una falla. La Figura **0.2** a) muestra un ejemplo en el que el protocolo finaliza con éxito (*Success*) luego que se envía el mensaje *request(ForecastRequest)*.

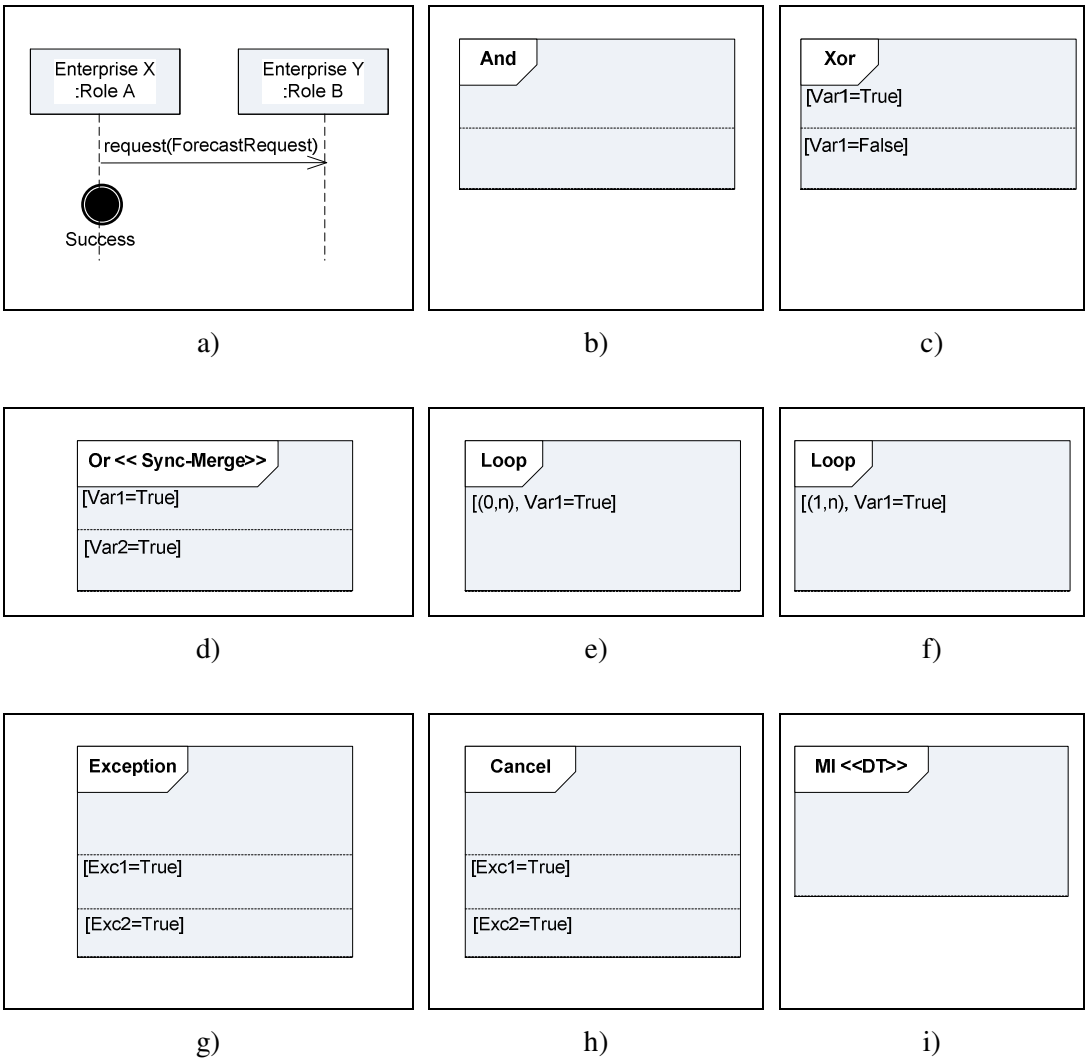


Fig. 0.2. Notación gráfica de los constructores de UP-ColBPIP

Un *Segmento de Flujo de Control* permite representar el flujo de control de los mensajes. Hay distintos tipos de segmentos, los cuales pueden estar compuestos de uno o más caminos de interacción. Un *Camino de Interacción (Interaction Path)* contiene una secuencia ordenada de elementos, los cuales pueden ser: mensajes, eventos de terminación, referencias a protocolos y segmentos de flujo de control anidados. La semántica de un segmento de flujo de control depende del tipo de flujo de control, los cuales pueden ser:

- *And* (Figura **0.2** b)), que representa la ejecución paralela de dos o más caminos de interacción. El flujo de control pasa al siguiente elemento del protocolo cuando todos los caminos finalizaron su ejecución.

- *Xor*, que representa que sólo uno de un conjunto de caminos de interacción alternativos puede ser ejecutado. Un *Xor* basado en datos (Figura **0.2** c)) contiene condiciones en los caminos a ser evaluadas para seleccionar el camino de ejecución a ejecutar. Un *Xor* basado en eventos depende de la ocurrencia de un evento de envío o recepción del primer mensaje de cada camino de interacción para seleccionar el camino que se ejecuta. En este caso los caminos no tienen asociados condición alguna. Se puede definir un temporizador en un camino de interacción para representar la ejecución de ese camino cuando ocurre un determinado evento de tiempo.
- *Or*, que representa que se pueden ejecutar caminos de interacción alternativos. Para permitir la ejecución de cada camino de interacción se debe evaluar una condición. Se pueden definir cuatro tipos de sincronización, los cuales se denotan con sus correspondientes etiquetas en la parte superior del segmento de flujo de control: (1) *Synchronizing Merge* (<<Sync-Merge>>) (Figura **0.2** d)) indica que el flujo de control pasa al siguiente elemento del protocolo cuando cada camino de interacción que fue ejecutado finaliza con su ejecución; (2) *Discriminator* (<<Disc>>) indica que el flujo de control pasa al siguiente elemento del protocolo cuando se completa la ejecución de un camino de interacción; (3) *N out of M* (<<N out of M>>) indica que el flujo de control pasa al siguiente elemento del protocolo cuando se completa la ejecución de N caminos de interacción, donde M representa al total de caminos de interacción que componen al segmento de flujo de control; (4) *Multi-merge* (<<Multi-Merge>>) indica que para cada camino de interacción que completa su ejecución, se continúa con la ejecución del protocolo en el elemento que sigue al *Or*.
- *Loop*, que representa que un camino de interacción se puede ejecutar varias veces. Puede ser del tipo *While* (Figura **0.2** e)), donde el camino de interacción se va a ejecutar mientras que se cumpla una determinada condición, con lo cual se puede ejecutar cero o más veces; o de tipo *Until* (Figura **0.2** f)), donde el camino de interacción se va a ejecutar hasta que se cumpla una condición, con lo cual se va a ejecutar al menos una vez.

- *Exception* (Figura **0.2** g)), que define el camino a seguir luego que aparece una determinada excepción. Consiste de un camino de interacción que representa el alcance de la excepción, el cual va a contener a todos los elementos que formen parte de dicho camino. Además consiste de uno o más caminos de interacción que representan los manejadores de las excepciones que deben ser capturadas y gestionadas. Un camino de interacción que representa a un manejador de excepción tiene una condición asociada que determina cuándo se genera la excepción. Una vez que el manejador de excepciones finaliza, el protocolo continúa su ejecución normal en el elemento que sigue al *Exception*.
- *Cancel* (Figura **0.2** h)), que define el camino a seguir luego que aparece una determinada excepción. A diferencia del *Exception*, el *Cancel* finaliza la ejecución del protocolo cuando el manejador de excepciones finaliza su ejecución. Se utiliza para finalizar un protocolo de manera consistente y coherente luego que ocurre una excepción.
- *Multiple Instances*, que representa múltiples instancias de un camino de interacción. Se pueden definir cuatro tipos de instancias múltiples, lo cual se indica con una etiqueta en la parte superior izquierda de un segmento de flujo de control: (1) en tiempo de diseño (<<DT>>) (Figura **0.2** i)); (2) en tiempo de ejecución (<<RT>>), donde se debe indicar en tiempo de ejecución la cantidad de instancias que se van a generar; (3) sin conocimiento previo en tiempo de ejecución (<<WRTK>>), donde se debe indicar la expresión de condición que permite la creación de nuevas instancias; (4) sin conocimiento (<<WS>>), donde no se conoce la cantidad de instancias que se van a generar ni en tiempo de diseño ni en tiempo de ejecución.
- *If*, que representa un camino de interacción que se ejecuta si la condición definida en el mismo es verdadera. Un *If* puede tener además otro camino de interacción con condición *Else*, el cual se va a ejecutar en caso que la condición del primer camino de interacción sea falsa.

Una *Restricción de Tiempo* indica un límite de tiempo asociado a los mensajes, segmentos de flujo de control o protocolos. Es decir, el tiempo límite disponible para la ejecución de dichos elementos. Esta restricción se puede definir utilizando tiempos relativos o absolutos.

La Figura **0.3** muestra un ejemplo de un protocolo de interacción de un *pronóstico de demanda colaborativo*, el cual describe un PNC que debe ser ejecutado como parte de un modelo de negocio de *Planificación, Pronóstico y Reaprovisionamiento Colaborativo* (VICS n.d.). Este protocolo define un PNC simple entre un cliente y un proveedor para colaborar y acordar en un pronóstico de demanda de productos a intercambiar.

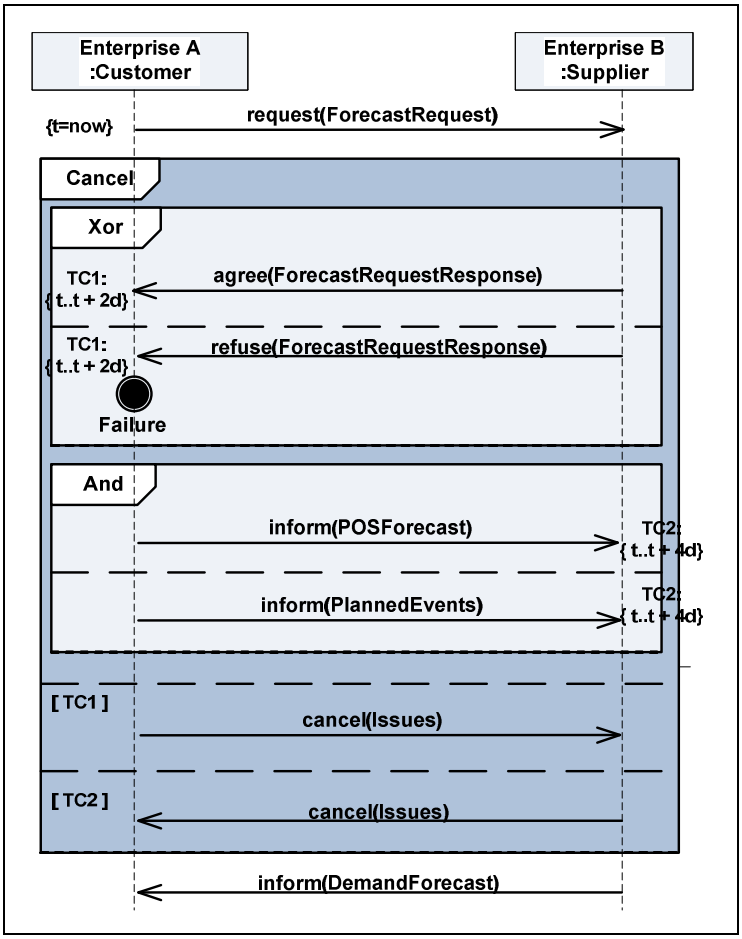


Fig. 0.3. Protocolo de pronóstico de demanda colaborativo

El PNC comienza con el cliente, el cual solicita un pronóstico de demanda. El mensaje de solicitud, cuyo acto de comunicación es *request* transmite los datos a considerar

en el pronóstico, como por ejemplo los productos. El proveedor gestiona la solicitud y debería responder mediante una aceptación o un rechazo. Si la solicitud es aceptada, el proveedor lleva a cabo el pronóstico requerido, lo cual es indicado por el acto de comunicación *agree*. En caso que la solicitud es rechazada, el PNC finaliza con una falla.

Si el proveedor acepta la solicitud, el cliente informa en paralelo un pronóstico de sus puntos de ventas y su política de venta planificada. Con esta información, el proveedor genera un pronóstico de demanda, se lo informa al cliente, y el PNC finaliza.

Los mensajes de respuesta que el proveedor envía y los mensajes de información que envía el cliente tienen definidas restricciones de tiempo con tiempos relativos que representan el tiempo límite para el envío y recepción de los mensajes. Como ejemplo, los tiempos límites de los mensajes *agree* y *refuse* indican que los mismos tienen que ser enviados a lo sumo dos días luego del envío del primer mensaje.

Para la gestión de excepciones de estos mensajes, se agregó el segmento Cancel. Este segmento contiene un camino de interacción que representa el alcance sobre el cual puede ocurrir una excepción y abarca los mensajes con restricciones de tiempo, y tiene además otros dos caminos de interacción que permiten gestionar las excepciones de tiempo definidas en los mensajes mencionados. En ambos caminos, la gestión de excepciones consiste en el envío de un mensaje de cancelación.

2.2.2. BPMN

BPMN (Business Process Modeling Notation) (OMG-BPMN 2011) es un lenguaje estándar de facto para el modelado de procesos de negocio. Aunque surgió para el modelado de procesos de negocio privados de las organizaciones, la última versión de BPMN introduce el concepto de coreografía *(Coreography*) para modelar PNCs desde una perspectiva global. Una coreografía define un flujo de actividades de interacción entre participantes. Está compuesta de nodos (actividades de coreografía, eventos, o "gateways") conectados por medio de arcos dirigidos. La Figura **0.4** muestra un subconjunto de los constructores de BPMN utilizados para el modelado de PNCs.

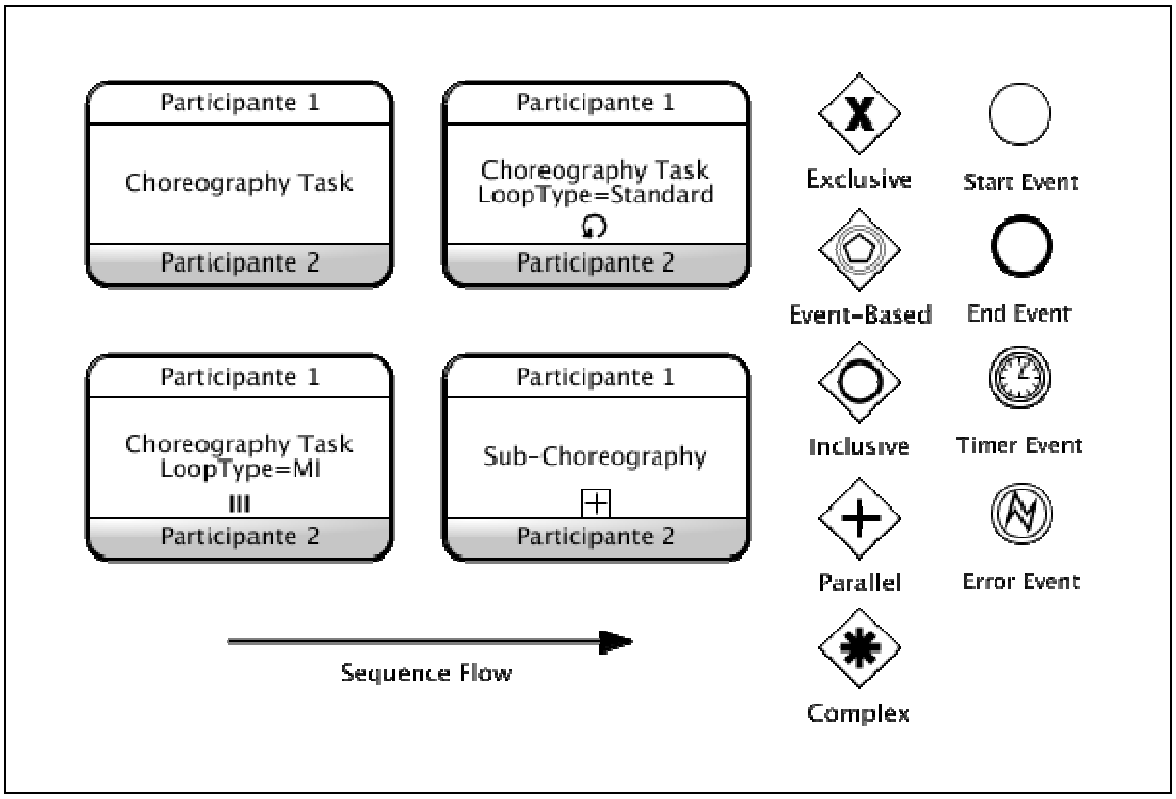


Fig. 0.4. Constructores de BPMN

El constructor *Choreography Task* representa una interacción entre participantes. El nombre de la tarea y de cada participante se muestra en diferentes bandas. Hay dos o más bandas para los participantes y una para el nombre de la tarea. La banda del participante que inicia la tarea no debe estar sombreada. *Choreography Task* puede ser utilizado para indicar una tarea repetitiva mediante un ciclo (atributo *LoopType=Standard*) o por instancias múltiples (atributo *LoopType=MI*). El constructor *Sub-Choreography* indica una composición de tareas de coreografía.

Un "gateway" es un constructor de ruteo, el cual puede ser utilizado como punto de bifurcación/decisión o sincronización/unión:

- *Exclusive*: es utilizado para crear caminos alternativos en una coreografía. La decisión sobre qué camino se ejecuta está basada en información contenida en los documentos de negocio intercambiados entre los participantes. También puede ser utilizado para unir caminos alternativos.
- *Event-Based*: es utilizado para representar un punto de bifurcación donde los caminos alternativos están basados en la ocurrencia de eventos.

- *Inclusive*: es utilizado para representar caminos cuya ejecución depende de condiciones específicas. Al igual que el *Exclusive*, la decisión sobre qué camino se ejecuta está basada en información de los documentos de negocio. Es también utilizado para modelar la sincronización de un dado número de caminos, de los cuales no todos están necesariamente activos.
- *Parallel*: es utilizado para crear caminos que se ejecutan al mismo tiempo. También es utilizado para la sincronización de caminos concurrentes.
- *Complex*: no tiene asignada una semántica específica de comportamiento. Cada diseñador le puede dar la semántica que considere adecuada. Puede ser utilizado, por ejemplo, para modelar uniones parciales, donde un dado número de caminos deben ser sincronizados.

En BPMN es posible definir eventos para indicar el inicio y fin de un PNC (*Start* y *End Event* respectivamente). También se pueden definir eventos intermedios (*Intermediate Event*), como por ejemplo un evento de tiempo (*Timer Event*), o un evento de error (*Error Event*). Un evento de tiempo o error en el borde de una tarea de coreografía es utilizado para gestionar una excepción. Si se dispara la excepción, la tarea se interrumpe y el proceso continúa en el flujo de excepción.

2.2.3. WS-CDL

WS-CDL (Web Services Choreography Description Language) (W3C-WSCDL 2005) es un lenguaje basado en XML que describe colaboraciones punto a punto e interoperables entre servicios Web de las organizaciones mediante la definición, desde un punto de vista global, del comportamiento común y observable, donde el intercambio ordenado de mensajes entre los servicios resulta en alcanzar una meta de negocio común entre las partes. Este lenguaje permite la especificación de PNCs basados en tecnologías o plataformas de servicios Web.

WS-CDL consta de las siguientes entidades:

- *roleType, relationshipType*, y *participantType*: en una coreografía, las interacciones ocurren entre roles ejecutados por organizaciones que se restringen por una relación. Una organización se modela de manera abstracta

por un tipo de participante (*participantType*), un rol por un tipo de rol (*roleType*), y una relación mediante un tipo de relación (*relationshipType*):

  - *participantType* agrupa un participante del comportamiento público de la coreografía, el que debe ser implementado por una organización.
  - *roleType* enumera el comportamiento observable potencial que un *participantType* puede exhibir para interactuar.
  - *relationshipType* identifica los compromisos mutuos que deben ser realizados para que la colaboración sea exitosa.

- *informationType, variable*, y *token*: una *variable* contiene información sobre objetos públicos en común en una colaboración, tales como la información a ser intercambiada o la información pública de los tipos de roles que participan en la colaboración. Una señal (*token*) es un alias que puede ser utilizado para referenciar partes de una variable. Las variables de intercambio de información, las de captura de estado, y las de señales tienen tipos de información (*informationTypes*) que definen el tipo de información que contienen dichas variables.
- *choreography*: una coreografía (*choreography*) define colaboraciones entre tipos de participantes (*participantTypes*) que interactúan entre sí.
  - *choreography life-line*: una "*life-line"* de una coreografía expresa la progresión de la colaboración. Primero, se establece la colaboración entre los participantes, luego se llevan a cabo interacciones que forman parte de la colaboración, y por último la colaboración finaliza de manera normal o anormal.
  - *choreography exception blocks*: un bloque de excepción (*exception block*) especifica qué acciones adicionales se deberían ejecutar si una coreografía se comporta de manera inesperada.
  - *choreography finalizer blocks*: un bloque de finalización (*finalizer block*) especifica qué acciones adicionales se deberían ejecutar para modificar el efecto de una coreografía que se completó previamente de manera exitosa, por ejemplo para confirmar o deshacer un efecto.

- *channelType*: un canal representa un punto de colaboración entre *participantTypes* especificando dónde y cómo se intercambia la información. En WS-CDL, los canales se modelan de manera abstracta por medio de tipos de canal (*channelTypes*).
- *workunit*: una unidad de trabajo (*workunit*) establece las restricciones que se deben cumplir para avanzar en la colaboración, y por lo tanto para que se lleven a cabo interacciones en una coreografía.
- *activities*: las actividades (*activities*) describen las acciones llevadas a cabo en una coreografía. Para definir una actividad se utiliza *ordering structure*, *workunit-notation*, o *basic activity*:
    - *ordering structures*: las estructuras de ordenamiento (*ordering structures*) combinan actividades con otras estructuras de ordenamiento de manera anidada para expresar reglas de ordenamiento de las acciones que se deben llevar a cabo en una coreografía. Pueden ser de tres tipos: *sequence*, *parallel*, y *choice*.
    - *workunit-notation*: se utiliza para dar soporte a la repetición de las actividades que se encuentran dentro de una *workunit*.
    - *basic activities*: describen las acciones de más bajo nivel de una coreografía, las cuales pueden ser:
        - *interaction activity*: una interacción (*interaction*) es el constructor básico de una coreografía. Representa un intercambio de información entre los participantes y una posible sincronización de cambios en la información pública.
        - *perform activity*: significa que se ejecuta una coreografía definida de manera separada de la actual.
        - *assign activity*: dentro de un *roleType* asigna el valor de una variable a otra.
        - *silentAction activity*: representa un punto específico en el cual un participante ejecuta acciones con detalles operacionales no observables.

- *noAction activity*: permite especificar un punto en el cual un participante no ejecuta ninguna acción.
- *finalize activity*: permite habilitar un *finalizerBlock*.

- *semantics*: la semántica (*semantics*) permite la creación de descripciones que pueden registrar las definiciones semánticas de cada componente en el modelo.

2.3. Estructura de Modelos de PNCs

La estructura del modelo de un PNC es determinada por la combinación de los elementos que se pueden definir a partir de un dado lenguaje de PNCs. El metamodelo del lenguaje juega un rol importante en este aspecto. En base a la estructura y a la combinación de elementos, se pueden distinguir dos tipos de lenguajes de PNCs: estructurados, también conocidos como basados en bloques, y no estructurados, también conocidos como basados en grafos (Kopp et al. 2009).

Un lenguaje no estructurado de PNCs, tal como BPMN, provee constructores no estructurados. El metamodelo de este tipo de lenguajes permite generar instancias de constructores que pueden ser combinadas formando un grafo no estructurado de elementos. Por el contrario, un lenguaje estructurado de PNCs, tal como UP-ColBPIP o WS-CDL, provee constructores estructurados, obligando a que cada elemento del modelo de un PNC sea anidado dentro de otro por medio de relaciones y restricciones definidas en el metamodelo o sintaxis del lenguaje. Esto implica que la estructura de los modelos estructurados de PNCs puede ser representada mediante un árbol (Vanhatalo, Völzer & Leymann 2007; Vanhatalo, Völzer & Koehler 2009).

Si bien los lenguajes no estructurados permiten definir modelos no estructurados de PNCs, también es posible usar estos lenguajes para definir modelos estructurados. En cambio, los lenguajes estructurados sólo permiten generar modelos estructurados de PNCs. La Figura **0.5** muestra un ejemplo en BPMN de un modelo no estructurado de PNC (arriba) y su correspondiente versión estructurada (abajo).

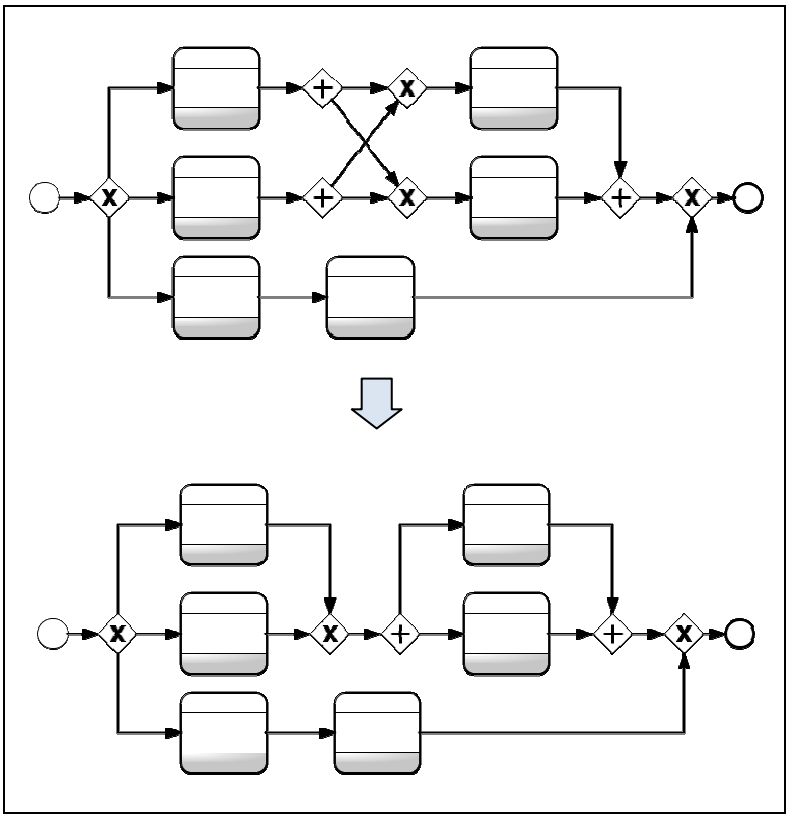


Fig. 0.5. PNC no estructurado (arriba) y su versión estructurada (abajo)

2.3.1. Modelos Estructurados de PNCs vs. No Estructurados

El tópico *estructuración* vs. *no estructuración* fue originalmente discutido en distintos trabajos en el dominio de los lenguajes de programación y compiladores (Ferrante, Ottenstein & Warren 1987; Graham & Wegman 1976). Actualmente, los lenguajes de programación más utilizados proveen sentencias (constructores) estructuradas. Sentencias del tipo *go to* que permiten la definición de programas no estructurados fueron dejadas de lado, para dar lugar a sentencias con una estructura basada en bloques de tipo SESE (Single Entry - Single Exit) (Johnson, Pearson & Pingali 1994). Esto se debió (entre otros motivos) a que las sentencias del tipo *go to* inducían a los programadores a cometer muchos errores.

En el dominio de procesos de negocio y "workflows", en cambio, predominan los lenguajes no estructurados, tales como BPMN, los diagramas de actividades de UML (OMG 2011), o las variantes de estos lenguajes como la utilizada en la herramienta WebSphere Business Modeler de IBM (IBM 2013). El lenguaje BPMN provee más de 50

constructores para desarrollar modelos no estructurados de procesos (Muehlen & Recker 2008). Esto permite que los modelos no estructurados de procesos tengan mayor expresividad que los estructurados (Liu & Kumar 2005), ya que ofrecen mayor libertad para elegir cómo combinar los constructores de un lenguaje. Sin embargo, los modelos no estructurados de procesos son más propensos a errores (Mendling, Neumann & Aalst 2007), y suelen dificultar su comprensión (Mendling, Reijers & Cardoso 2007).

Los modelos estructurados de procesos, en cambio, si bien limitan las posibles combinaciones de los constructores, usualmente son más fáciles de comprender (Mendling, Neumann & Aalst 2007; Polyvyanyy & Bussler 2013). Aunque se ha comprobado que existen modelos no estructurados de procesos que no pueden ser representados de manera estructurada (Vanhatalo, Völzer & Koehler 2009), los modelos utilizados para mostrar estos ejemplos no tienen utilidad en casos reales (Polyvyanyy & Bussler 2013). En casos reales, en general, es posible obtener versiones estructuradas de modelos a partir de modelos no estructurados de procesos, aunque como contraparte, algunos modelos estructurados pueden tener mayor tamaño que los no estructurados (Mendling, Reijers & Cardoso 2007).

Otro aspecto a considerar en cuanto a la estructuración de los modelos de procesos es que mejora el rendimiento de los métodos de verificación (Vanhatalo, Völzer & Leymann 2007; Vanhatalo, Völzer & Koehler 2009), e incluso facilita la interpretación de resultados, ya que los errores quedan acotados a un conjunto de constructores que forman un bloque.

Además de las ventajas y desventajas de utilizar modelos estructurados de procesos, es necesario considerar que en una metodología de desarrollo de colaboraciones inter-organizacionales se utilizan distintos lenguajes de modelado y especificación de PNCs. Algunos motores de ejecución de procesos soportan lenguajes como BPEL y WS-CDL, los cuales son estructurados. Esto significa que si la plataforma de ejecución está basada en BPEL y WS-CDL, las partes no estructuradas de un modelo BPMN no podrían ser representadas en dichos lenguajes y sería necesario un etapa de estructuración previa a la generación de las especificaciones (Zhao et al. 2006). La transformación de modelos no estructurados basados en BPMN a BPEL o WS-CDL no es una tarea sencilla de llevar a cabo (Weidlich, Großkopf & Weske 2008).

Debido a las ventajas mencionadas en relación al uso de modelos estructurados, principalmente en cuanto a su verificación, los métodos de verificación y transformación que se proponen en la tesis se enfocan en modelos estructurados de PNCs. En la siguiente sección se describen las características estructurales y de comportamiento de este tipo de modelos.

2.3.2. Modelos Estructurados de PNCs

Un modelo de PNC es estructurado si cada elemento de bifurcación/decisión tiene asociado un elemento de sincronización/unión tal que con los elementos del modelo es posible armar una estructura de árbol, donde cada nodo del árbol representa tanto la bifurcación/decisión como la sincronización/unión. De esta manera, un modelo estructurado de PNC está compuesto por bloques, donde cada bloque tiene una semántica de comportamiento establecida por la semántica de los elementos de bifurcación/decisión y sincronización/unión que lo componen. Además, cada bloque está compuesto de un conjunto de secuencias, donde la ejecución de dichas secuencias depende de la semántica de comportamiento del bloque.

Se considera que un lenguaje de PNCs estructurados tiene dos constructores primitivos $Interaction$ y $Sequence$, y otros constructores para representar otras semánticas de comportamiento específicas, tales como paralelismo, exclusión mutua, ciclos, o excepciones. La Figura **0.6** muestra la representación estructurada en forma de árbol de algunos ejemplos de estos tipos de constructores de flujo de control estructurados. Los mismos representan constructores usualmente utilizados en distintos lenguajes de procesos, que no pertenecen a un lenguaje específico.

Los constructores primitivos $Interaction$ (Int), $Sequence(Seq)$, y el constructor no primitivo $Termination$ $(Term)$ son representados por un único nodo. En cambio, los demás constructores están compuestos, en general, por un nodo que determina la semántica de comportamiento del constructor y de un conjunto de nodos del tipo $Sequence$. Por ejemplo, un constructor *And* puede estar compuesto de dos o más secuencias que representan los caminos que se ejecutan en paralelo. Algo similar ocurre con los constructores *Xor* u *Or*, los cuales están compuestos de secuencias alternativas, donde las del *Xor* son mutuamente excluyentes, y las del *Or* pueden ser concurrentes. Un constructor

Loop está compuesto de una única secuencia que se ejecuta de manera iterativa. Un constructor del tipo *Exception* está compuesto de una secuencia que representa el alcance (*Scope*) de la excepción, y un conjunto de secuencias que representan a los manejadores (*Handler*) de las distintas excepciones definidas en el constructor *Exception*.

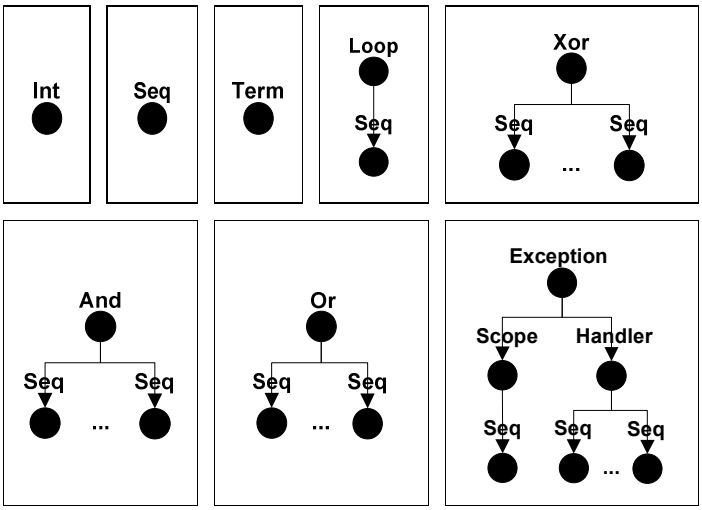
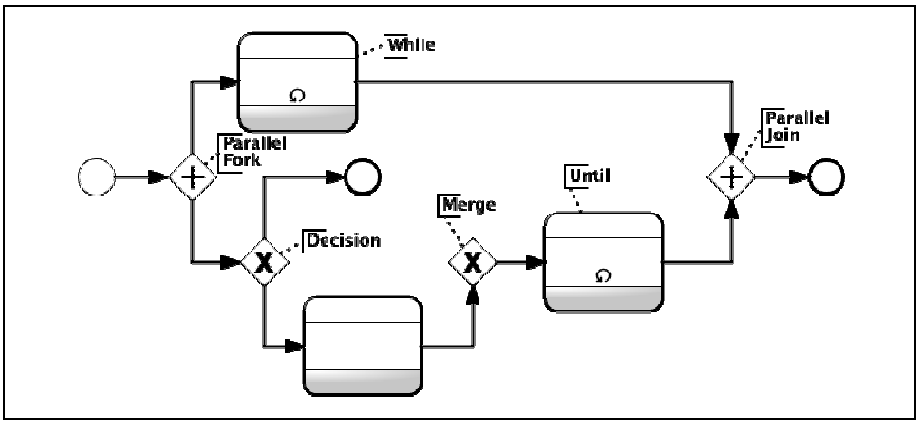


Fig. 0.6. Ejemplos de constructores de PNCs estructurados

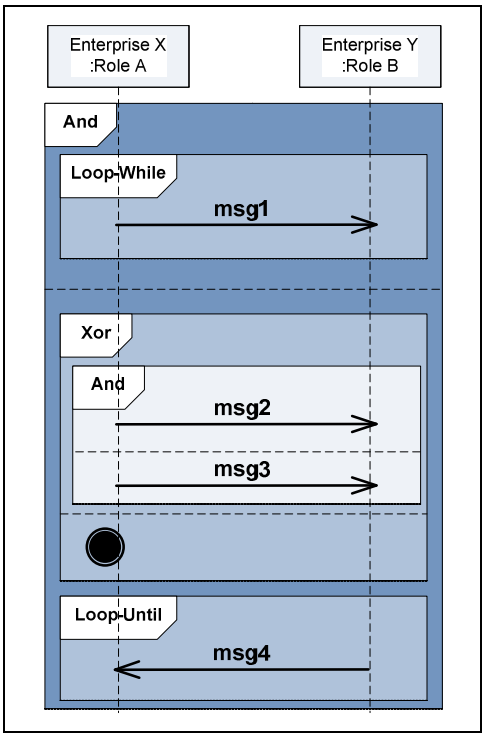
Los constructores no primitivos pueden tener una semántica de comportamiento de apertura y una semántica de comportamiento de cierre. La Figura **0.7** muestra un ejemplo de un modelo estructurado de PNC. Las Figuras **0.7** a) y b) muestran a este modelo definido con los lenguajes BPMN y UP-ColBPIP respectivamente, mientras que la Figura **0.7** c) muestra la vista estructurada de dicho modelo.

En la vista de árbol de la Figura **0.7** c) cada nodo representa a uno o más constructores del modelo definido en BPMN (Figura **0.7** a)). Por ejemplo, el nodo Seq_0 (nodo *Seq* con Id 0) representa al modelo de PNC $SP1$, el cual es una secuencia que comienza en el elemento de BPMN *Start event* y finaliza en el elemento *End event*. Como $SP1$ es un modelo estructurado, el mismo puede ser visto como una secuencia de elementos. En la vista estructurada se omite al elemento *Start event* que inicia el PNC y al elemento *End event* que cierra el PNC, y se muestra solamente la secuencia de elementos contenidos dentro los mismos. En este caso la secuencia está compuesta solamente del

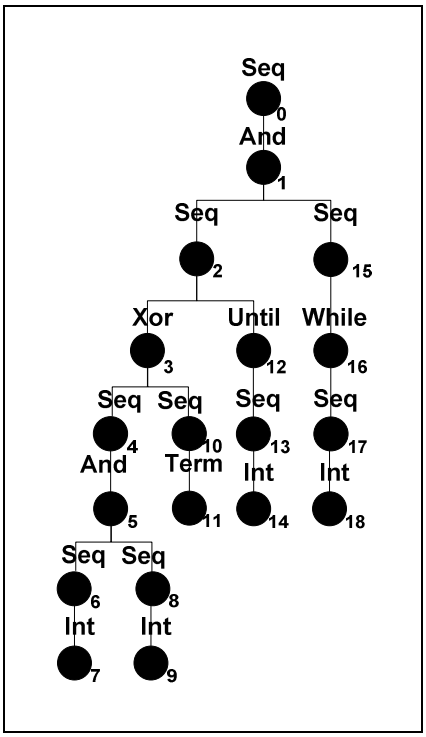
elemento And_1. Todos los demás elementos del modelo de PNC se encuentran anidados dentro de dicho elemento.



a) Modelo del PNC $SP1$ definido con BPMN



b) Modelo del PNC $SP1$ definido con UP-ColBPIP



c) Vista estructurada $SP1$

Fig. 0.7. Modelos del PNC estructurado $SP1$

El elemento And_1 (nodo *And* con Id 1) de la vista estructurada, representa al constructor estructurado que se forma con los dos elementos *Parallel* del modelo BPMN, y

el elemento Xor_3 (nodo *Xor* con Id 3), representa al constructor estructurado que se forma con los dos elementos *Exclusive*. Esto significa que en la vista de árbol, cada nodo puede tener asociado una semántica de comportamiento de apertura y una semántica de comportamiento de cierre. Por otro lado, los flujos de secuencia que van desde un *Parallel* al otro son representados por los elementos Seq_2 y Seq_{15} respectivamente. Un elemento de flujo de control que represente bifurcación o decisión, siempre va a estar compuesto de secuencias (elementos $Sequence$), donde cada secuencia representa un posible camino de ejecución.

Los elementos definidos en el modelo UP-ColBPIP (Figura **0.7** b)) tienen un mapeo directo a la vista estructurada de la Figura **0.7** c), debido a que UP-ColBPIP es un lenguaje estructurado. El elemento Seq_0 de la vista estructurada representa al protocolo de interacción. El elemento And_1 junto con los elementos Seq_2 y Seq_{15} representan al elemento *And* de UP-ColBPIP. Las secuencias Seq_2 y Seq_{15} representan a los dos caminos de interacción que componen al *And*. El elemento Xor_3 junto con los elementos Seq_4 y Seq_{10} representan al elemento *Xor* de UP-ColBPIP, el cual también está compuesto de dos caminos de interacción. El elemento $Until_{12}$ junto con el elemento Seq_{10} representan al elemento *Loop-Until* de UP-ColBPIP, el cual está compuesto de un camino de interacción.

Además de una semántica estructural, un lenguaje de PNC tiene una semántica de comportamiento. La *semántica de comportamiento* describe la evolución del estado de una instancia de un modelo a lo largo del tiempo, y se define a nivel de metamodelo. El uso de constructores de flujo de control es esencial para definir el comportamiento de PNCs, ya que los constructores de flujo de control determinan la semántica de comportamiento de los elementos de un modelo de PNC.

Los lenguajes usualmente proveen constructores básicos con una semántica de comportamiento simple, como secuencia, bifurcación y sincronización para representar concurrencia, y decisión y unión para representar exclusión mutua. La semántica de comportamiento de otros constructores tales como ciclos, cancelaciones, manejo de excepciones, o instancias múltiples pueden ser representadas mediante la combinación de constructores básicos.

La semántica de comportamiento de un constructor es influenciada por la estructura de un modelo de proceso. Por ejemplo, en un lenguaje no estructurado, el comportamiento

de un constructor de bifurcación (o decisión) es independiente del constructor de sincronización (o unión), mientras que en un lenguaje estructurado, se obliga a que el comportamiento de ambos tipos de constructores para bifurcación (o decisión) y sincronización (o unión) sea definido en un constructor único e indivisible.

2.4. Lenguajes para la Formalización y Verificación del Comportamiento de PNCs

La semántica de comportamiento de los constructores de los lenguajes de PNCs puede ser capturada y descripta formalmente por un "framework" matemático que permite representar la dinámica de los constructores. La formalización permite prevenir ambigüedades en los lenguajes de PNCs, así como la verificación de sus modelos para determinar el correcto comportamiento de los PNCs.

Si bien existen diversos métodos y lenguajes formales que permiten analizar y verificar modelos en distintas etapas del desarrollo de software, en esta tesis se utilizan las redes de Petri. Éstas son apropiadas para el modelado de sistemas o procesos concurrentes y proveen además diferentes técnicas para verificar las propiedades deseadas de un modelo de proceso (van der Aalst 1997; van der Aalst 1998). Otro aspecto importante es que pueden ser utilizadas tanto para verificación como para validación de modelos, permitiendo la reutilización de modelos formales en ambas técnicas. Han sido utilizadas con éxito en la formalización y verificación de constructores de flujo de control para procesos de negocio tanto en contextos intra-organizacionales (van der Aalst 1998; Russell et al. 2006; Dijkman, Dumas & Ouyang 2008; Fahland et al. 2009) como inter-organizacionales (Van Der Aalst 2003; Stuit & Szirbik 2009; van der Aalst et al. 2010; Norta & Eshuis 2010). En esta tesis se aplican al dominio de PNCs y colaboraciones inter-organizacionales.

Las redes de Petri se originaron a partir del trabajo de Carl Adam Petri (Petri 1962). Desde entonces han sido utilizadas y estudiadas en diversas áreas. En (Murata 1989) es posible encontrar una revisión histórica de la aplicación de las redes de Petri, ejemplos de modelado, propiedades de comportamiento, y subclases de redes Petri junto con un análisis de las mismas. A continuación se presentan las redes de Petri clásicas, las Workflow Nets,

y las redes de Petri Jerárquicas y Coloreadas, las cuales son utilizadas para la formalización y verificación del comportamiento de procesos de negocio.

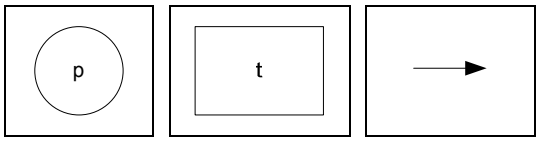
2.4.1. Redes de Petri Clásicas

Las redes de Petri permiten la descripción formal de sistemas a través de modelos gráficos y matemáticos. Admiten expresar concurrencia, sincronización, exclusión mutua, y conflictos. El soporte gráfico puede ser utilizado de manera similar a los diagramas de flujo o de bloques, mientras que el soporte matemático permite utilizar ecuaciones de estado, ecuaciones algebraicas y otros modelos matemáticos para analizar el comportamiento de los sistemas.

Definición 0.1. ***(Red de Petri)*** **Una red de Petri es una tripleta** $(P,\ T,\ F)$**, donde:**

− P es un conjunto finito de posiciones,
− T es un conjunto finito de transiciones,
− $F \subseteq (P \times T) \cup (T \times P)$ es un conjunto de arcos (relación de flujo).

Una red de Petri es un tipo particular de grafo dirigido, junto con un estado inicial al cual se lo llama *marcado inicial (o estado inicial)*, M_0. El grafo N de una red de Petri es un grafo dirigido, con pesos, y bipartito que consiste de dos tipos de nodos, llamados *posiciones* y *transiciones*, donde los *arcos* van de las posiciones a las transiciones o de las transiciones a las posiciones. Gráficamente las posiciones (Figura **0.8** a)) se representan como círculos (o elipses), las transiciones (Figura **0.8** b)) como rectángulos, y los arcos como líneas con flechas que indican el origen y el destino de la conexión (Figura **0.8** c)).



a) Posición b) Transición c) Arco

Fig. 0.8. Elementos de un red de Petri

Las posiciones representan condiciones y las transiciones representan eventos. Una transición tiene una determinada cantidad de *posiciones de entrada* y *salida* que representan respectivamente pre-condiciones y post-condiciones de un evento. A una posición p se lo llama *posición de entrada* de una transición t sii existe un arco dirigido desde p a t. A una posición p se la llama *posición de salida* de una transición t sii existe un arco dirigido desde t a p. Se usa $\bullet t$ para denotar las posiciones de entrada de la transición t. De manera análoga $t\bullet$ denota las posiciones de salida de la transición t, $p\bullet$ denota las transiciones que comparten a p como posición de entrada, y $\bullet p$ denota las transiciones que comparten a p como posición de salida.

Una posición puede contener cero o más señales, las cuales se denotan como puntos negros. El estado, usualmente llamado marcado, es la distribución de señales sobre la posiciones, es decir, $M \in P \to \mathbb{N}$, donde $\mathbb{N}$ representa a los números naturales. La comparación de estados se define como un orden parcial. Para dos estados cualesquiera M_1 y M_2, $M_1 \leq M_2$ sii para toda $p \in P: M_1(p) \leq M_2(p)$, siendo $M_1(p)$ el marcado o cantidad de señales del estado M_1 en p.

El número de señales puede variar durante la ejecución de una red. Las transiciones cambian el estado de una red de acuerdo a la siguiente *regla de disparo*:

1. Se dice que una transición t está *habilitada* sii cada posición de entrada p de t contiene al menos una señal.
2. Una transición habilitada puede ser *disparada* (ejecutada). Si se dispara la transición t, entonces t *consume* una señal de cada posición de entrada p de t y *produce* una señal en cada posición de salida p de t.

Se utiliza (PN, M) para indicar a una red de Petri PN con un estado inicial M. Se utiliza M_p para denotar al estado en el cual hay señales solamente en la posición p. La Figura **0.9** muestra un ejemplo de una red de Petri, en la cual hay tres posiciones p1, p2, y

p3, y dos transiciones t1 y t2. La transición t1 está habilitada, ya que p1 tiene tres señales y p3 tiene una señal. En cambio, la transición t2 no está habilitada, ya que la posición p2 no tiene ninguna señal. Si se dispara t1, se consume una señal de p1 y una de p3, y se produce una señal en p2.

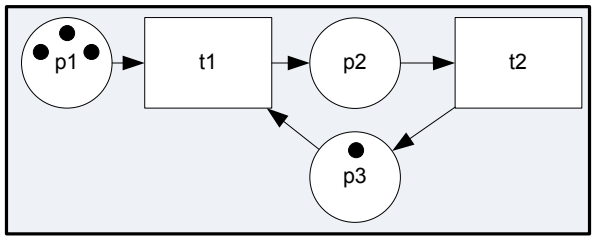


Fig. 0.9. Elementos de un red de Petri

A continuación se introducen algunas propiedades y características de las redes de Petri.

Definición 0.2. ***(Viva, Acotada, Camino, Red fuertemente conectada)***

- Una red de Petri (PN, M) está *viva* sii para cada estado alcanzable M' y cada transición t, existe un estado M'' alcanzable a partir de M' que hablita a la transición t.
- Una red de Petri (PN, M) está *acotada* sii para cada posición p existe un número natural n tal que para cada estado alcanzable el número de señales en p es menor que n.
- Sea PN una red de Petri, un *camino* C que va de un nodo n_1 a un nodo n_k es una secuencia $\langle n_1, n_2, \dots, n_k \rangle$ tal que $(n_i, n_{i+1}) \in F$ para $1 \leq i \leq k-1$.
- Una red de Petri está fuertemente conectada sii para cada par de nodos (es decir, posiciones y transiciones) x e y, existe un camino que va de x a y.

Definición 0.3. ***(Estado muerto, Estado origen, Transición muerta)***

- Un estado muerto es un estado M en el cual no hay transiciones habilitadas.
- Un estado origen M_{home} es un estado que puede ser alcanzado desde cualquier estado M.
- Una transición t es una transición muerta si no existen estados alcanzables tal que t está habilitada.

2.4.2. Workflow Nets

Una *WorkFlow Net (WF-Net)* es una red de Petri que modela la definición de un proceso de workflow. Una WF-Net satisface dos requerimientos. Primero, tiene una única posición de entrada (i) y una única posición de salida (o). Segundo, cada transición y posición debería estar ubicada en un camino que va de i a o.

Definición 0.4. ***(WorkFlow Net)*** **Una red de Petri** $PN = (P, T, F)$ es una WorkFlow Net (WF-Net) sii:

1. PN tiene dos posiciones especiales: i y o. La posición i es una posición de entrada: $\bullet i = \emptyset$. La posición o es una posición de salida: $o\bullet = \emptyset$.
2. Si se agrega una transición t^* a PN que conecte la posición i con la posición o (es decir, $\bullet t^* = \{o\}$ y $t^*\bullet = \{i\}$), entonces la red resultante es fuertemente conectada.

Solidez de una WF-Net

Para determinar si una WF-Net es correcta, se propuso la propiedad *solidez*. Una WF-Net que verifica esta propiedad se dice que es sólida. Para ello debe cumplir que: la red pueda finalizar en algún momento; cuando finalice debe haber una sola señal en la posición o y todas las demás posiciones deben estar vacías; y no deben haber transiciones muertas, es decir, debería ser posible ejecutar cada transición de la WF-Net. Esto se define de la siguiente manera.

Definición 0.5. ***(Solidez)*** (van der Aalst 1998) Una WF-Net $PN = (P, T, F)$ es sólida sii:

1. Para cada estado M alcanzable desde el estado M_i existe una secuencia de disparos que va desde el estado M al estado M_o. Formalmente: $\forall_M \left(M_i \overset{*}{\rightarrow} M\right) \Rightarrow \left(M \overset{*}{\rightarrow} M_o\right)$.
2. El estado M_o es el único estado alcanzable desde el estado M_i con al menos una señal en la posición o. Formalmente: $\forall_M \left(M_i \overset{*}{\rightarrow} M \wedge M \geq M_o\right) \Rightarrow (M = M_o)$.
3. No hay transiciones muertas en (PN, M_i). Formalmente: $\forall_{t \in T} \exists_{M,M'} M_i \overset{*}{\rightarrow} M \overset{t}{\rightarrow} M'$**.**

La propiedad solidez refiere a la dinámica de una WF-Net. El primer requerimiento en la Definición **0**.5 establece que a partir del estado inicial M_i, siempre es posible alcanzar el estado con una señal en la posición o (estado M_o). El segundo requerimiento establece que cuando la posición o tiene una señal, todas las demás posiciones deberían estar vacías. El tercer requerimiento establece que no hay transiciones muertas en el estado inicial M_i, es decir, a partir del estado M_i, para cada transición t es posible alcanzar un estado en el que t esté habilitada**.**

Dada una WF-Net $PN = (P, T, F)$, se quiere decidir si PN es sólida. Para esto, se define una red extendida $\overline{PN} = (\bar{P}, \bar{T}, \bar{F})$. $\overline{PN}$ es una red de Petri que resulta de agregar una transición extra t^* que conecta la posición o con la posición i. La red extendida $\overline{PN} = (\bar{P}, \bar{T}, \bar{F})$ se define de la siguiente manera: $\bar{P} = P$, $\bar{T} = T \cup \{t^*\}$, y $\bar{F} = F \cup \{\langle o, t^* \rangle, \langle t^*, i \rangle\}$.

Teorema 0.1. Una WF-Net PN es sólida sii $(\overline{GI}_N, M_i)$ está viva y es acotada.

Demostración.
Este teorema fue demostrado en (van der Aalst 1997).

2.4.3. Redes de Petri Jerárquicas y Coloreadas

Una red de Petri Coloreada (CP-Net) (Jensen & Kristensen 2009) es una tupla $CPN = (P, T, A, \Sigma, V, C, G, E, I)$ donde P es un conjunto finito de posiciones, T es un

conjunto finito de transiciones, y A es un conjunto finito de arcos que conecta posiciones y transiciones (y viceversa). Una CP-Net tiene un conjunto finito de conjuntos de colores (o tipos) Σ, un conjunto finito de variables V de un dado color (tipo) de Σ, y una función de conjunto de color C que asigna un conjunto de color a cada posición. Las transiciones pueden tener una *función de guarda* G, y los arcos pueden tener *funciones de expresión de arco* E. I es una función de inicialización de una CP-Net.

Definición 0.6. ***(Red de Petri Coloreada)*** Una red de Petri Coloreada (CP-Net) (Jensen & Kristensen 2009) es una tupla $CPN = (P, T, A, \Sigma, V, C, G, E, I)$, donde:

1. P es un conjunto finito de posiciones.
2. T es un conjunto finito de transiciones tal que $P \cap T$ es un conjunto vacío.
3. $A \subseteq P \times T \cup T \times P$ es un conjunto arcos dirigidos.
4. Σ es un conjunto finito y no vacío de conjuntos de colores.
5. V es un conjunto de variables de tipo tal que $Type[v] \in \Sigma$ para toda variable $v \in V$.
6. $C: P \rightarrow \Sigma$ es una función de conjunto de color que asigna un conjunto de color a cada posición.
7. $G: T \rightarrow EXPR_V$ es una función de guarda que asigna una guarda a cada transición t tal que $Type[G(t)] = Bool$.
8. $E: A \rightarrow EXPR_V$ es una función de expresión de arco que asigna una expresión de arco a cada arco a tal que $Type[E(a)] = C(p)_{MS}$ donde p es la posición conectada al arco a.
9. $I: P \rightarrow EXPR_\emptyset$ es una función de inicialización que asigna una expresión de inicialización a cada posición p tal que $Type[I(p)] = C(p)_{MS}$.

Un módulo CP-Net (Jensen & Kristensen 2009) es una tupla $CPN_M = (CPN, T_{sub}, P_{port}, PT)$, donde CPN es una CP-Net que tiene un conjunto de posiciones puerto P_{port}, las cuales tienen un tipo de puerto PT. Puede tener además un conjunto de transiciones de substitución T_{sub}.

Definición 0.7. ***(Módulo CP-Net)*** (Jensen & Kristensen 2009) Un módulo CP-Net es una tupla $CPN_M = (CPN, T_{sub}, P_{port}, PT)$, donde:

1. $CPN = (P, T, A, \Sigma, V, C, G, E, I)$ es una CP-Net.
2. $T_{sub} \subseteq T$ es un conjunto de *transiciones de substitución*.
3. $P_{port} \subseteq P$ es un conjunto de *posiciones puertos*.
4. $PT: P_{port} \rightarrow \{IN, OUT, I/O\}$ es una *función de tipo de puerto* que asigna un tipo de puerto a cada posición puerto.

Una red de Petri Jerárquica y coloreada (HCP-Net) (Jensen & Kristensen 2009) es una tupla $CPN_H = (S, SM, PS, FS)$, donde S es un conjunto de módulos CP-Net. Cada transición de substitución tiene su correspondiente módulo CP-Net por medio de la función de submódulo SM. Las posiciones de entrada y salida de un módulo CP-Net se asocian con las correspondientes posiciones de la transición de substitución por medio de la *función de relación puerto-enlace (port-socket)* PS. FS es un conjunto de *conjuntos de fusión* de posiciones. P^s y T^s son el conjunto de posiciones y transiciones respectivamente del módulo $s \in S$. $PT(p)$ es el tipo de puerto de la posición p. $Type[v]$ es el tipo (color) de una variable v.

Definición 0.8. ***(Red de Petri Jerárquica y Coloreada)*** (Jensen & Kristensen 2009) Una red de Petri Jerárquica y Coloreada (HCP-Net) es una tupla $\mathrm{CPN_H} = (\mathrm{S}, \mathrm{SM}, \mathrm{PS}, \mathrm{FS})$, donde**:**

1. S es un conjunto finito de módulos. Cada módulo es un módulo CP-Net $s = ((P^s, T^s, A^s, \Sigma^s, V^s, C^s, G^s, E^s, I^s), T^s_{sub}, P^s_{port}, PT^s)$**.** Se requiere que $(P^{s_1} \cup T^{s_1}) \cap (P^{s_2} \cup T^{s_2}) = \emptyset$ para todo $s_1, s_2 \in S$ tal que $s_1 \neq s_2$.
2. $SM: T_{sub} \rightarrow S$ es una función de submódulo que asigna un submódulo a cada transición de substitución. Se requiere que la jerarquía de los módulos sea no cíclica.
3. PS es una función de relación puerto-enlace (port-socket) que asigna una relación puerto-enlace $PS(t) \subseteq P_{sock}(t) \times P^{SM(t)}_{port}$ a cada transición de substitución t.
4. $PS \subseteq 2^P$ es un conjunto de conjuntos de fusión no-vacíos.

2.5. Trabajos Relacionados

En esta sección se presentan los trabajos relacionados en las áreas de verificación formal, verificación por medio de anti-patrones y alineación de comportamiento de PNCs.

2.5.1. Verificación de PNCs con Lenguajes Formales

Existen propuestas para la verificación de modelos de procesos inter-organizacionales o PNCs basadas en el formalismo de redes de Petri. En (Van der Aalst 1998; Norta & Eshuis 2010) se proponen enfoques para la verificación del comportamiento de modelos de procesos inter-organizacionales mediante IOWF-Nets (Inter-Organizational Workflow Nets). Para llevar a cabo la verificación, estos enfoques incluyen actividades privadas de las organizaciones, lo cual compromete (completa o parcialmente) la autonomía de las organizaciones. Este es un aspecto no deseable en colaboraciones inter-organizacionales, ya que las actividades internas de las organizaciones quedan expuestas a todos los participantes. En (Stuit & Szirbik 2009) se propone un método de verificación basado en HCP-Nets para PNCs dirigidos por agentes de software. Este método está basado en la propiedad de solidez utilizada en WF-Net para determinar si el modelo de interacción de agentes es correcto. Sin embargo, sólo permite determinar si existe un camino que lleve un PNC a finalizar con éxito, y no comprueba si el PNC está libre de bloqueos y fallos de sincronización. Finalmente, es de destacar que estos enfoques de verificación proponen que los PNCs sean modelados usando un lenguaje formal. El uso de un lenguaje formal para el modelado conceptual de PNCs no es adecuado, ya que las personas que participan en la definición de estos procesos, tales como analistas de negocio, no conocen de estos formalismos o les resulta difícil comprenderlos.

Existen enfoques basados en formalismos diferentes a las redes de Petri que permiten realizar la verificación sobre modelos o especificaciones de PNCs dependientes de la tecnología. En (Diaz et al. 2005) se presenta un método para transformar Servicios Web descriptos con WS-CDL a una orquestación de autómatas de tiempo. El método utiliza la herramienta UUPPAAL (Larsen, Pettersson & Yi 1997) para simular y verificar el sistema. En (Yang et al. 2006) se presenta el lenguaje CDL, el cual formaliza una parte de WS-CDL. Este lenguaje permite razonar sobre las propiedades que deberían ser satisfechas

por el sistema especificado mediante el uso de herramientas como SPIN (Holzmann 2003). En (Li et al. 2007) se propone la formalización de modelos de coreografías WS-CDL utilizando álgebra de procesos, y se muestra cómo determinar si una coreografía está libre de bloqueos. En (Carbone, Honda & Yoshida 2008) se proponen dos maneras diferentes de definir software centrado en comunicaciones utilizando cálculo formal. Una está basada en flujos de mensajes globales, los cuales se originan de WS-CDL y es llamado Cálculo Global, y otra basada en comportamientos locales representados con cálculo-π. Proponen una proyección donde cada descripción bien definida en el cálculo global tiene una representación en el cálculo local. Cuando se aplica a interacciones bien estructuradas, la proyección satisface un conjunto de propiedades como preservación de tipos, solidez, y completitud. En (Cambronero et al. 2011) se propone la transformación de PNCs definidos en WS-CDL a una red de autómatas de tiempo para verificar coreografías de Servicios Web. Utilizan la herramienta de chequeo de modelos UPPAAL para verificar: la ausencia de bloqueos; que ciertos estados son alcanzables; pre y post condiciones de ciertas actividades; y restricciones de tiempo. En estos enfoques, la verificación se realiza sobre modelos o especificaciones de PNCs dependientes de la tecnología. Por lo cual, los errores son detectados en etapas tardías del desarrollo de los PNCs, luego de generada la solución tecnológica. En el contexto de metodologías de desarrollo dirigido por modelos, la detección de errores a nivel de la especificación de un PNC, implica: retroceder al modelo definido en etapas anteriores, corregirlo y generar nuevamente la especificación; o bien, corregir directamente la especificación. Esto último implica que la especificación quede inconsistente con el modelo a partir del cual fue generada. Esto difícilta el mantenimiento del modelo en relación al código. Finalmente, es de destacar que estos enfoques de verificación están fuertemente acoplados a los lenguajes para los cuales fueron definidos. Esto dificulta la reutilización de los métodos en distintas etapas del desarrollo de PNCs, y requiere conocer e implementar diferentes métodos y herramientas de verificación.

Las limitaciones de las propuestas para la verificación de modelos de PNCs identificadas, permiten inferir la necesidad de enfoques de modelado y verificación que provean alto nivel de abstracción e independencia de las tecnologías de implementación.

Del análisis de los trabajos relacionados con la verificación de PNCs con lenguajes formales se puede resumir que: existen pocos métodos que se enfocan en verificar la

perspectiva global de los PNCs; estos métodos no soportan constructores complejos para sincronización avanzada, instancias múltiples, excepciones y cancelaciones, lo cual afecta al criterio de completitud; no se realizaron análisis de rendimiento de los mismos; y están altamente acoplados a sus respectivos lenguajes, lo que dificulta su reutilización en diferentes etapas del desarrollo de colaboraciones inter-organizacionales.

2.5.2. Verificación de PNCs con Anti-Patrones

La definición de un anti-patrón de comportamiento de PNCs propuesta en esta tesis (Capítulo 5) se basa en la idea de obtener el menor conjunto posible de elementos que represente a un determinado error, de manera tal que dicho error pueda ser generalizado. En el área de testeo de software, el proceso de obtener el menor conjunto de elementos posibles que producen un error es conocido como *simplificación* (Zeller & Hildebrandt 2002). En la simplificación se eliminan todos los detalles que son irrelevantes para producir una falla, y se utilizan los elementos restantes para generalizar el problema.

Uno de los primeros trabajos en el cual se propuso sistematizar el proceso de simplificación fue (Zeller & Hildebrandt 2002), en el que se utiliza un algoritmo basado en delta debugging (Zeller 1999). El algoritmo recibe como entrada un programa con una falla, al cual se le aplica un proceso de simplificación que elimina distintos elementos del programa. El algoritmo se detiene cuando al remover un elemento la falla desaparece.

En el área de procesos de negocios existen distintos trabajos en los cuales se propuso la definición y utilización de anti-patrones de comportamiento (Onoda et al. 1999; Liu & Kumar 2005; Dongen, Mendling & Aalst 2006; Koehler & Vanhatalo 2007; Gruhn & Laue 2009; Decker 2009; Laue & Awad 2010; Kühne et al. 2010; Han et al. 2013). Sin embargo, ninguno de ellos ha sido aplicado a procesos de negocio colaborativos que representan la vista global de una colaboración entre organizaciones. Además, en dichos trabajos no se consideró la definición de anti-patrones para flujos de control avanzados, lo que hace difícil que se alcance el criterio de completitud de estos métodos de verificación.

Por otra parte, tampoco se propusieron métodos para el descubrimiento y definición sistemática de anti-patrones de comportamiento. En estos trabajos, los anti-patrones son definidos mediante una inspección manual de librerías de modelos de procesos de negocio, o mediante la inspección de casos de estudio de la literatura. Si bien la inspección de

librerías de procesos permite detectar problemas de casos reales, sólo se enfoca en problemas comunes a los diseñadores que modelaron los procesos de dicha librería. Podrían existir otros tipos de errores que no es posible detectar mediante la inspección manual de una librería de procesos.

Del análisis de los trabajos relacionados al uso de anti-patrones de comportamiento se puede resumir que: no existen propuestas de verificación de PNCs basadas en anti-patrones de comportamiento que satisfagan simultáneamente el criterio de completitud y de rendimiento; y no hay métodos que permitan descubrir y determinar de manera sistemática los anti-patrones de comportamiento de PNCs.

2.5.3. Alineación de Comportamiento

Existen diferentes enfoques que se relacionan con la alineación entre un modelo de PNC y su especificación o entre modelos de PNCs, y enfoques que no son específicos del dominio de PNCs pero podrían aplicarse al mismo.

En el dominio de PNCs se pueden mencionar las siguientes propuestas. En (Decker & Weske 2007) se presentan nociones de compatibilidad y se las clasifica en estructural y de comportamiento. La compatibilidad estructural indica que para cada mensaje que puede ser enviado/recibido, el correspondiente socio de interacción debe estar disponible para recibir/enviar tal mensaje. La compatibilidad de comportamiento está relacionada al flujo de control entre el intercambio de diferentes mensajes dentro de una colaboración. En (Dijkman et al. 2009) se aborda el problema de alineación de procesos por correspondencia léxica y estructural, sin embargo, el comportamiento de los procesos no es tenido en cuenta. En (Danylevych, Karastoyanova & Leymann 2010) se propone un formalismo para modelar redes de servicios y se realiza un mapeo de los constructores propuestos en el formalismo a constructores de BPMN. En (Weidlich et al. 2009) se describen distintos aspectos y desafíos actuales relacionados a la alineación de modelos de procesos. En los enfoques mencionados el análisis se lleva cabo en el proceso global obtenido mediante la interconexión de los procesos de interfaz de las organizaciones que interactúan.

En el dominio de procesos de negocio se pueden mencionar las siguientes propuestas. En (Basten & van der Aalst 2001) se propone la herencia de procesos, la cual está relacionada con la teoría del paradigma orientado a objetos, donde atributos y

comportamiento son heredados por una subclase a partir de una superclase. La idea detrás de herencia de procesos es chequear si la especificación de un proceso hereda el comportamiento de su modelo. Este enfoque está basado en bi-simulación, el cual es utilizado para comparar dos definiciones de procesos. Con la bi-simulación (van Glabbeek & Weijland 1996) es posible determinar si un proceso puede simular el comportamiento de otro proceso.

En (Corrales, Grigori & Bouzeghoub 2006) y en (Minor, Tartakovski & Bergmann 2007) se aplican técnicas de *correspondencias de grafos* para la comparación de modelos de procesos. El enfoque es utilizado para comparar un modelo de proceso con una colección de procesos y determinar cuál de estos es el más similar. Dicho enfoque se relaciona a otros trabajos en el área de *medidas de similaridad* de modelos de procesos de negocio (Ehrig, Koschmider & Oberweis 2007; Li, Reichert & Wombacher 2007; Lu & Sadiq 2007; Madhusudan, Zhao & Marshall 2004; van Dongen, Dijkman & Mendling 2008; Wombacher 2006). Estos trabajos se relacionan con la alineación de comportamiento debido a que en la misma se asume que los modelos de procesos deben ser similares.

Se han definido también técnicas para determinar diferencias entre modelos de procesos (Dijkman 2007; Dijkman 2008; Küster et al. 2008; Weber, Reichert & Rinderle-Ma 2008). Estas técnicas asumen que existe un registro de cambios (change log) o que se analizaron los mapeos entre elementos de procesos. En (Dijkman 2007) se presenta una clasificación sistemática de diferencias entre procesos similares. Este trabajo describe discrepancias relacionadas al flujo de control, asignación de recursos, y correspondencias de actividades entre dos modelos que deben ser integrados. El trabajo propuesto en (Dijkman 2008) se enfoca sólo en la detección de discrepancias en el flujo de control.

La alineación entre modelos y especificaciones o código ha sido también estudiada en el dominio de métodos de transformación de modelos, aplicables a enfoques de desarrollo dirigido por modelos. En (Varró & Pataricza 2003) se presenta una técnica para verificar formalmente mediante chequeo de modelos que una transformación de modelos preserva las propiedades del modelo de entrada de la transformación. En (Dijkman et al. 2004) se presenta un enfoque basado en la refinación de estructura y comportamiento para determinar consistencia entre modelos UML definidos a nivel de negocio y a nivel tecnológico. En (Nejati et al. 2007) se aborda el problema de comparar y unir modelos del

comportamiento de máquinas de estado que están estructuradas jerárquicamente, sin considerar constructores para caminos paralelos.

Del análisis de los trabajos relacionados con la alineación entre modelos y especificaciones o código se puede resumir que: las propuestas para determinar alineación están basadas en técnicas de bisimulación, equivalencia de trazado, o chequeo de modelos, las cuales pueden requerir tiempos de cómputos exponenciales; si bien la alineación y la verificación de comportamiento están claramente relacionadas al comportamiento de PNCs, las propuestas existentes se enfocan en resolver ambos problemas por separado.

3 Formalización de Procesos de Negocio Colaborativos con GI-Nets

En este capítulo se presenta el lenguaje formal redes de Interacción Global (GI-Nets) (Sección 3.1), el cual es un tipo particular de redes de Petri que permite formalizar PNCs. En base a dicho lenguaje se propone un método para generar modelos formales GI-Nets a partir de modelos o especificaciones de PNCs (Sección 3.2), y las formalizaciones de los lenguajes UP-ColBPIP (Sección 3.3) y BPMN (Sección 3.4). Para mostrar la aplicabilidad del enfoque de formalización se introduce un caso de estudio en el cual se muestra la generación de una GI-Net que formaliza a un modelo de PNC definido con UP-ColBPIP (Sección 3.5). Finalmente, se presentan las conclusiones (Sección 3.6).

3.1. Lenguaje Formal Redes de Interacción Global

Con el objetivo de proveer un lenguaje formal que permita la formalización de modelos o especificaciones de PNCs definidos con lenguajes de PNCs que no proveen una semántica formal de sus constructores, se identificaron los siguientes requerimientos:

1. *Formalización de diferentes tipos de interacciones*. No existe acuerdo o consenso sobre cuál es el constructor adecuado para representar interacciones inter-organizacionales en PNCs desde una perspectiva global. Por ejemplo, BPMN utiliza actividades de interacción (OMG-BPMN 2011), UP-ColBPIP utiliza mensajes (Villarreal, Salomone & Chiotti 2007; Villarreal et al. 2010), UMM utiliza un concepto de transacciones comerciales (Huemer et al. 2008), y WS-CDL utiliza un concepto de interacciones (W3C-WSCDL 2005).
2. *Formalización del comportamiento de constructores de flujo de control complejos de lenguajes de PNCs*. Los PNCs requieren definir el flujo de control de la perspectiva global de las interacciones inter-organizacionales, el cual puede ser simple (tales como paralelismo y decisiones exclusivas) o complejo (tales como

cancelación, instancias múltiples, y sincronizaciones avanzadas de caminos en paralelo)

3. *Modularidad*, para facilitar la reutilización de modelos formales de constructores cuya semántica es idéntica o similar en distintos lenguajes de PNCs.
4. El lenguaje formal debería permitir la *verificación de propiedades de comportamiento* de modelos formales con flujos de control complejos. Este requerimiento es abordado en el capítulo siguiente.

Para dar soporte a estos requerimientos se propone el lenguaje formal *Redes de Interacción Global (Global Interaction Nets, o GI-Nets)*, el cual está basado en el formalismo de redes de Petri Jerárquicas y Coloreadas (Hierarchical and Colored Petri Nets, o HCP-Nets) (Jensen & Kristensen 2009). El lenguaje permite definir redes de Interacción Global que representan modelos formales de PNCs.

Una red de Interacción Global (GI-Net) es una red de Petri compuesta de una jerarquía de módulos de Interacción Global. Esta jerarquía de módulos define un árbol estructurado que permite formalizar modelos estructurados de PNCs.

Un *módulo de Interacción Global (módulo GI)* es el elemento utilizado para formalizar los constructores de interacciones y de flujo de control. Permite abstraer al lenguaje GI-Nets de las diferencias entre lenguajes de PNCs mencionadas en el requerimiento 1, y dar soporte a una representación jerárquica y modular. De esta manera, es posible formalizar los conceptos de interacciones y de flujo de control de acuerdo a la semántica específica de cada lenguaje de PNC.

Definición 3.1. ***(GI-Net)*** Dada una HCP-Net $GI_N = (S, SM, PS, FS)$ (Definición **0**.8) y un módulo CP-Net (red de Petri Coloreada) (Definición **0**.7) $r \in S$, GI_N es una *GI-Net* sii se cumplen las siguientes restricciones:

1. S es un conjunto finito y ordenado de módulos GI, estructurado como un árbol ordenado, tal que la raíz del árbol es el módulo $r \in S$,
2. para cada transición $t \in T^r$, t está asociada con un módulo GI por medio de la función de sub-módulo SM,
3. para cada módulo GI $s \in S$ tal que $s \neq r$, existe un único camino directo desde r a s,
4. $P_I \subset P^r$ es un conjunto no vacío de posiciones de interacción tal que para cada posición $p \in P_I$ el conjunto de colores de p es GINT,
5. $P_C \subset P^r$ es un conjunto de posiciones de control tal que para cada posición $p \in P_C$ el conjunto de colores de p es CTRL,
6. existe solamente una posición de interacción de entrada $ip \in P_I$ tal que $\bullet ip = \emptyset$,
7. existe solamente una posición de interacción de salida $op \in P_I$ tal que $op\bullet = \emptyset$, y op es miembro del conjunto de fusión $end \subset FS$,
8. para cada módulo $s \in S$ y cada elemento $n \in (P^s \cup T^s)$, n se encuentra en un camino directo desde ip a op,
9. $|p\bullet| = |\bullet p| = 1$, y para cada posición $p \subset P^r$ y transición $t \subset T^r$ tal que $p \neq ip$ y $p \neq op$, se cumple que $|p\bullet| = |\bullet p| = |t\bullet| = |\bullet t| = 1$,
10. Para cada $s_1, s_2 \in S$ y cada $p_1 \in P^{s_1}$, $p_2 \in P^{s_2}$ tal que $s_1 \neq s_2$, si p_1, p_2 son miembros del mismo conjunto de fusión $f \subset FS$ y $f \neq end$, entonces $Type[p_1] = Type[p_2] = CTRL$.

La Definición **3.1** expresa que una HCP-Net es una GI-Net sii:

(1) Una GI-Net es un árbol ordenado de módulos.

(2) Cada transición definida en el módulo r, el cual es la raíz del árbol, debe estar asociada con un módulo GI.

(3) Cada módulo GI está conectado a la raíz del árbol, es decir, existe un único camino directo desde el módulo raíz de la GI-Net a cada módulo.

(4 y 5) Una GI-Net está compuesta de dos tipos de posiciones: *posiciones de interacción* que tienen el color *GINT*, utilizadas para modelar estados de las interacciones en PNCs; y posiciones de control que tienen el color *CTRL*, utilizadas para restringir y controlar el flujo de control de las interacciones dentro de un módulo GI. Las señales en las posiciones de control representan restricciones en la red, y permiten modelar distintos tipos de flujos de control complejos de interacciones.

(6 y 7) El módulo raíz de la GI-Net tiene solamente una posición de interacción de entrada y una de salida. La posición de salida pertenece al conjunto de fusión *end*. Toda posición de este conjunto referencia a la posición de salida *op*.

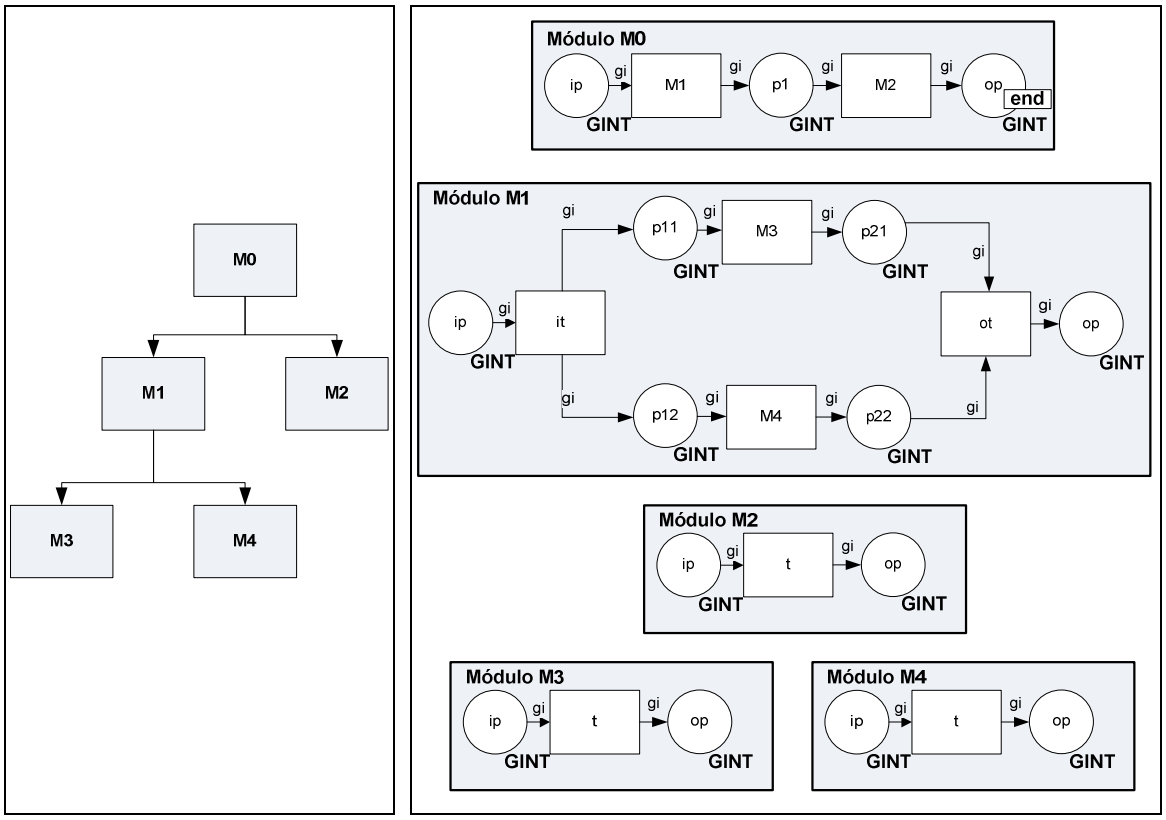
(8) Toda posición o transición definida en una GI-Net debe estar en un camino directo que va desde la posición de entrada a la posición de salida.

(9) El módulo r que es la raíz de una GI-Net debe estar compuesto de una secuencia de posiciones y transiciones.

(10) Si existen dos posiciones que pertenecen a dos módulos distintos y al mismo conjunto de fusión, siendo dicho conjunto distinto del conjunto de fusión *end*, entonces las posiciones deben ser de control, es decir, su color debe ser de tipo *CTRL*.

Definición 3.2. ***(Módulo GI)*** Un modulo CP-Net $GI_M = (CPN, T_{sub}, P_{port}, PT)$ es un *módulo GI*, donde $CPN = (P, T, A, \Sigma, V, C, G, E, I)$, si y sólo si se cumplen las siguientes restricciones:

1. $P_I \subset P$ es un conjunto no vacío de posiciones de interacción tal que para cada posición $p \in P_I$ el conjunto de colores de p es *GINT*,
2. $P_C \subset P$ es un conjunto de posiciones de control tal que para cada posición $p \in P_C$ el conjunto de colores de p es *CTRL*,
3. existe solamente una posición de entrada $ip \in P_I$ tal que $PT(ip) = IN$ y $\bullet ip = \emptyset$,
4. existe solamente una posición de salida $op \in P_I$ tal que $PT(op) = OUT$ y $\bullet op = \emptyset$,
5. para cada elemento $n \in (P \cup T)$ donde $n \neq op$, existe un camino directo desde ip a n.



a) Estructura de la GI-Net b) Módulos de la GI-Net

Fig. 3.1. Ejemplo de una GI-Net

La Figura **3.1** muestra un ejemplo de una GI-Net. La Figura **3.1** a) muestra una representación jerárquica de la GI-Net, esto es un árbol con cuatro módulos GI, siendo el módulo M0 la raíz de la GI-Net. M1 y M2 son sub-módulos de la raíz, mientras que M3 y M4 son sub-módulos de M1. M2, M3 y M4 son hojas del árbol y no tienen sub-módulos. Cada módulo GI está conectado a la raíz del árbol, es decir, existe un camino directo en la red desde cada módulo a la raíz.

La Figura **3.1** b) muestra la representación gráfica en términos de redes de Petri de la GI-Net, con cada uno de los módulos GI que la conforman. Cada módulo GI es en sí una red de Petri Coloreada definida por un conjunto de posiciones (círculos), transiciones (rectángulos), y arcos, los cuales determinan el comportamiento de la GI-Net. Los colores de las posiciones de los módulos de esta GI-Net son de tipo *GINT*, lo que indica que estas posiciones representan el flujo de interacciones. No hay módulos con posiciones de tipo *CTRL* en esta GI-Net, ya que no cuenta con elementos de flujos de control complejos.

Todos los módulos están conformados por una única posición de interacción de entrada y una única posición de interacción de salida, respetando la definición de módulo GI.

El módulo M0 es la raíz de la GI-Net y está formado por una secuencia de posiciones y transiciones que expresan el orden de ejecución en secuencia de los sub-módulos de la GI-Net. Tiene una única posición de entrada (*ip*) y una única posición de salida (*op*). La posición de salida *op* pertenece al conjunto de fusión *end*, indicado con una etiqueta *end* sobre dicha posición. Toda posición que pertenezca al conjunto de fusión *end* tiene una etiqueta *end* y referencia a la posición *op* del módulo M0, y funciona como si fuera una misma y única posición. Con lo cual, cuando se coloca una señal en una de las posiciones del conjunto de fusión *end*, aparece una señal en todas las demás posiciones que pertenecen a dicho conjunto. En el ejemplo hay una sola posición que pertenece a este grupo, y esa posición es la posición final *op*, dado que la GI-Net tiene un único camino o forma de finalizar.

Cada transición en el módulo M0 referencia a un módulo. La transición M1 referencia al módulo M1, mientras que la transición M2 referencia al módulo M2. El módulo M1 representa la ejecución en paralelo de los módulos M3 y M4, referenciados por las transiciones M3 y M4 de dicho módulo. La transición *it* representa la bifurcación de los dos caminos en paralelo, mientras que la transición *ot* representa la sincronización de los dos caminos. Todo módulo está conformado por una única posición de interacción de entrada y una única posición de interacción de salida.

Cada módulo tiene relaciones puerto-enlace (port-socket) (ver Definición **0**.7) con los módulos que tiene anidados. Por ejemplo, las posiciones *ip* y *p1* del módulo M0 tienen una relación puerto-enlace con las posiciones *ip* y *op* del módulo M1 respectivamente. Esto implica que si hay una señal en la posición *ip* de M0, también habrá una señal en la posición *ip* de M1 que habilitará el comienzo de M1. Cuando haya una señal en la posición *op* de M1, habrá una señal en la posición p1 de M0. Los demás módulos de la GI-Net están relacionados entre sí por medio de este tipo de relaciones puerto-enlace.

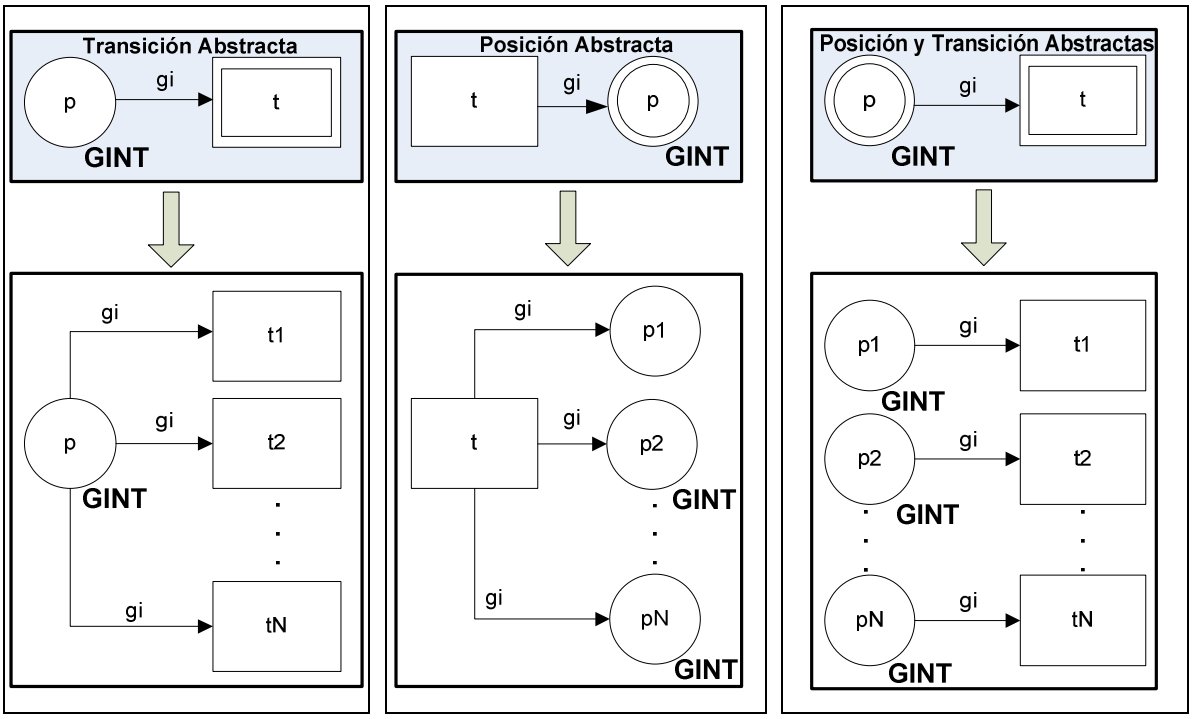
El comportamiento de ejecución de esta GI-Net es el siguiente: la GI-Net comienza en M0 cuando hay una señal en la posición *ip*, la que habilita la ejecución de la transición M1 que referencia al módulo M1. Cuando se inicia la ejecución de M1 la transición *it* deposita una señal en *p11* y *p12*, lo que permite que se ejecuten en paralelo los módulos M3

y M4. Cuando finaliza la ejecución de dichos módulos se deposita una señal en *p21* y *p22*, habilitando la transición *ot*. Cuando se ejecuta esta transición deposita una señal en la posición *op*, y de acuerdo a la relación puerto-enlace de los módulos M0 y M1, se deposita también la señal en la posición *p1*, por lo que queda habilitado el módulo M2 para su ejecución. Una vez que M2 finaliza su ejecución, es decir se ejecutó la transición *t*, se deposita una señal en la posición *op* del módulo M0, y finaliza la ejecución de la GI-Net.

3.1.1. Módulos Abstractos y Concretos

La formalización de un lenguaje de PNCs requiere la definición de una semántica formal para cada constructor del lenguaje. Un constructor o concepto de un lenguaje es un elemento abstracto utilizado para crear instancias de elementos de dicho constructor en un modelo de PNC. Un modelo de PNC contiene elementos que representan instancias de constructores del lenguaje usado. De esta manera, la definición formal de un constructor debería ser realizada también de manera general y abstracta, y debería permitir la representación formal de todas las posibles instancias de cada constructor.

Para dar soporte a este requerimiento se propone el concepto de *módulos GI abstractos*, que formalizan constructores de los lenguajes de PNC, y *módulos GI concretos*, que formalizan instancias de dichos constructores en los modelos de PNC. Un módulo GI abstracto define formalmente a un constructor de un lenguaje, y representa al conjunto de todos los posibles módulos GI concretos que formalizan las instancias de dicho constructor. Un módulo GI concreto define formalmente un elemento de un modelo de PNC, el cual se corresponde con un módulo GI abstracto que formaliza al constructor del elemento.



a) Transición abstracta b) Posición abstracta c) Posición y transición abstractas

Fig. 3.2. Representación de posiciones y transiciones abstractas

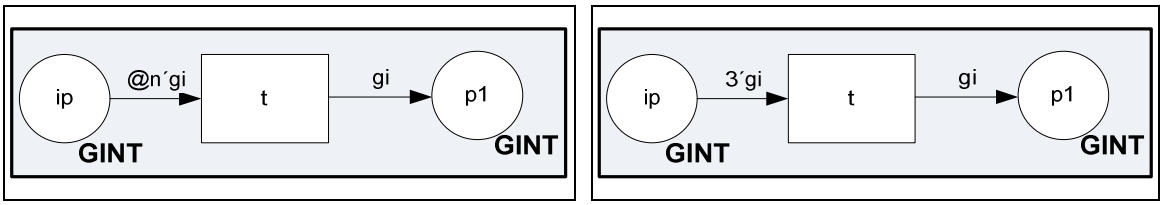
Debido a que los módulos abstractos deben ser definidos de manera general, es necesario utilizar expresiones como $p\bullet$, la cual representa el conjunto completo de transiciones de salida conectadas a la posición p. Sin embargo, este tipo de definiciones en redes de Petri sólo pueden ser realizadas mediante la notación algebraica, ya que la notación gráfica no las contempla.

Para solucionar este problema se proponen *posiciones* y *transiciones abstractas* (Figura **3.2**), las cuales representan a un conjunto de posiciones y transiciones respectivamente, y son definidas gráficamente con un borde de línea doble. La Figura **3.2** a) muestra en la parte superior una posición *p* conectada a una transición abstracta *t*. Esto representa a una única posición *p* que está conectada a un conjunto de transiciones t_1 a t_N, como se muestra en la parte inferior de dicha figura. La Figura **3.2** b) muestra en la parte superior una posición abstracta *p* conectada a una transición *t*. Esto representa a un conjunto de posiciones p_1 a p_N que están conectadas a una única transición *t*, como se muestra en la parte inferior de dicha figura. La Figura **3.2** c) muestra en la parte superior una posición abstracta *p* conectada a una transición abstracta *t*. Esto representa a un

conjunto de posiciones p_1 a p_N que están conectadas a un conjunto de transiciones t_1 a t_N, donde cada posición p_i se conecta a una transición t_i, para todo $0 < i \leq N$, como se muestra en la parte inferior de dicha figura. A continuación se definen formalmente estos conceptos.

Definición 3.3. ***(Posiciones y transiciones abstractas)*** Una posición abstracta P_a es un conjunto de posiciones tal que para una dada transición t, se cumple que $P_a \subseteq t\bullet$ o $P_a \subseteq \bullet t$, es decir, P_a está incluido en el conjunto de posiciones de salida o de entrada de t. De manera análoga, una transición abstracta T_a es un conjunto de transiciones tal que para una dada posición p, se cumple que $T_a \subseteq p\bullet$ o $T_a \subseteq \bullet p$, es decir, T_a está incluido en el conjunto de transiciones de salida o de entrada de p.

Los arcos conectados a posiciones y/o transiciones también pueden contener expresiones abstractas. Cuando se genera un módulo GI concreto, los arcos asociados a estas expresiones deben ser configurados con información específica del elemento de PNC que se está formalizando con el módulo GI concreto. Por ejemplo, un módulo abstracto podría tener un arco en el cual sea necesario indicar la cantidad de caminos paralelos que puede tener un constructor, o la cantidad de señales que son necesarias para que se habilite una determinada transición. Sin embargo, esta información no está disponible cuando se formaliza el constructor, sino recién cuando se formaliza un elemento de PNC que es instancia de dicho constructor. En este tipo de situaciones se define en el arco una expresión abstracta que debe ser definida por cada módulo GI concreto de acuerdo al elemento de PNC que es formalizado.



a) Expresión abstracta b) Expresión concreta

Fig. 3.3. Expresiones abstractas y concretas

Los elementos a definir en una expresión abstracta se indican con el signo @ (arroba) (Figura **3.3**). La Figura **3.3** a) muestra un ejemplo de un arco que contiene la expresión abstracta $@n´gi$ que representa la cantidad de señales que se necesitan para que se habilite la transición *t*, mientras que la Figura **3.3** b) muestra un arco en el cual se define dicha expresión, indicando que se necesitan tres señales en la posición *ip* para que la transición *t* sea habilitada.

Un módulo GI abstracto puede estar compuesto de posiciones abstractas, transiciones abstractas, o expresiones abstractas.

Definición 3.4. ***(Módulo GI abstracto)*** Un módulo de Interacción Global abstracto es un módulo de Interacción Global $GI_N^a = (CPN, T_{sub}, P_{port}, PT)$, donde $CPN = (P, T, A, \Sigma, V, C, G, E, I)$ (según Definición **0.6**) y puede existir un conjunto de posiciones abstractas $P_a \subset P$, un conjunto de transiciones abstractas $T_a \subset T$, y/o un conjunto de expresiones abstractas.

Un módulo GI concreto no puede estar compuesto de transiciones, posiciones, y/o expresiones abstractas.

Definición 3.5. ***(Módulo GI concreto)*** Un módulo de Interacción Global concreto es un módulo de Interacción Global $GI_N^c = (CPN, T_{sub}, P_{port}, PT)$, donde $CPN = (P, T, A, \Sigma, V, C, G, E, I)$ (según Definición **0.6**) tal que para cada posición $p \in P$ y transición $t \in T$, p no es una posición abstracta, t no es una transición abstracta, y además GI_N^c no contiene expresiones abstractas.

Si bien un módulo GI abstracto puede contener transiciones, posiciones, y expresiones abstractas, se podría dar la situación en la cual un módulo GI abstracto que formaliza a un dado constructor esté compuesto solamente de elementos concretos. Esto significaría que los módulos GI concretos son iguales para todas las posibles instancias que se pueden generar a partir de dicho constructor. Un módulo GI concreto no puede estar compuesto de posiciones ni de transiciones abstractas, y es derivado a partir de un módulo GI abstracto. Una GI-Net, que formaliza un modelo de PNC definido con un lenguaje de PNC no formal, está compuesta solamente de módulos GI concretos que formalizan los elementos del modelo de PNC.

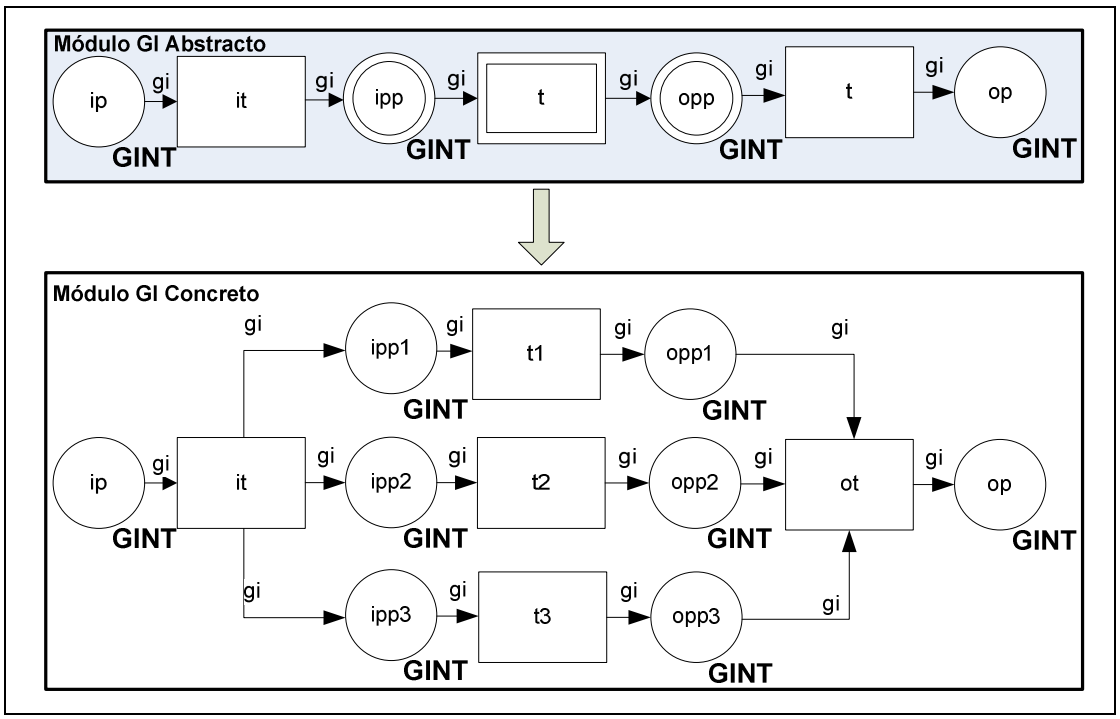


Fig. 3.4. Derivación de módulo GI concreto a partir de módulo GI abstracto

La Figura **3.4** muestra un ejemplo de un módulo GI concreto derivado de un módulo GI abstracto. El módulo abstracto combina posiciones y transiciones concretas con posiciones y transiciones abstractas. Las posiciones abstractas *ipp* y *opp* junto con la transición abstracta *t* representan que puede haber un determinado número de caminos en paralelo en los módulos concretos que se pueden derivar de la misma. El módulo concreto derivado tiene tres caminos en paralelo, donde las posiciones *ipp1*, *ipp2*, e *ipp3* fueron derivadas a partir de la posición abstracta *ipp*, las transiciones *t1*, *t2*, y *t3* fueron derivadas

de la transición abstracta *t*, y las posiciones *opp1*, *opp2*, y *opp3* fueron derivadas de la posición abstracta *opp*.

3.2. Generación de Modelos Formales de PNCs

La formalización de modelos o especificaciones de PNC con GI-Nets requiere de:

(1) La formalización de los constructores del lenguaje utilizado para definir los modelos o especificaciones de PNC. Esto se realiza definiendo los módulos GI abstractos que formalizan cada constructor de dicho lenguaje.

(2) Generación de las GI-Nets. Para cada modelo o especificación de PNC, se define la GI-Net que formaliza dicho PNC, de acuerdo a los módulos abstractos definidos.

Para dar soporte a la generación automática de GI-Nets, se propone un patrón formal de transformación para GI-Nets (Figura **3.5**). Este patrón permite transformar un modelo o especificación de PNC de entrada (definido con un lenguaje no formal) en un modelo de PNC de salida expresado como una GI-Net. Dicho patrón de transformaciones fue definido siguiendo los principios del desarrollo dirigido por modelos propuesto en (Kurtev 2005). Una *transformación de modelos* es el proceso de convertir, a través de una máquina de transformación, un modelo de entrada basado en un metamodelo de origen, en un modelo de salida basado en un metamodelo de destino, utilizando una determinada *especificación de transformación* (Iacob, Steen & Heerink 2008).

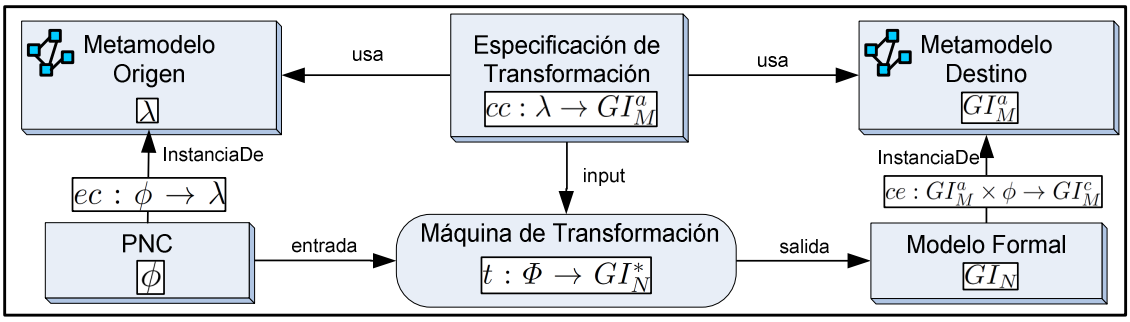


Fig. 3.5. Patrón de transformaciones para GI-Nets

El patrón de transformaciones para GI-Nets provee una *máquina de transformación* definida por la función t, cuya entrada es un modelo de PNC ϕ y cuya salida es un *modelo*

formal de PNC representado por la GI-Net GI_N. La función t está compuesta de las funciones internas ec, cc, y ce las cuales permiten llevar a cabo el proceso de transformación. La función ec asocia cada elemento del PNC ϕ con su correspondiente constructor en el metamodelo de origen. La especificación de transformación para GI-Nets utiliza como metamodelo de origen el correspondiente al lenguaje de PNC de origen, y como metamodelo de destino el representado por el conjunto de módulos GI abstractos que formalizan los constructores del lenguaje de entrada. La especificación de transformación está compuesta de *reglas de transformación* definidas por la función cc, la cual asocia cada constructor del metamodelo de origen con un módulo GI abstracto que define la representación formal de dicho constructor. Finalmente, la función ce define una asociación entre cada módulo GI abstracto definido en el metamodelo de destino con un conjunto de módulos GI concretos del modelo formal GI-Net GI_N.

El hecho de que la función t genera una GI-Net estructurada implica que el PNC de entrada ϕ de la función t debe ser estructurado (Capítulo 2, Sección 2.3.2). A continuación, se presenta la definición formal de *Especificación de Transformación*, la cual está expresada de manera general, ya que permite la definición de especificación de transformaciones para GI-Nets, así como también especificaciones de transformación para cualquier otro lenguaje.

Definición 3.6. ***(Especificación de transformación)*** Una Especificación de Transformación es una tupla $TE = (\lambda_s, \lambda_t, cc)$, donde:

1. λ_s es un conjunto finito de constructores que define el metamodelo del lenguaje de origen,
2. λ_t es un conjunto finito de constructores que define el metamodelo del lenguaje de destino,
3. $cc: \lambda_s \rightarrow \lambda_t$ es una función de relación biyectiva que define una asociación de un conjunto de constructores $C_s \subset \lambda_s$ a un conjunto de constructores $C_t \subset \lambda_t$.

En base a la definición de *Especificación de Transformación* se define formalmente un Patrón de Transformación para GI-Nets.

Definición 3.7. ***(Patrón de transformación para GI-Nets)*** Dada una especificación de transformación $TE = (\lambda, GI_M^a, cc)$, un patrón de transformación para GI-Nets es una tupla $T = (TE, \Phi, \phi, GI_M^c, GI_N^*, ec, ce, GI_N, t)$, donde:

1. λ es un conjunto finito de constructores que define el metamodelo de un lenguaje de PNCs,
2. GI_M^a es un conjunto finito de módulos GI abstractos donde cada módulo GI abstracto $AM \in GI_M^a$ formaliza un constructor $c \in \lambda$,
3. Φ es el conjunto de todos los posibles PNCs que pueden ser definidos a partir de λ,
4. ϕ es un PNC definido por un conjunto finito de elementos tal que $\phi \in \Phi$,
5. GI_M^c es el conjunto de todos los posibles módulos GI concretos que pueden ser generados a partir de cada modulo GI abstracto $AM \in GI_M^a$,
6. GI_N^* es el conjunto de todas las posibles GI-Nets que formalizan el conjunto de modelos de PNCs Φ tal que para cada PNC $\phi \in \Phi$ existe exactamente una GI-Net $GI_N \in GI_N^*$ que formaliza a ϕ, y cada GI-Net $GI_N \in GI_N^*$ está asociada exactamente a un modelo de PNC $\phi \in \Phi$,
7. $ec: \phi \to \lambda$ es una *función de relación* sobreyectiva que define una asociación entre un elemento $e \in \phi$ y un constructor $c \in \lambda$,
8. $cc: \lambda \to GI_M^a$ es una *función de relación* biyectiva que define una asociación entre un conjunto de constructores $C \subset \lambda$ con un módulo GI abstracto $AM \in GI_M^a$,
9. $ce: GI_M^a \times \phi \to GI_M^c$ es una *función de relación* inyectiva que define una asociación entre un módulo GI abstracto $AM \in GI_M^a$ y un módulo GI concreto $CM \in GI_M^c$,
10. GI_N es una GI-Net estructurada tal que $GI_N \in GI_N^*$, y está compuesta de un conjunto de módulo GI concretos $CM \subset GI_M^c$,
11. $t: \Phi \to GI_N^*$ es una *función de mapeo* biyectiva que define una transformación desde un modelo de PNC $\phi \in \Phi$ a una GI-Net $GI_N \in GI_N^*$ por medio de la composición de funciones $ce \circ cc \circ ec$.

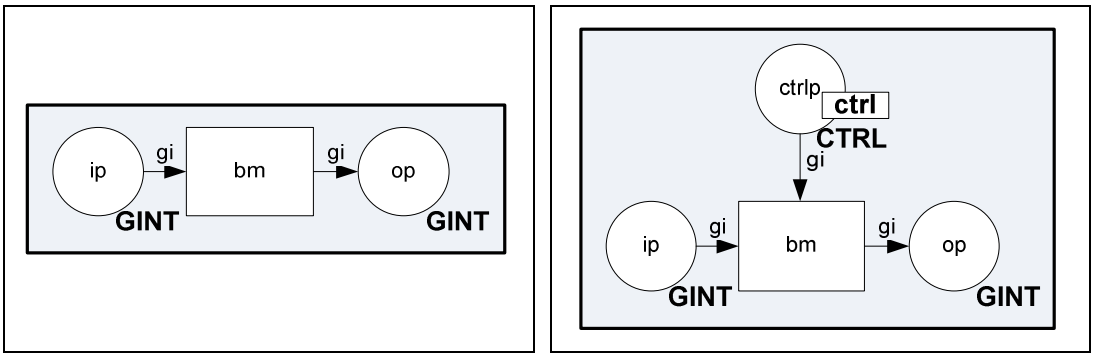
De esta manera, este patrón formal de transformación puede ser utilizado para definir y materializar un patrón de transformación específico que posibilite generar modelos formales GI-Nets a partir de modelos o especificaciones no formales de PNCs.

3.3. Formalización del Lenguaje UP-ColBPIP

En esta sección se presenta la formalización del lenguaje UP-ColBPIP mediante GI-Nets. A continuación se presentan los módulos GI abstractos que formalizan los siguientes constructores de UP-ColBPIP: *Business Message*, *Interaction Path*, *Xor*, *And*, *Or*, *Loop-While*, *Loop-Until*, *Multiple Instances, Exception*, *Cancel*, y *Termination*. En el Anexo A se muestran ejemplos de módulos GI concretos para estos constructores.

3.3.1. Business Message

La Figura **3.6** muestra el módulo GI abstracto del constructor *Business Message*, el cual en UP-ColBPIP representa una interacción. El *Business Message* se representa de dos maneras distintas de acuerdo a si se encuentra o no dentro del alcance de una excepción o cancelación. En caso afirmativo, su módulo abstracto se representa como se muestra en la Figura **3.6** a), donde las posiciones *ip* y *op* representan el inicio y fin del módulo respectivamente y la transición *bm* representa el envío del mensaje de negocio. En cambio, si se encuentra dentro del alcance de una excepción o cancelación su módulo abstracto se representa como se muestra en la Figura **3.6** b), donde se adiciona la posición *ctrlp* que pertenece al conjunto de fusión *ctrl*, haciendo referencia de esta manera a la posición *ctrlp* del módulo de *Exception* o de *Cancel* (descriptos luego en esta sección). Esta posición permite bloquear la ejecución de un mensaje en caso que se esté ejecutando una excepción o cancelación.



a) Business Message b) Business Message dentro de una excepción

Fig. 3.6. Módulo GI abstracto del constructor Business Message

3.3.2. Interaction Path

La Figura **3.7** muestra el módulo GI abstracto de *Interaction Path*, que en UP-ColBPIP representa una secuencia de mensaje y/o de elementos de flujo de control. Las posiciones *ip* y *op* representan el inicio y fin del módulo respectivamente y la transición *ip* representa una secuencia de interacciones.

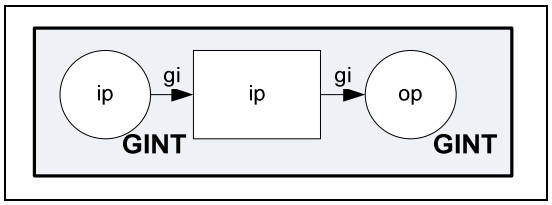


Fig. 3.7. Módulo GI abstracto del Interaction Path

3.3.3. Xor

Un constructor *Xor* de UP-ColBPIP representa que sólo uno de un conjunto de caminos de interacción alternativos puede ser ejecutado. La Figura **3.8** muestra el módulo GI abstracto que formaliza el constructor *Xor*. Este módulo consiste de dos posiciones *ip* y *op* que representan la entrada y salida del módulo respectivamente. Tiene además una

transición abstracta *t* que conecta a las posiciones mencionadas. Esta transición representa a todos los caminos posibles de ejecución del *Xor* que son mutuamente excluyentes.

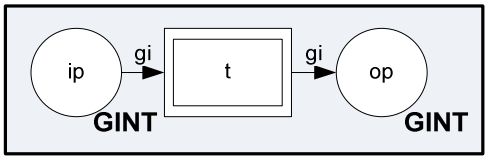


Fig. 3.8. Módulo GI abstracto del constructor Xor

Si hay una señal en la posición *ip*, todas las transiciones representadas por *t* son habilitadas y sólo una de ellas puede ser ejecutada. Una vez ejecutada una transición que consumió la señal de *ip*, las demás quedan inhabilitadas debido a que no hay más señales que las habiliten, representando de esta manera la exclusión mutua entre los caminos de interacción. Cuando finaliza su ejecución se coloca una señal en la posición de salida *op*, representando la unión de los caminos alternativos.

3.3.4. And

Un constructor *And* de UP-ColBPIP representa la ejecución paralela de caminos de interacción, tanto la bifurcación como sincronización de los mismos. La sincronización se lleva a cabo luego que finalizaron todos los caminos paralelos.

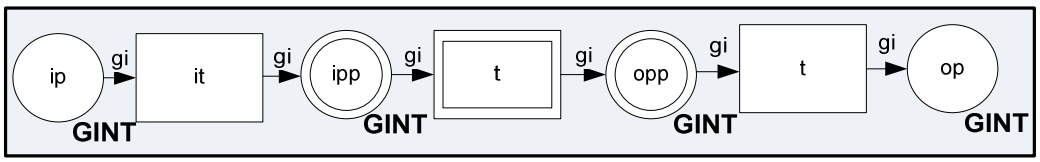


Fig. 3.9. Módulo GI abstracto del constructor And

La Figura **3.9** muestra el módulo GI abstracto del constructor *And*. Las posiciones *ip* y *op* representan la entrada y salida del módulo respectivamente. Las transiciones *it* y *ot* representan la bifurcación y sincronización de los caminos paralelos. Las posiciones abstractas *ipp* y *opp* junto con la transición abstracta *t* permiten representar a todos los posibles caminos de interacción paralelos del *And*.

Si hay una señal en la posición *ip*, se habilita la transición *it*. La ejecución de dicha transición coloca una señal en todas las posiciones representadas por *ipp*, indicando de esta manera el paralelismo de los caminos de interacción. La finalización de la ejecución de todos los caminos se logra cuando existe una señal en cada posición representada por *opp*. Una vez que finaliza la ejecución de todos los caminos, se habilita la transición *ot* que permite llevar a cabo la sincronización y depositar una señal en la posición de salida *op*.

3.3.5. Or/Synchronizing Merge

El constructor *Or* con sincronización *Synchronizing Merge* de UP-ColBPIP representa la ejecución de dos o más caminos de interacción alternativos no excluyentes, donde todos los caminos que se ejecuten deben ser sincronizados. La ejecución de más de un camino implica paralelismo. La Figura **3.10** muestra el módulo GI abstracto de este constructor. Este módulo contiene dos posiciones *ip* y *op* que representan la entrada y salida del módulo respectivamente, y dos transiciones *it* y *ot* que representan la bifurcación y sincronización del *Or* respectivamente. Las posiciones abstractas *ipp* y *opp* junto con la transición abstracta *t* permiten representar a todos los posibles caminos de interacción alternativos del *Or*.

La transición *it* está conectada a las posiciones abstractas *ipp* y *opp* por medio de dos arcos. La expresión @*cond* en estos arcos indica una variable booleana que debe ser evaluada para determinar si se deposita una señal en la posición *ipp* o en la posición *opp*. La transición *it* deposita una señal en la posición *ipp* si la evaluación de la expresión @*cond* del arco que conecta a *it* con *ipp* retorna falso. Esto representa la ejecución de uno de los caminos del *Or*. Por otro lado, si la evaluación de la expresión @*cond* del arco que conecta a *it* con *opp* retorna verdadero se deposita una señal en la posición *opp*. Esto representa que no se ejecuta uno de los caminos del *Or*.

Cada módulo GI concreto que formaliza a un elemento que es instancia del constructor *Or/Synchronizing Merge* debe tener definida una variable booleana distinta por cada camino de interacción que tenga el elemento formalizado por dicho módulo.

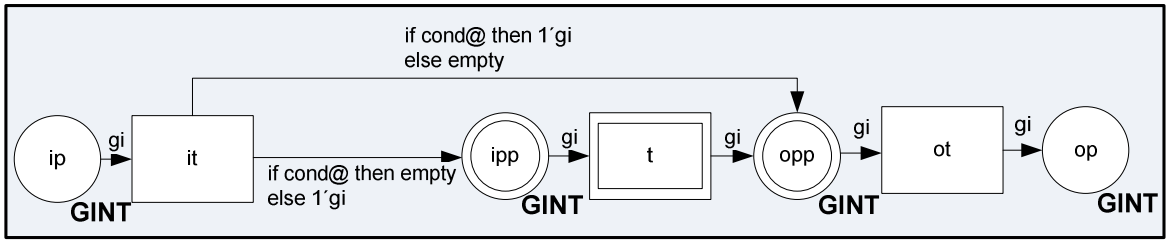


Fig. 3.10. Módulo GI abstracto del constructor Or con Synchronizing Merge

Si hay una señal en la posición *ip*, se puede ejecutar la transición *it*. Si las variables booleanas representadas por *cond@* retornan verdadero, se deposita una señal en las posiciones representadas por *ipp*, y las posiciones representadas por *opp* quedan vacías. Esto representa la ejecución de todos los caminos de interacción que forman parte del *Or*. En cambio, si las variables representadas por *cond@* retornan falso, se deposita una señal en las posiciones representadas por *opp*, y las posiciones representadas por *ipp* quedan vacías. Esto representa la no ejecución de los caminos de interacción que forman parte del *Or*. Las variables representadas por *cond@* son independientes entre sí, lo que permite que se ejecuten uno, más de uno, o ninguno de los caminos del *Or*. Una vez que finaliza la ejecución de los caminos, se habilita la transición *ot* que permite llevar a cabo la sincronización y depositar una señal en la posición final *op*.

3.3.6. Or/N-out-of-M

El constructor *Or* con sincronización *N-out-of-M* de UP-ColBPIP representa la ejecución de dos o más caminos de interacción alternativos, donde *N* de un total de *M* caminos deben ser sincronizados, y el resto deben ser descartados. La Figura **3.11** muestra el módulo GI abstracto de este constructor. Las posiciones de entrada y salida son *ip* y *op* respectivamente. La parte de la bifurcación que representa al *Or* es análoga a la descripta para el constructor *Or/SyncMerge*. Esta parte incluye a los elementos *ip*, *it*, *ipp*, *t*, y *opp*.

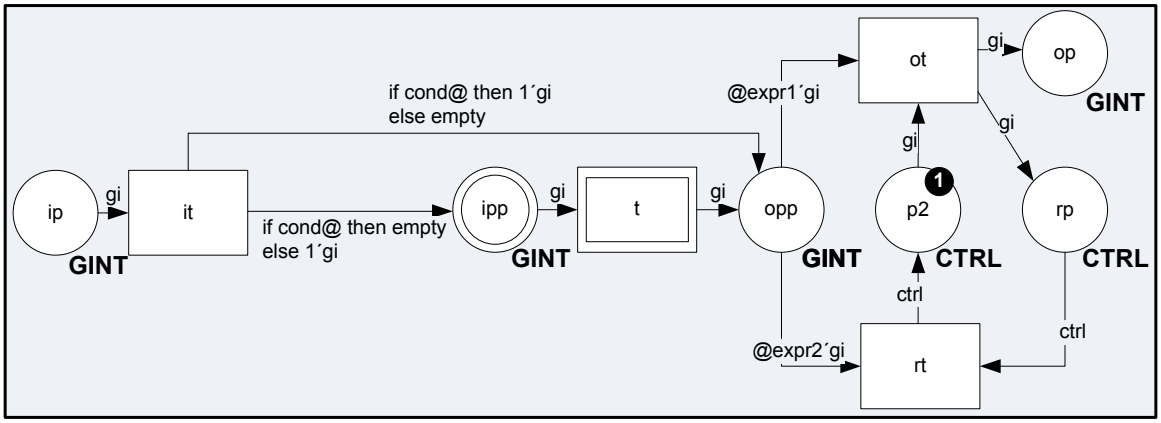


Fig. 3.11. Módulo GI abstracto del constructor Or con N-out-of-M

La sincronización *N-out-of-M* se representa por el conjunto de posiciones y transiciones que siguen a continuación de la transición abstracta *t*, la cual representa a los caminos de interacción del *Or*. En la posición *opp* se depositan todas las señales de los caminos paralelos que deben ser sincronizados. La transición *ot* se habilita si se cumple la expresión @*expr1*, mientras que la transición *rt* se habilita si se cumple la expresión @*expr2*. Las expresiones @*expr1* y @*expr2* indican la cantidad de señales que debe haber en la posición *opp* para que se habiliten dichas transiciones. En este caso debe haber al menos *n* señales para que se habilite *ot* y al menos *m-n* señales para que se habilite *rt*, donde *m* representa la cantidad máxima de caminos en paralelo y *n* la cantidad de caminos que deben sincronizarse. Cada módulo GI concreto que formaliza a un elemento que es instancia del constructor *Or/N-out-of-M* debe definir la expresión @*expr1* en base a la cantidad de caminos que deben ser sincronizados (*n*) y la expresión @*expr2* en base a la cantidad de caminos que deben ser descartados (*m-n*).

Las posiciones de tipo *CTRL p2* y *rp* y la transición *rt* se utilizan para controlar que se lleve a cabo la sincronización de los *n* caminos y que se descarten los *m-n* caminos que no deben ser sincronizados. La posición *p2* tiene en el estado inicial una señal indicada con un círculo negro y el número uno.

3.3.7. Or/Discriminator

El constructor *Or* con sincronización *Discriminator* de UP-ColBPIP representa la ejecución de dos o más caminos de interacción alternativos, donde *1* de un total de *M* caminos deben ser sincronizados, y los restantes deben ser descartados. La sincronización *Discriminator* es un caso especial del *N-out-of-M* donde la cantidad de caminos a sincronizar es uno (*n=1*). Para ello, es necesario modificar el arco que va de la posición *opp* a la transición *ot* en el módulo GI abstracto de la Figura **3.11**. Este arco debe contener la expresión *gi* en lugar de *@expr1´gi*, indicando de esta manera que basta con que haya una señal en la posición *opp* para que se habilite la transición *ot*.

3.3.8. Loop

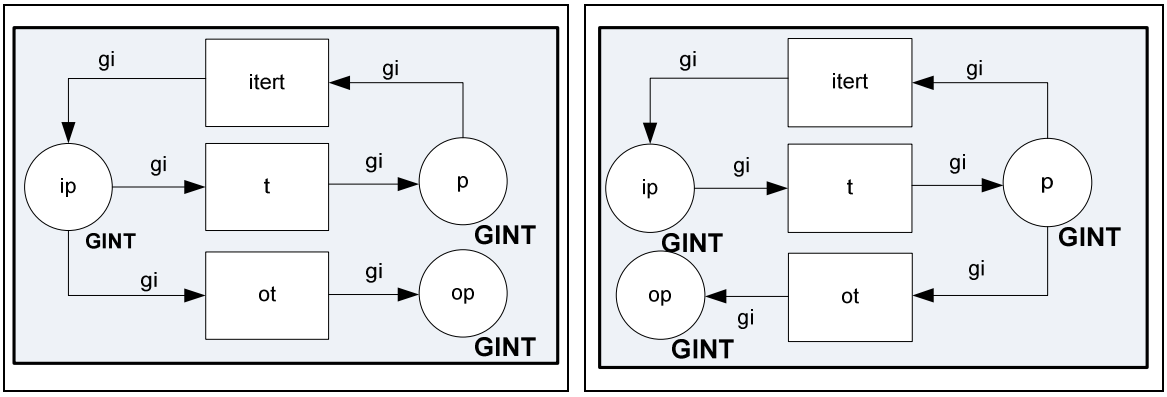
El constructor *Loop* de UP-ColBPIP representa un camino de interacción que se puede ejecutar varias veces. En un *Loop* de tipo *While*, el camino de interacción se va a ejecutar mientras se cumpla una determinada condición, es decir, cero o más veces. En un *Loop* de tipo *Until* el camino de interacción se va a ejecutar hasta que se cumpla una determinada condición, es decir, una o más veces.

La Figura **3.12** muestra los módulo GI abstractos de los constructores *Loop While* (Figura **3.12** a)) y *Loop Until* (Figura **3.12** b)). Estos módulos consisten de las posiciones *ip* y *op* que representan la entrada y salida del módulo respectivamente. La transición *t* representa al contenido del *Loop*, es decir, los elementos que se van a ejecutar de manera iterativa. La posición *p* y la transición *itert* permiten devolver el flujo de control al inicio del ciclo. Finalmente, la transición *ot* finaliza las iteraciones y representa la salida del *Loop*.

La diferencia entre ambos módulos radica en que en el *Loop Until* la transición *ot* que finaliza el ciclo está conectada a la posición *p*, lo que permite que el contenido del ciclo se ejecute al menos una vez antes de la finalización del mismo. En cambio, en el *Loop While* la transición *ot* está conectada a la posición de inicio *ip*, lo que implica que el contenido del ciclo puede que no sea ejecutado.

Como este módulo GI abstracto no posee ni transiciones ni posiciones abstractas todos los módulos GI concretos derivados del mismo van a ser iguales al mostrado en la

Figura **3.12**. Es decir, todas las instancias de *Loop While* y *Loop Until* serán formalizadas de la misma manera de acuerdo a los módulos que se muestran en dicha figura.



a) Loop While b) Loop Until

Fig. 3.12. Módulo GI del constructor Loop

3.3.9. Multiple Instances

El constructor *Multiple Instances* representa la ejecución en paralelo de instancias múltiples de un camino de interacción y su sincronización. El número de instancias a ejecutarse y sincronizarse puede ser conocido en tiempo de diseño, en tiempo de ejecución o bien no conocido. En UP-ColBPIP existe un constructor *Multiple Instances* para cada una de estas variantes. La Figura **3.13** muestra el módulo GI abstracto del constructor *Multiple Instances*, el cual es aplicable cuando se conocen el número de instancias en tiempo de diseño. Este módulo contiene dos posiciones *ip* y *op* que representan la entrada y salida del módulo respectivamente, y dos transiciones *it* y *ot* que representan la bifurcación y sincronización de las instancias múltiples respectivamente. El arco que va desde la transición *it* a la posición *ipp* representa una bifurcación que permite generar distintas instancias, y el que va desde la posición *opp* a la transición *ot* representa la sincronización de dichas instancias. Esto se representa mediante la expresión abstracta @´gi. Cada módulo GI concreto debe definir esta expresión de acuerdo a la cantidad de instancias que se quieren representar. La transición *t* representa al camino de interacción que se va a ejecutar con instancias múltiples.

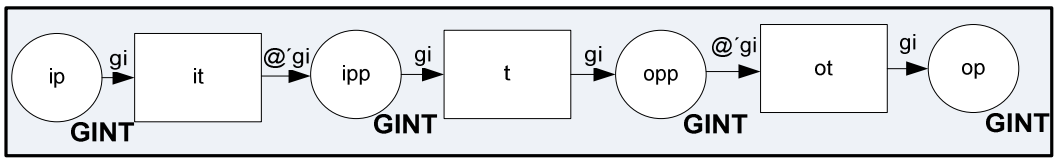


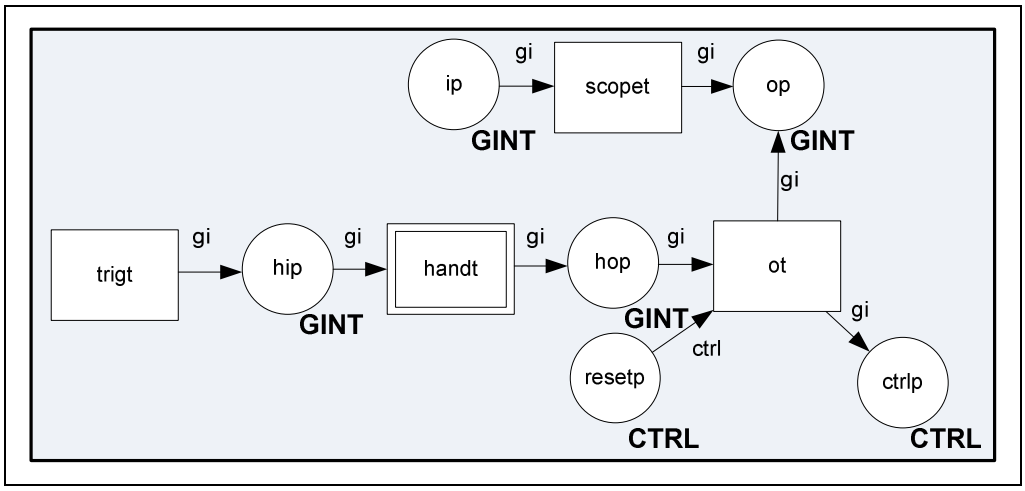
Fig. 3.13. Módulo GI abstracto del constructor Multiple Instances

3.3.10. Exception

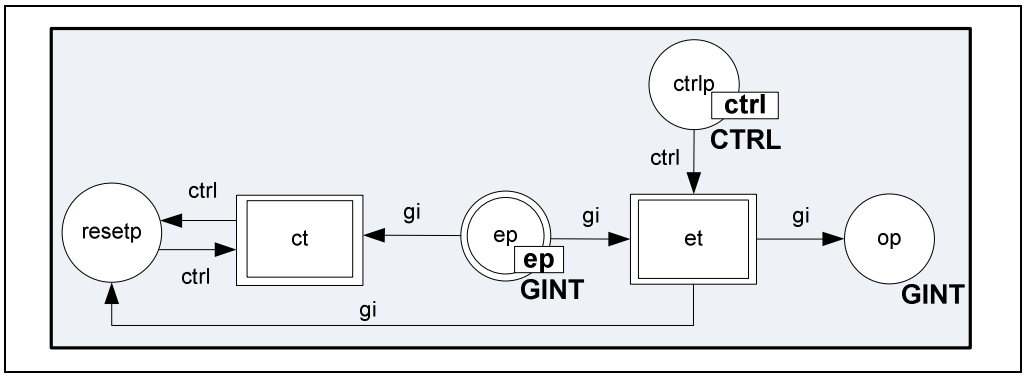
El constructor *Exception* de UP-ColBPIP representa el manejo de excepciones. Consiste de un camino de interacción que indica el alcance de la excepción, y de uno o más caminos de interacción que representan los manejadores de excepciones que capturan y gestionan excepciones que ocurren dentro del alcance de la misma. Un camino manejador de excepción define el camino a seguir luego que ocurre el tipo de excepción que captura. Luego que la excepción fue manejada, el proceso continua su ejecución. La Figura **3.14** muestra el módulo GI abstracto del constructor *Exception*. El módulo está divido en dos partes, el constructor *Exception* (Figura **3.14** a)) y el disparador (Trigger) de las excepciones (Figura **3.14** b)).

El módulo Exception contiene las posiciones *ip* y *op* que representan la entrada y salida del módulo respectivamente, y la transición *scopet* que representa un módulo que contiene a todos los elementos que están dentro del alcance de la excepción. Si no ocurre ningún evento de excepción, el módulo finaliza y deposita una señal en la posición final *op*. La transición *trigt* representa al disparador de la excepción y referencia al módulo Trigger (Figura **3.14** b)), el cual no es independiente, sino que es parte del módulo Exception. Por razones de modularidad y claridad para explicar el comportamiento del mismo, se lo presenta en una figura separada. La transición abstracta *handt* representa a todos los manejadores de excepciones definidos en el constructor *Exception*.

Las posiciones de tipo *CTRL resetp* y *ctrlp* son utilizadas para dirigir el flujo de control cuando se ejecuta una excepción. La posición *ctrlp* es utilizada para bloquear el flujo de control dentro del alcance de una excepción mientras se ejecuta el manejador de excepciones. En cambio, la posición *resetp* es utilizada para bloquear el disparo de nuevas excepciones cuando se está ejecutando el manejador de excepciones.



a) Módulo Exception



b) Módulo Trigger

Fig. 3.14. Módulo GI abstracto del constructor Exception

La Figura **3.14** b) muestra la formalización de un disparador de excepciones (Trigger). La posición abstracta *ep* es el punto de entrada a este módulo debido a que referencia a todos los puntos de excepción dentro del alcance del constructor *Exception*. Esto es posible debido a que *ep* pertenece al conjunto de fusión *ep* indicado con etiqueta *ep* sobre dicha posición. La posición de salida *op* está asociada a la posición *hip* del módulo Exception, con lo cual si hay una señal en la posición *op*, también habrá una señal en la posición *hip*. Esto permite habilitar al manejador de excepciones representado por la transición abstracta *handt*. Las posiciones *resetp* y *ctrlp* referencian a las posiciones análogas del módulo Exception. La transición abstracta *et* permite ejecutar la excepción depositando una señal en la posición *op*, mientras que la transición abstracta *ct* se encarga

de limpiar todas las señales que quedan en las posiciones dentro del alcance de la excepción luego que se ejecuta dicha excepción.

3.3.11. Cancel

El constructor *Cancel* de UP-ColBPIP representa el camino a seguir cuando ocurre una determinada excepción. Su comportamiento es similar al del *Exception*, pero se diferencia de éste en que luego del manejo de la excepción el PNC finaliza su ejecución. La Figura **3.15** muestra el módulo GI abstracto del constructor *Cancel*. Este módulo está divido en dos partes como ocurre con el módulo del constructor *Exception*: el módulo Cancel (Figura **3.15**) y el módulo Trigger (Disparador) de las excepciones. El módulo Cancel es similar al módulo Exception (Figura **3.14** a)). La única diferencia está en la posición de salida de la transición *ot*, ya que en este caso, una vez que se ejecuta la cancelación, el proceso debe finalizar, con lo cual el Cancel tiene una posición *endp* que referencia a la posición de finalización de la GI-Net. El módulo Trigger es el mismo que el definido para el constructor *Exception* en la Figura **3.14** b).

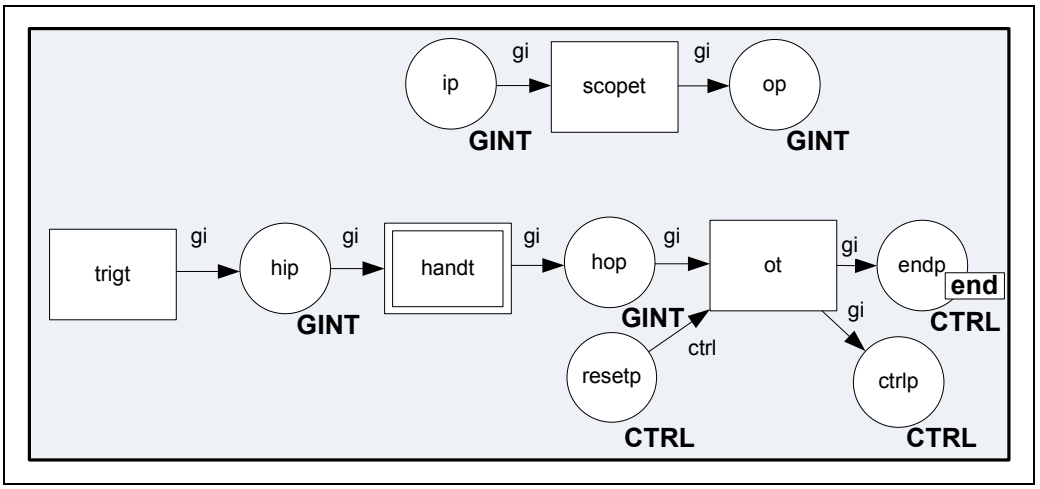


Fig. 3.15. Módulo GI abstracto del constructor Cancel

3.3.12. Termination

La Figura **3.16** muestra el módulo GI abstracto del constructor *Termination*, el cual está compuesto de las posiciones *ip* y *endp*. La posición *endp* representa que el protocolo de interacción llegó a su fin, y pertenece al conjunto de fusión *end* de una GI-Net. Todas las posiciones que pertenezcan a este conjunto de fusión están asociadas entre sí y representan a una única posición que es la posición final del protocolo, permitiendo de esta manera que haya una única posición de fin en todo el protocolo de interacción.

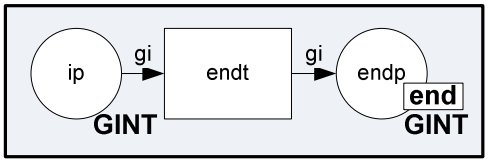


Fig. 3.16. Módulo GI abstracto del constructor Termination

3.4. Formalización del Lenguaje BPMN

En esta sección se presenta la formalización del lenguaje BPMN mediante GI-Nets. En la formalización se consideran los siguientes constructores de BPMN: *Choreography Task, Intermediate Event, End Event, Parallel Gateway, Exclusive Gateway, Event-Based Gateway, Inclusive Gateway,* y *Complex Gateway*. Los constructores *Exclusive Gateway* y *Event-Based Gateway* no difieren en su semántica de comportamiento desde el punto de vista del flujo de control, por lo que en la formalización sólo se hace referencia al *Exclusive Gateway*, pero los resultados se aplican para ambos constructores por igual. El constructor *Complex Gateway* es un caso particular, ya que no tiene una semántica específica de comportamiento asignada en BPMN. En esta tesis se lo utiliza para representar una sincronización parcial donde un número *n* de un total de *m* caminos que deben ser sincronizados.

Debido a que BPMN es un lenguaje no estructurado, sus constructores para bifurcación/decisión y sincronización/unión (gateways) permiten la definición de procesos no estructurados. Por lo tanto, para formalizar los constructores de BPMN con GI-Nets, es

necesario definirlos de manera estructurada, es decir, se debe definir un constructor estructurado a partir de cada posible combinación de un gateway de bifurcación/decisión con un gateway de sincronización/unión.

Tabla 3.1. Constructores estructurados de BPMN

	Parallel	**Exclusive**	**Inclusive**	**Complex**
Parallel	Parallel/Parallel	Parallel/Exclusive	Parallel/Inclusive	Parallel/Complex
Exclusive	Exclusive/Parallel	Exclusive/Exclusive	Exclusive/Inclusive	Exclusive/Complex
Inclusive	Inclusive/Parallel	Inclusive/Exclusive	Inclusive/Inclusive	Inclusive/Complex

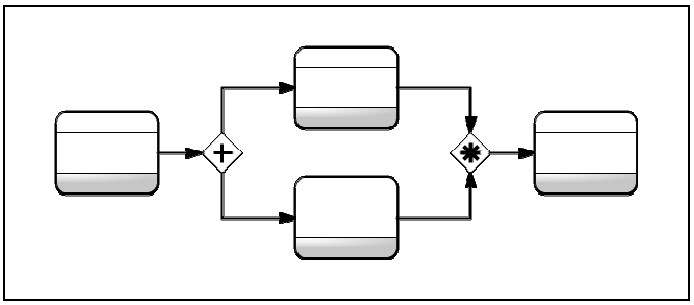


Fig. 3.17. Constructor estructurado de BPMN con los gateways Parallel/Complex

La Tabla **3.1** muestra los constructores estructurados de BPMN que se forman a partir de las combinaciones de los gateways de BPMN mencionados. Las filas representan los gateways de bifurcación o decisión, mientras las columnas representan a los gateways de sincronización o unión. La intersección de cada fila y columna determina un *constructor estructurado de BPMN*, el cual está compuesto de dos gateways, el de bifurcación/decisión y el de sincronización/unión. La Figura **3.17** muestra un ejemplo del constructor estructurado *Paralell/Complex* que surge de la intersección de la primer fila y última columna de la Tabla **3.1**.

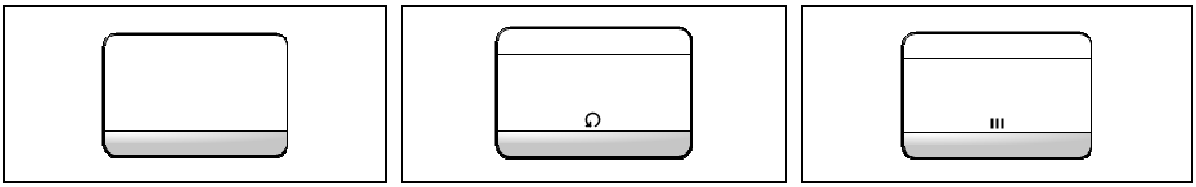
3.4.1. Formalización Reutilizando Módulos GI Abstractos

De acuerdo al análisis realizado acerca de la semántica de los constructores de BPMN y de UP-ColBPIP, se determinó que existen constructores cuyas semánticas de comportamiento son similares. Para estos constructores los módulos GI abstractos definidos para UP-ColBPIP pueden ser reutilizados para formalizar constructores en BPMN y

viceversa. Dentro de este grupo de constructores se encuentran *Choreography Task*, *Intermediate Event*, *End Event, Sequence* Flow y algunas combinaciones de los constructores *Parallel Gateway, Exclusive Gateway, Event-Based Gateway, Inclusive Gateway,* y *Complex Gateway*. La Tabla **3.2** muestra los constructores de flujo de control estructurados de BPMN y UP-ColBPIP que comparten la misma definición formal basada en GI-Nets. La Figura **3.18** muestra la representación gráfica de estos constructores en BPMN.

Tabla 3.2. Constructores de BPMN y UP-ColBPIP con la misma semántica de comportamiento

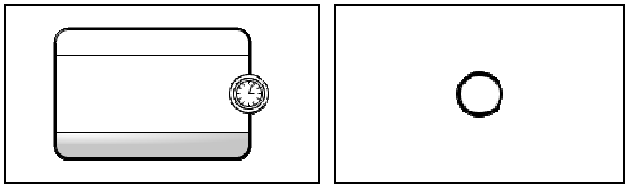
BPMN	UP-ColBPIP
Choreography Task	Business Message
Choreography Task (LoopType=Standard)	Loop
Choreography Task (LoopType=MI Parallel)	Multiple Instances
Intermediate Event	Exception
End Event	Termination
Parallel/Parallel	And
Parallel/Inclusive	And
Exclusive/Exclusive	Xor
Exclusive/Inclusive	Xor
Inclusive/Inclusive	Or/SyncMerge
Inclusive/Complex	Or/N-out-of-M



a) Choreography Task

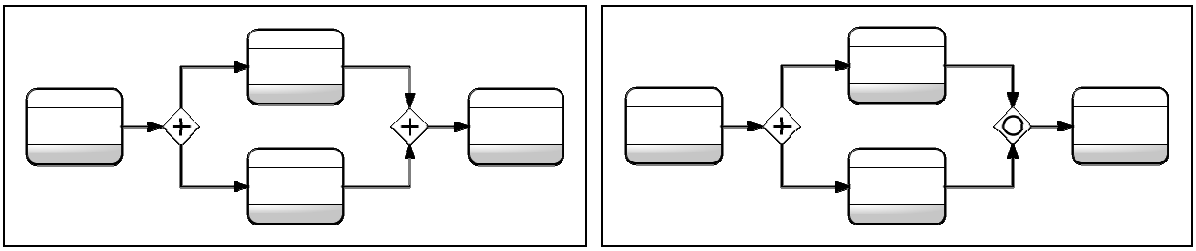
b) Choreography Task (LoopType=Standard)

c) Choreography Task (LoopType=MI Parallel)



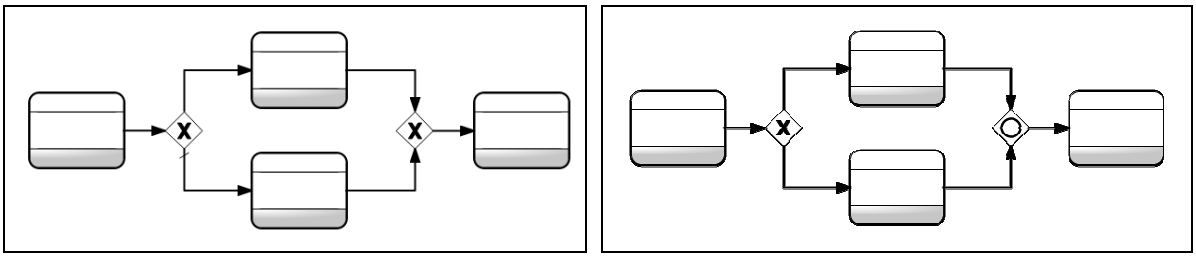
d) Intermediate Event

e) End Event



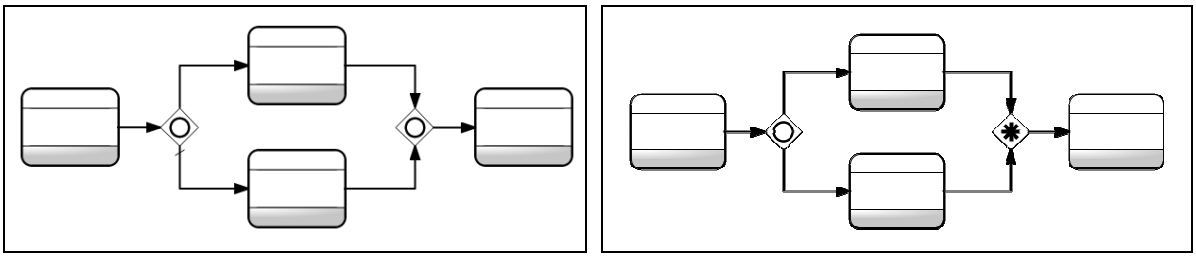
f) Parallel/Parallel

g) Parallel/Inclusive



h) Exclusive/Exclusive

i) Exclusive/Inclusive



j) Inclusive/Inclusive

k) Inclusive/Complex

Fig. 3.18. Constructores estructurados compartidos entre BPMN y UP-ColBPIP

Los constructores *Coreography Task* de BPMN (Figura **3.18** a)) y *Business Message* de UP-ColBPIP son utilizados para modelar interacciones entre organizaciones. Aunque representan abstracciones conceptuales diferentes para PNCs, ambos constructores tienen una semántica de comportamiento similar que considera que una interacción requiere el envío y recepción de mensajes (Figura **3.6**).

Mediante *Coreography Task* es también posible modelar un ciclo de interacciones utilizando el atributo *LoopType=Standard* (Figura **3.18** b)). El comportamiento de *Coreography Task* con este atributo es análogo al comportamiento del constructor *Loop* de UP-ColBPIP (Figura **3.12**). El atributo *LoopType* también puede ser configurado para tener el valor *MI Parallel* (Figura **3.18** c)). Con este valor *Coreography Task* permite tener múltiples instancias de interacciones. Este comportamiento es similar al del constructor *Multiple Instances* de UP-ColBPIP, con lo cual ambos constructores pueden ser formalizados con el mismo módulo GI abstracto (Figura **3.13**).

El constructor *Intermediate Event* de BPMN (Figura **3.18** d)) asociado a una actividad, sea ésta una *Coreography Task* o *subproceso*, indica la ocurrencia de un evento en el cual se interrumpe la actividad. El flujo de control continúa en un punto predeterminado e indica el conjunto de interacciones a realizar luego de ocurrido el evento. Este comportamiento es similar al comportamiento del constructor *Exception* de UP-ColBPIP (Figura **3.14**), en donde el comportamiento del evento asociado a una tarea puede ser formalizado como el disparador de la excepción, y el flujo de control que sigue al evento puede ser formalizado como un manejador de excepción.

En BPMN también se puede definir un evento de terminación explícita del PNC mediante el constructor *End Event* (Figura **3.18** e)). Este constructor tiene la misma semántica de comportamiento que el constructor *Termination* de UP-ColBPIP y su formalización se muestra en la Figura **3.16**.

Existen además otros constructores de BPMN que al ser combinados entre sí para formar constructores estructurados tienen una semántica de comportamiento similar a la de constructores de UP-ColBPIP. Este es el caso de los constructores estructurados de BPMN *Parallel/Parallel* (Figura **3.18** f)), y *Parallel/Inclusive* (Figura **3.18** g)), cuyas semánticas de comportamiento son análogas a la del constructor *And* de UP-ColBPIP (Figura **3.9**).

Los constructores estructurados de BPMN *Exclusive/Exclusive* (Figura **3.18** h)) y *Exclusive/Inclusive* (Figura **3.18** i)) pueden ser formalizados con el mismo módulo GI abstracto que el definido para el constructor *Xor* de UP-ColBPIP (Figura **3.8**). La semántica de comportamiento del constructor *Inclusive/Inclusive* de BPMN (Figura **3.18** j)) es análoga a la del *Or* con sincronización *Synchronizing Merge* de UP-ColBPIP (Figura **3.10**). En esta tesis el constructor *Complex gateway* es utilizado para sincronizar *n* caminos de un total de *m*. Esto implica que la semántica de comportamiento del constructor *Inclusive/Complex* de BPMN (Figura **3.18** k)) es análoga a la del *Or* con sincronización *N-out-of-M* de UP-ColBPIP (Figura **3.11**).

En las siguientes secciones se presenta la definición formal basada en GI-Nets de los constructores estructurados de BPMN cuya semánticas de comportamiento difieren de los constructores de UP-ColBPIP, los cuales son: *Parallel/Exclusive*, *Parallel/Complex*, *Exclusive/Parallel*, *Exclusive/Complex*, *Inclusive/Parallel*, e *Inclusive/Exclusive*.

3.4.2. Parallel/Exclusive

La Figura **3.19** muestra el módulo GI abstracto del constructor estructurado de BPMN que combina los gateways *Parallel* y *Exclusive*. Este módulo contiene dos posiciones *ip* y *op* que representan la entrada y salida del módulo respectivamente, y la transición *it* que representan la bifurcación del *Parallel*. La posición abstracta *ipp* junto con la transición abstracta *t* permiten representar a todos los posibles flujos de secuencia paralelos del *Parallel*. La unión del *Exclusive* gateway se representa por la transición abstracta *t*, la cual está conectada a la posición final *op*.

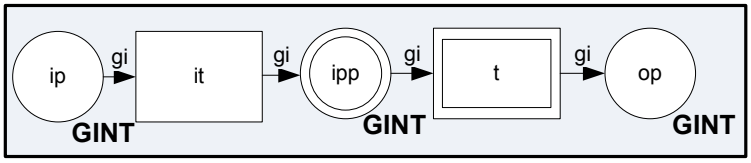


Fig. 3.19. Módulo GI abstracto de Parallel/Exclusive

Si hay una señal en la posición *ip*, se puede ejecutar la transición *it*. La ejecución de dicha transición coloca una señal en las posiciones representadas por *ipp*, permitiendo de

esta manera el paralelismo entre los flujos de secuencia definidos por las posiciones y transiciones representadas por *ipp* y *t*. Por cada transición representada por *t* que finaliza su ejecución se deposita una señal en la posición *op*, representando de esta manera al *Exclusive gateway*.

3.4.3. Parallel/Complex

La Figura **3.20** muestra el módulo GI abstracto del constructor estructurado de BPMN que combina un *Parallel* gateway y un *Complex* gateway. Este módulo contiene dos posiciones *ip* y *op* que representan la entrada y salida del módulo respectivamente, y la transición *it* que representa la bifurcación del *Parallel*. La posición abstracta *ipp* junto con la transición abstracta *t* permiten representar a todos los posibles flujos de secuencia paralelos del *Parallel*. La sincronización del *Complex* gateway se representa por el conjunto de posiciones y transiciones que siguen a continuación de la transición abstracta *t*. La descripción de esta parte del constructor es idéntica a la realizada para el constructor *Or/N-out-of-M* de UP-ColBPIP (Figura **3.11**).

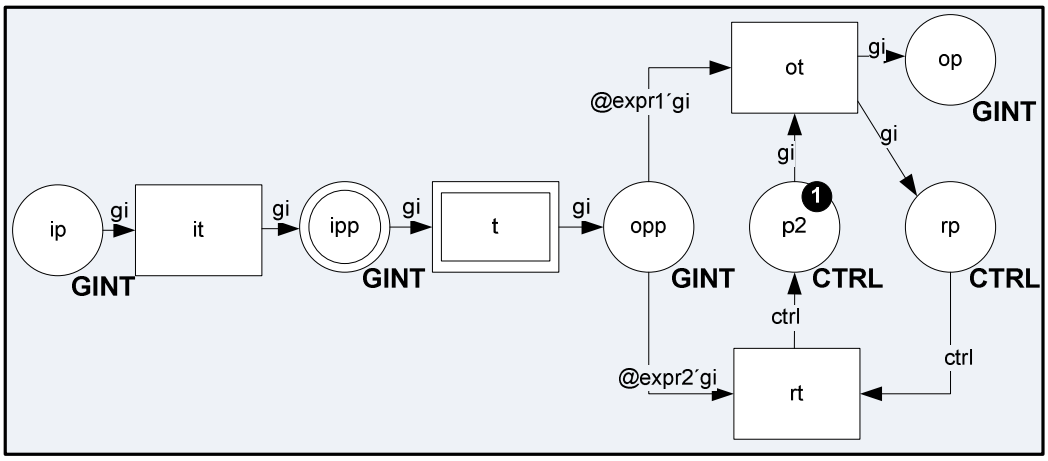


Fig. 3.20. Módulo GI abstracto del constructor Parallel/Complex

3.4.4. Exclusive/Parallel

La Figura **3.21** muestra el módulo GI abstracto de los constructores estructurados de BPMN que se forman a partir de la combinación de un *Exclusive* gateway y un *Parallel* gateway. Este módulo contiene dos posiciones *ip* y *op* que representan la entrada y salida del módulo respectivamente, y la transición *ot* que representa la sincronización *Parallel*. Tiene además una transición abstracta *t* que conecta a las posiciones mencionadas. La posición abstracta *opp* junto con la transición abstracta *t* permiten representar a todos los posibles flujos de secuencia alternativos del constructor.

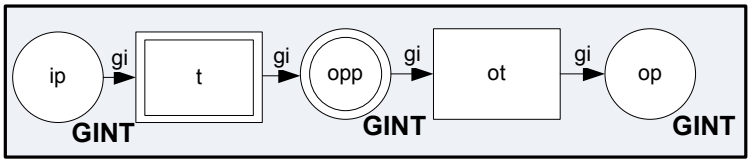


Fig. 3.21. Módulo GI abstracto del constructor Exclusive/Parallel

Si hay una señal en la posición *ip*, sólo una de las transiciones representadas por *t* puede ser ejecutada. Una vez ejecutada una transición, las demás quedan inhabilitadas, representando de esta manera la exclusión mutua entre los flujos de secuencia. Si hay una señal en alguna de las posiciones representadas por *opp* se habilita la transición *ot* que representa la sincronización *Parallel*, la cual al ejecutarse coloca una señal en la posición de salida *op*.

3.4.5. Exclusive/Complex

La Figura **3.22** muestra el módulo GI abstracto de los constructores estructurados de BPMN que combinan un *Exclusive* gateway y un *Complex* gateway. Este módulo contiene dos posiciones *ip* y *op* que representan las posiciones inicial y final del módulo respectivamente. La transición abstracta *t* permite representar a flujos de secuencia alternativos del *Exclusive* gateway. La sincronización del *Complex* gateway se representa por el conjunto de posiciones y transiciones que siguen a continuación de la transición abstracta *t*. La descripción de la semántica de comportamiento de estos elementos que

representan al *Complex* gateway es igual a la realizada para el constructor *Parallel/Complex* (Figura **3.20**).

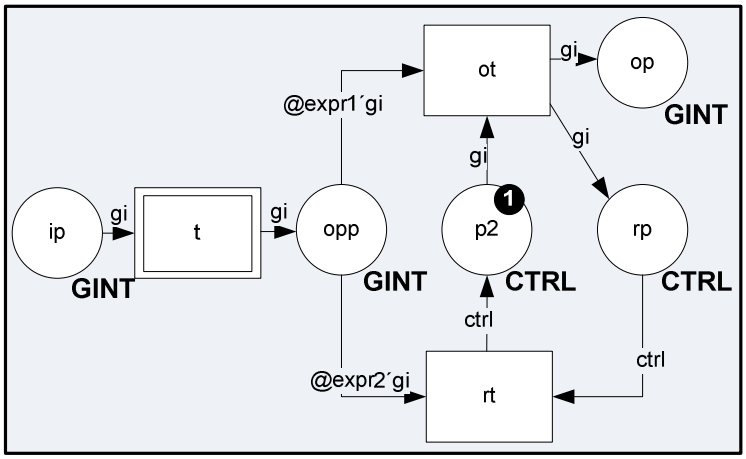


Fig. 3.22. Módulo GI abstracto del constructor Exclusive/Complex

3.4.6. Inclusive/Parallel

La Figura **3.23** muestra el módulo GI abstracto del constructor estructurado de BPMN que se forma a partir de la combinación de un *Inclusive* gateway y un *Parallel* gateway. Este módulo contiene dos posiciones *ip* y *op* que representan las posiciones inicial y final del módulo respectivamente, y dos transiciones *it* y *ot* que permiten realizar la bifurcación y sincronización del constructor respectivamente. Las posiciones abstractas *ipp* y *opp* junto con la transición abstracta *t* permiten representar a todos los posibles flujos de secuencia del *Inclusive* gateway.

La transición *it* deposita una señal en la posición *ipp* si la evaluación de la expresión *@cond* del arco que conecta a dichos elementos retorna falso. Caso contrario no se deposita señal alguna en la posición *ipp*. La expresión *@cond* en estos arcos indica a una variable booleana que debe ser evaluada para determinar si se deposita una señal en la posición *ipp*.

Cada módulo GI concreto que formaliza a un elemento que es instancia del constructor *Inlusive/Parallel* debe tener definida una variable booleana distinta por cada arco que conecte la transición *it* en base a la cantidad de flujos de secuencia que tiene dicho elemento.

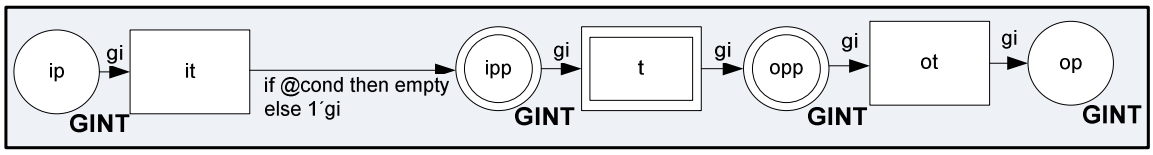


Fig. 3.23. Módulo GI abstracto de Inclusive/Parallel

3.4.7. Inclusive/Exclusive

La Figura **3.24** muestra el módulo GI abstracto del constructor estructurado de BPMN que combina un *Inclusive* gateway y un *Exclusive* gateway. Este módulo GI abstracto difiere del constructor *Inclusive/Parallel* en que la transición *t* se conecta directamente a la posición *op*. La descripción de los demás elementos es análoga a la de dicho constructor.

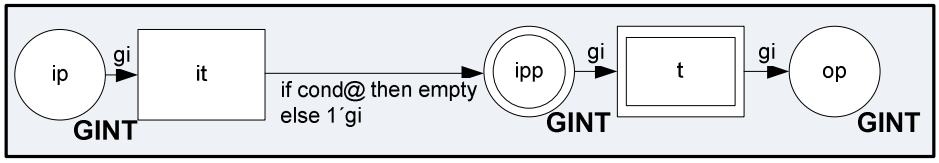


Fig. 3.24. Módulo GI abstracto de Inclusive/Exclusive

3.5. Caso de Estudio: Gestión de Pronóstico de Demanda

A continuación se aplica la formalización con GI-Nets a un caso de estudio de un PNC denominado *gestión de pronóstico de demanda*, con el propósito de definir un modelo formal del mismo. El PNC utilizado corresponde al desarrollo de una colaboración inter-organizacional del dominio de la Gestión Integrada de Cadenas de Suministro ("Supply Chain Integrated Management"), que tiene como principal objetivo la gestión colaborativa del aprovisionamiento de equipos informáticos desde una empresa proveedora ("TK Computers") a una empresa cliente ("Computer's Market").

El modelo conceptual PNC *gestión de pronóstico de demanda* definido con el lenguaje UP-ColBPIP se muestra en la Figura **3.25**. Dicho modelo expresa que el proceso comienza con el cliente quien solicita al proveedor generar un pronóstico de demanda de un producto (mensaje *request(ForecastRequest)*). En dicha solicitud se indican los datos a considerar en el pronóstico, como ser el producto objeto del pronóstico, el horizonte de tiempo, los períodos, etc. El proveedor procesa dicha solicitud y puede responder de dos maneras diferentes. En un caso, el proveedor acuerda la solicitud del cliente y se compromete a generar el pronóstico de demanda solicitado. En el otro caso, el proveedor rechaza la solicitud. Esto significa que el proceso finaliza con una falla. Esto se representa en el modelo UP-ColBPIP con un constructor *Xor*, donde uno de los caminos de interacción tiene el mensaje *refuse(ForecastRequestResponse)* y luego un *Termination* que indica que el PNC finaliza con una falla. El otro camino de interacción tiene el mensaje *agree(ForecastRequestResponse)* indicando la aceptación de la solicitud.

Si el proveedor acepta la solicitud, el cliente debe enviarle un pronóstico de ventas del producto para cada uno de sus cinco puntos de ventas (POS), y un plan de eventos programados (promociones y estrategias de ventas) para un horizonte de tres meses. Esto se representa en el modelo UP-ColBPIP con el constructor *And*, que tiene en uno de los caminos paralelos un constructor *Multiple Instances* donde el mensaje *inform(POSForecast)* se va a enviar en paralelo por cada uno de los puntos de venta. El otro camino del *And* envía el mensaje *inform(PlannedEvents)*. El cliente tiene un plazo de dos días para enviar el plan de eventos programados. Si ocurre que el cliente no responde en ese plazo, se debe manejar dicha excepción, donde el proveedor debe enviar una cancelación y el proceso finaliza (*cancel(DemandForecast)*).

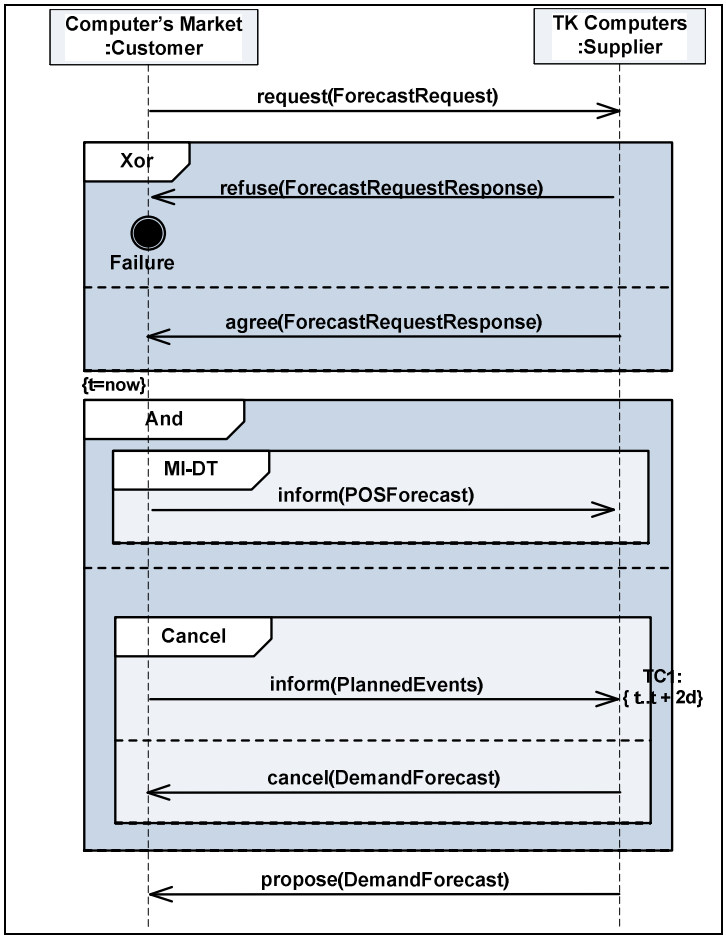


Fig. 3.25. PNC Gestión de Pronóstico de Demanda definido con UP-ColBPIP

3.5.1. Formalización con GI-Nets

Para poder generar modelos formales de GI-Nets a partir de modelos UP-ColBPIP se materializa el patrón de transformación definido en la Sección 3.2. El patrón de transformación materializado es $T = (TE, \Phi, \phi, GI_M^c, GI_N^*, ec, ce, GI_N, t)$, siendo $TE = (\lambda_s, \lambda_t, cc)$, donde:

- λ_s es el conjunto de constructores de UP-ColBPIP,
- λ_t son los módulos GI abstractos que formalizan los constructores de UP-ColBPIP presentados en la Sección 3.3,
- ϕ es el modelo UP-ColBPIP a formalizar mostrado en la Figura **3.25**,
- GI_N es la GI-Net que se genera y formaliza al modelo ϕ.

Se define a t como la máquina de transformación de un modelo UP-ColBPIP a un modelo GI-Net, la cual se muestra en el Algoritmo **3.1**. El algoritmo recibe como entrada un modelo o especificación de PNC. La variable *ginet* representa la GI-Net que formaliza al PNC de entrada. Para cada elemento del PNC se obtiene: su constructor (variable *construct*), el módulo GI abstracto de dicho constructor (variable *agim*), y se genera el módulo GI concreto del elemento (variable *cgim*). Luego se agrega el módulo GI concreto a la GI-Net con la función *addGIModule*, y se invoca a la función *t1*, que permite generar de manera recursiva los módulos concretos de cada elemento del PNC. Una vez obtenidos todos los módulos concretos la función *t* devuelve la GI-Net resultante (variable *ginet*).

La función *t1* recibe como entrada una GI-Net (variable *ginet*) y un elemento de modelo o especificación de PNC (variable *element*). Primero se obtienen los elementos hijos dentro de la estructura de árbol del PNC. Para cada uno de estos elementos se obtiene el módulo GI concreto y se lo agrega a la GI-Net resultante de manera similar a la función *t*. Una vez obtenidos todos los módulos concretos la función *t1* devuelve la GI-Net resultante (variable *ginet*) que va a ser utilizada por la función *t*.

Algoritmo 3.1. Pseudocódigo para la máquina de transformación *t*.

```
Function t (PNC)
ginet ⟵ initGINet()
for each element ∈ PNC
    construct = ec(element)
    agim = cc(construct)
    cgim = ce(agim,element)
    ginet ⟵ addGIModule(ginet,cgim)
    ginet ⟵ t1(ginet,element)
end
return ginet


Function t1 (ginet, element)
children ⟵ getChildren(element)
for each e ∈ children
    construct = ec(e)
    agim = cc(construct)
    cgim = ce(agim,e)
    ginet ⟵ addGIModule(ginet,cgim)
    ginet ⟵ t1(ginet,e)
end
return ginet
```

Para el PNC gestión de pronóstico de demanda (Figura **3.25**) se aplicó el patrón de transformación *T* explicado. La Figura **3.26** muestra la estructura de árbol de la GI-Net generada con el patrón que formaliza a dicho PNC. Cada nodo del árbol representa un elemento del PNC. El módulo raíz (PNC) es el protocolo de interacción. Los nodos IP1 e IP2 representan los caminos de interacción de los elementos como el *Cancel*, *Xor*, etc.

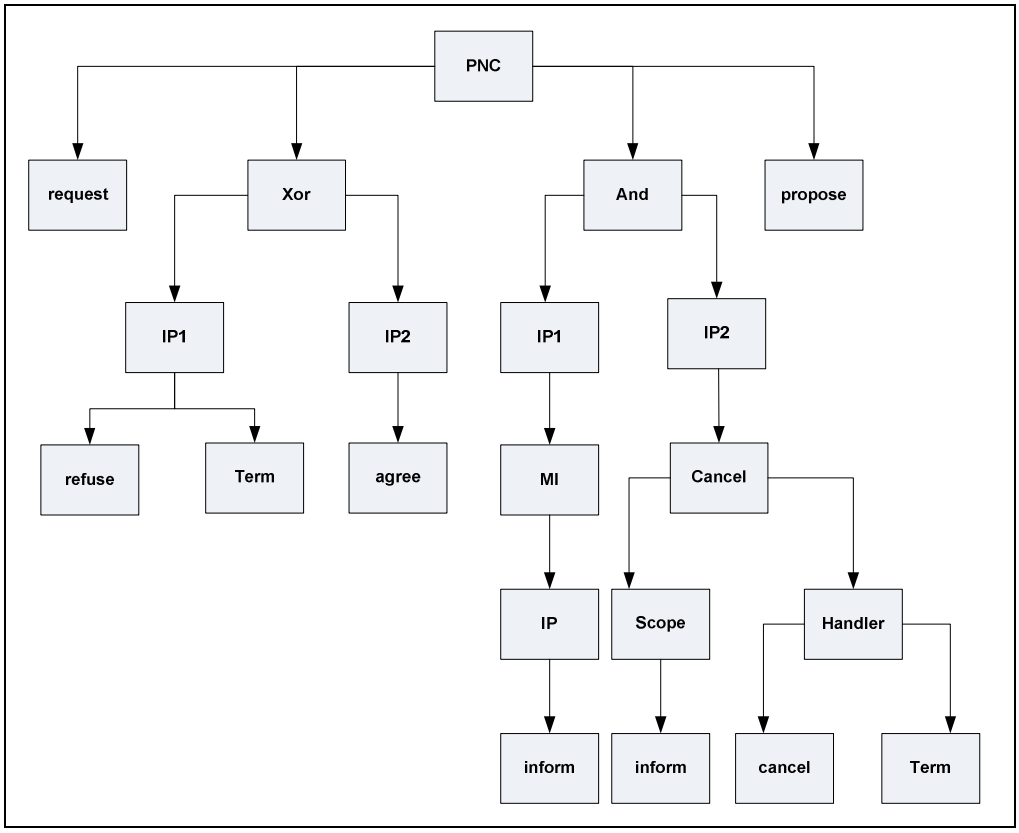


Fig. 3.26. Estructura de la GI-Net que formaliza el PNC Gestión de Pronóstico de Demanda

A continuación se presentan los módulos GI concretos más destacados de la GI-Net generada. El primer elemento que se formaliza es el protocolo de interacción, el cual se transforma en el módulo raíz de la GI-Net (Figura **3.27**). Este módulo está compuesto por una secuencia de elementos, donde cada transición referencia a un módulo GI concreto. Contiene la posición inicial de la GI-Net llamada *ip* y la posición final llamada *op*. La posición final *op* pertenece al conjunto de fusión *end*, indicado con una etiqueta *end* sobre dicha posición.

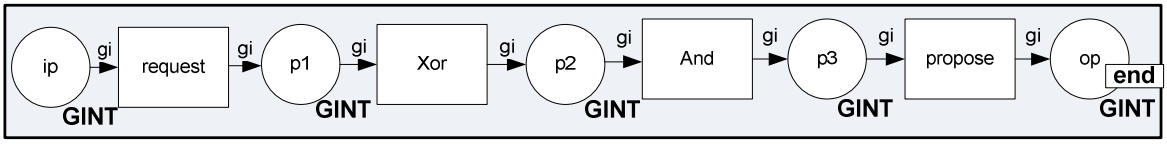


Fig. 3.27. Módulo GI concreto del protocolo de interacción

La transición *request* referencia al módulo *request* (Figura **3.28**) que representa el envío y recepción del mensaje de negocio *request(ForecastRequest)* del PNC. Para generar este módulo concreto se utiliza el módulo abstracto de la Figura **3.6** a). Este módulo concreto se define como hijo del nodo raíz en la estructura de árbol de la GI-Net (Figura **3.26**). Los módulos *refuse*, *agree*, *inform*, *cancel*, y *propose* tienen la misma estructura que el módulo *request*, ya que son todos mensajes de negocio.

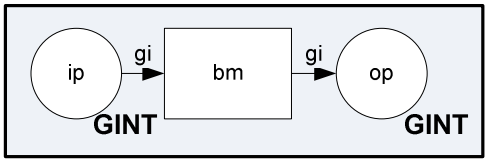


Fig. 3.28. Módulo GI concreto del mensaje de negocio request(ForecastRequest)

El siguiente módulo GI concreto que se genera es el del *Xor* que se muestra en la Figura **3.29**. Para generar dicho módulo se utiliza el módulo abstracto de la Figura **3.8**. El elemento *Xor* del modelo UP-ColBPIP está compuesto de dos caminos de interacción que son mutuamente excluyentes. De esta manera se utiliza el módulo abstracto para generar un módulo concreto que formaliza a un *Xor* con dos caminos de interacción por medio de las transiciones *IP1* y *IP2*. Estas transiciones son mutuamente excluyentes y cada una referencia a un módulo GI concreto que formaliza el camino de interacción correspondiente.

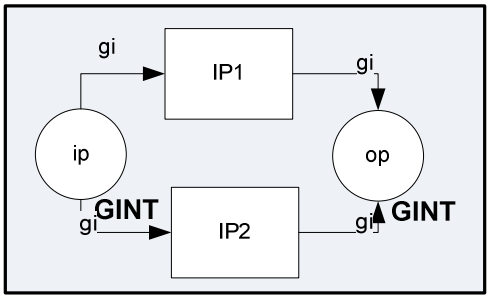
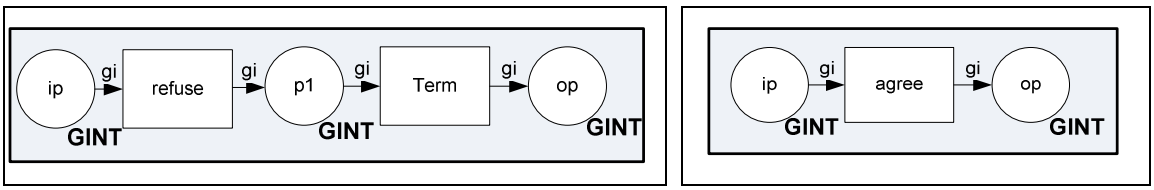


Fig. 3.29. Módulo GI concreto Xor



a) IP1 b) IP2

Fig. 3.30. Módulos GI concretos referenciados por el Xor

El camino de interacción IP1, que contiene al mensaje *refuse* y al elemento *Termination*, es formalizado por el módulo GI concreto que se muestra en la Figura **3.30** a), el cual está compuesto de las transiciones *refuse* y *Term*. La transición *refuse* referencia al módulo GI concreto que formaliza al mensaje *refuse(ForecastRequestResponse)*. Este módulo es análogo al presentado en la Figura **3.28** para el mensaje *request(ForecastRequest)*. El módulo GI concreto del elemento *Termination* se muestra en la Figura **3.31**. Este módulo contiene una posición que pertenece al conjunto de fusión *end*, indicado con una etiqueta *end* sobre dicha posición. De esta manera, cuando se coloca una señal en la posición *endp*, se está colocando también una señal en la posición final de la GI-Net (Figura **3.27**), es decir, la ejecución de cualquiera de los módulos *Termination* de la GI-Net deposita una señal en la posición final de la GI-Net, indicando la finalización del PNC. Todos los elementos *Termimation* del PNC comparten la misma definición formal.

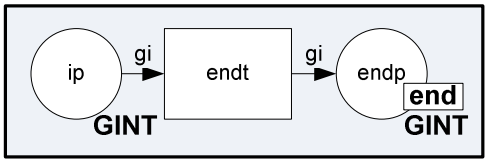


Fig. 3.31. Módulo GI concreto del elemento Termination

El otro camino de interacción del *Xor* es formalizado por el módulo GI concreto IP2 mostrado en la Figura **3.30** b), el cual contiene la transición *agree* que representa al módulo que formaliza el mensaje *agree(ForecastRequestResponse).*

El siguiente módulo que se genera es el correspondiente al elemento *And*, el cual tiene dos caminos de interacción que se ejecutan en paralelo. Para ello se utiliza el módulo abstracto de la Figura **3.9** que permite generar el módulo concreto que formaliza a un *And*. El módulo que se genera tiene dos caminos de interacción paralelos representados por las transiciones *IP1* y *IP2* (Figura **3.32**). Cada transición referencia a un módulo GI concreto que representa a un camino de interacción.

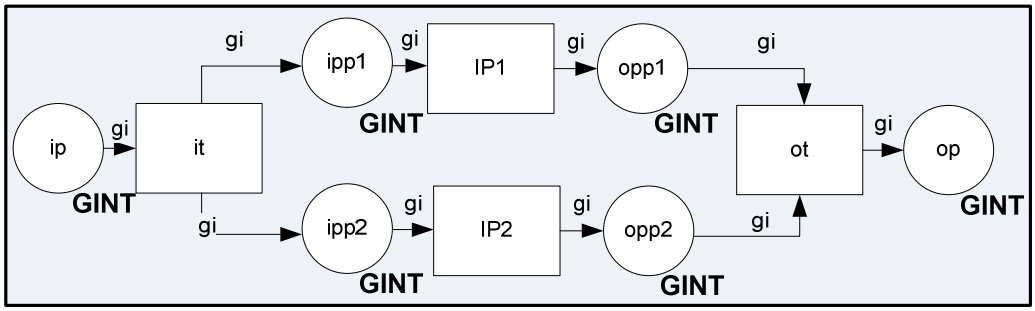
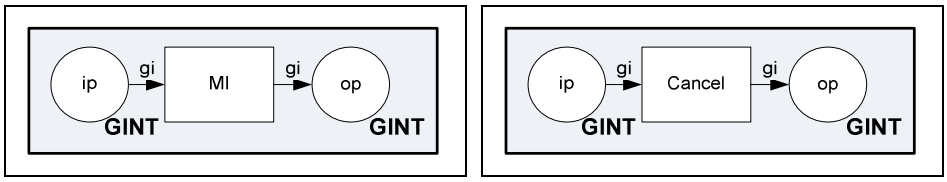


Fig. 3.32. Módulo GI concreto And

El camino de interacción IP1 del elemento *And* contiene al elemento *Multiple Instances*, mientras que IP2 contiene al elemento *Cancel*. Por lo tanto, el módulo GI concreto IP1 (Figura **3.33** a)) está compuesto de la transición *MI* que referencia al módulo MI. El otro camino de interacción se muestra en la Figura **3.33** b), el cual está compuesto de la transición *Cancel* que referencia al módulo Cancel.



a) IP1 b) IP2

Fig. 3.33. Módulos GI concretos referenciados por el And

El módulo GI concreto MI que formaliza al elemento *Multiple Instances* (Figura **3.34**) se obtiene a partir del módulo abstracto mostrado en la Figura **3.13**. El arco que va de la transición *it* a la posición *ipp* representa la generación de cinco instancias. El arco que va de la posición *opp* a la transición *ot* representa que se deben sincronizar las cinco instancias. La transición inform referencia al módulo inform que formaliza al mensaje *inform(POSForecast)*. Dicho módulo se formaliza de la misma manera que el del mensaje *request(ForecastRequest)* mostrado en la Figura **3.28**.

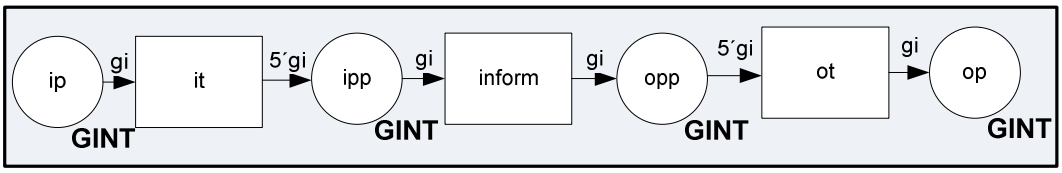
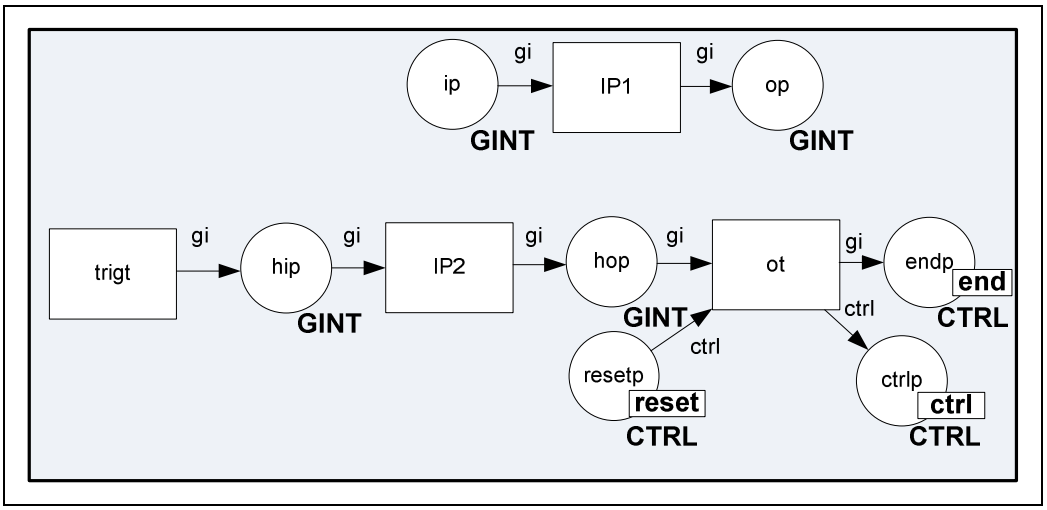
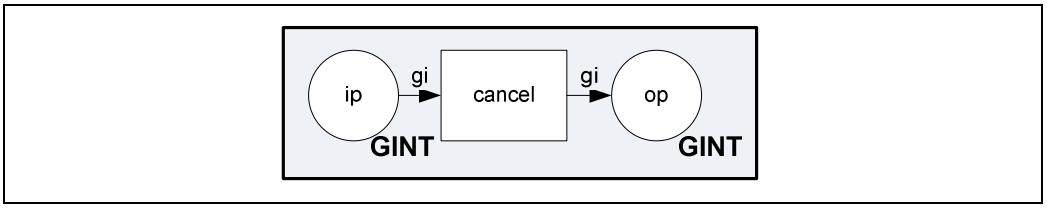


Fig. 3.34. Módulo GI concreto MI

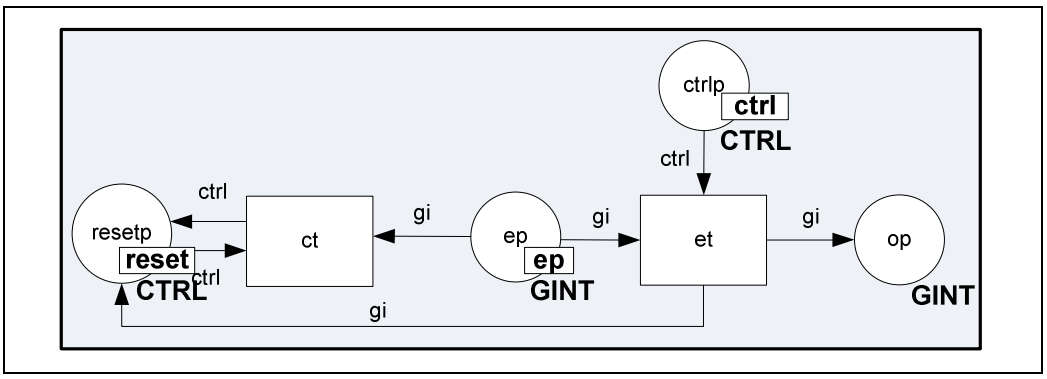
El siguiente elemento que se formaliza es el *Cancel*, el cual se define a partir del módulo abstracto definido en la Figura **3.15**. El módulo concreto que se obtiene se define como hijo del nodo IP2 del elemento *And* en la estructura de árbol de la GI-Net (Figura **3.26**). El *Cancel* está formado por dos caminos de interacción, donde el primer camino representa el alcance y el segundo camino representa al manejador de excepciones. El módulo GI concreto que formaliza a este elemento se muestra en la Figura **3.35**. Este módulo se divide en tres partes: el módulo principal del *Cancel* (Figura **3.35** a)), el manejador de excepciones (Figura **3.35** b)), y el disparador de excepciones (Figura **3.35** c)).



a) Cancel



b) Módulo GI concreto IP2 (Manejador de excepciones del Cancel)



c) Disparador de excepciones del Cancel

Fig. 3.35. Módulo GI concreto del Cancel

El módulo principal está compuesto de tres transiciones principales: *IP1, IP2, y trigt.* La transición *IP1* referencia al módulo GI concreto *Scope* del camino de interacción que contiene los elementos que están dentro del alcance del *Cancel*. Esto se ve reflejado en la estructura de árbol mediante el módulo *Scope* que es hijo del nodo *Cancel* (Figura **3.26**). La transición *IP2* referencia al módulo GI concreto *Handler* que representa al camino de interacción que contiene los elementos que manejan la excepción. En este caso contiene la invocación al mensaje de negocio *cancel(DemandForecast)* (Figura **3.35** b)). Finalmente, la

transición *trigt* referencia a la parte del módulo GI abstracto del *Cancel* que contiene al disparador de excepciones (Figura **3.35** c)).

El módulo *Scope* que representa el alcance del *Cancel* formaliza al camino de interacción que contiene al mensaje *inform(PlannedEvents)*. El módulo GI concreto de este camino de interacción se muestra en la Figura **3.36**.

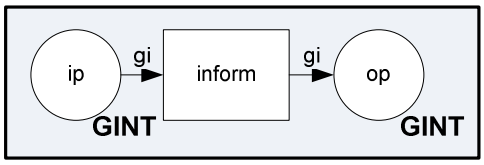


Fig. 3.36. Módulo GI concreto Scope

El elemento que se encuentra dentro del alcance del *Cancel* es el mensaje de negocio *inform(PlannedEvents)*. A diferencia del módulo GI concreto de los demás mensajes del PNC, como el mostrado en la Figura **3.28**, el módulo que formaliza al mensaje *inform(PlannedEvents)* (Figura **3.37**) agrega la posición de control *ctrlp* que pertenece al conjunto de fusión *ctrl*. Este conjunto de fusión contiene a las posiciones de control que son utilizadas para bloquear la ejecución de los caminos de interacción cuando ocurre una determinada excepción. En este caso, el mensaje *inform(PlannedEvents)* se encuentra dentro del alcance del elemento *Cancel*, con lo cual es necesario agregar esta posición de control. Eso no ocurre con los demás mensajes del PNC, ya que no son parte del alcance del *Cancel*, y su ejecución no puede ser interrumpida debido a la ocurrencia de una excepción.

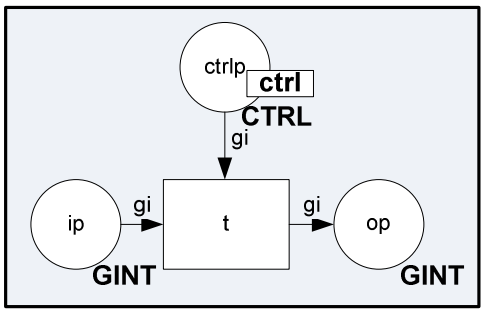


Fig. 3.37. Módulo GI concreto del mensaje de negocio inform(PlannedEvents)

Propose es el último módulo que se genera y representa al mensaje *propose(DemandForecast)*. El módulo concreto de este elemento es análogo al presentado en la Figura **3.28**.

3.6. Conclusiones

El lenguaje formal redes de Interacción Global (GI-Nets) definido en base al formalismo de redes de Petri Jerárquicas y Coloreadas y el patrón de transformación para GI-Nets, permiten generar un modelo formal GI-Net que formaliza un modelo o especificación estructurado de PNC.

El lenguaje GI-Nets propuesto es independiente de la semántica de los lenguajes de modelado/especificación de PNCs. Esto lo hace flexible y adaptable, ya que puede ser utilizado para formalizar PNCs definidos con distintos lenguajes y, por lo tanto, ser utilizado en la formalización de modelos o especificaciones de PNCs que son requeridas en diferentes etapas del desarrollo dirigido por modelos de sistemas de información inter-organizacionales. El uso de módulos para estructurar una GI-Net posibilita esta independencia.

El lenguaje GI-Nets permite formalizar los constructores de un lenguaje de modelado/especificación de PNCs de manera separada de los elementos que conforman un modelo/especificación de PNC con dicho lenguaje. Los módulos GI abstractos permiten formalizar los constructores de un lenguaje, es decir, definen la estructura que deben tener los modelos formales, sin hacer referencia a ninguna instancia de constructor en particular. Los módulos GI abstractos pueden ser reutilizados o adaptados para formalizar constructores de diferentes lenguajes de PNCs, cuyas semánticas son iguales o similares.

Un modelo o especificación de PNC se formaliza a través de una GI-Net compuesta de módulos GI concretos. Un módulo GI concreto formaliza un elemento de un modelo de PNC, sea éste un elemento de flujo de control o de interacción, y se deriva y define de acuerdo al módulo GI abstracto que formaliza el constructor al que refiere el elemento formalizado.

El lenguaje GI-Nets permite además la formalización de constructores complejos para sincronización avanzada, manejo de excepciones y cancelaciones, e instancias

múltiples. La formalización de estos constructores complejos obliga a la definición de estados intermedios que no representan interacciones y que son utilizados para controlar el comportamiento de un PNC. Las GI-Nets permiten diferenciar entre estados de interacción (GINT) y estados de control (CTRL). Esto es posible debido a que las GI-Nets están basadas en las HCP-Nets, que a diferencia de las redes de Petri clásicas, permiten la definición de distintos tipos de estados.

El patrón formal de transformación de GI-Nets define de manera general los requerimientos de transformación para generar modelos formales GI-Net de PNCs. El mismo sigue los principios del desarrollo dirigido por modelos, y puede ser materializado y aplicado para generar GI-Nets a partir de modelos de PNCs definidos con un lenguaje no formal. Uno de los principales beneficios de este patrón de transformación es que los módulos GI abstractos funcionan como componentes generativos para obtener los módulos GI concretos, ya que a partir de los mismos es posible generar todos los posibles módulos GI concretos que formalizan los elementos que pueden ser parte de un modelo de PNC.

Debido a que una GI-Net está estructurada en formal de árbol, el modelo de PNC a partir del cual es generada debe ser también estructurado. En lenguajes estructurados (tal como UP-ColBPIP) que sólo generan modelos estructurados de PNC esto es directo. Sin embargo, para formalizar modelos no estructurados de PNCs, (tales como aquellos definidos con BPMN), los mismos tienen que ser primeramente transformados a modelos estructurados antes de generar sus correspondientes GI-Nets. Esto se puede lograr mediante la aplicación de métodos de estructuración de procesos propuestos en otros trabajos (Vanhatalo, Völzer & Koehler 2009).

El lenguaje GI-Nets permitió formalizar los constructores de los lenguajes UP-ColBPIP y BPMN, definiendo para cada uno un módulo GI abstracto. Los constructores formalizados incluyen constructores complejos con sincronización avanzada, manejo de excepciones y cancelaciones, e instancias múltiples. Algunos de estos módulos GI abstractos (Xor, And, Or, Multiple Instance, Loop) han sido definidos de acuerdo a modelos de redes de Petri coloreadas utilizados para expresar patrones de workflow (Russell et al. 2006), mientras que otros módulos son nuevas propuestas de formalización, tales como los constructores de interacción, terminación, excepción y cancelación.

Finalmente, un caso de estudio mostró la aplicabilidad del enfoque de formalización. Se materializó el patrón de transformación de GI-Nets y se utilizaron los módulos GI abstractos definidos para el lenguaje UP-ColBPIP, formalizando con los mismos un modelo conceptual de PNC basado en UP-ColBPIP.

4 Verificación Formal del Comportamiento de Procesos de Negocio Colaborativos

En este capítulo se presenta un método formal de verificación de PNCs que utiliza el enfoque de formalización en base a GI-Nets propuesto en el Capítulo 3. Primero, se describen las etapas del método (Sección 4.1) y se introduce la propiedad Solidez de Interacción Global que permite analizar el comportamiento de un PNC y determinar la presencia de bloqueos (Sección 4.2). El método se aplica a dos casos de estudio en los cuales se verifica el comportamiento de PNCs definidos con UP-ColBPIP y BPMN (Sección 4.3). Finalmente, se presentan las conclusiones (Sección 4.4).

4.1. Método de Verificación de PNCs basado en GI-Nets

Para dar soporte a la verificación de modelos o especificaciones de PNCs, se propone un método formal de verificación, que se basa en transformaciones de modelos (Figura **4.1**). El mismo consta de dos etapas: formalización y verificación.

Etapa de formalización: en esta etapa se utiliza el lenguaje GI-Nets para formalizar los constructores del lenguaje usado para definir los modelos o especificaciones a verificar, y el patrón de transformación para GI-Nets (Capítulo 3). El método recibe como entrada un modelo o especificación de PNC a verificar, definido en un dado lenguaje, junto con los módulos GI abstractos que formalizan los constructores de dicho lenguaje.

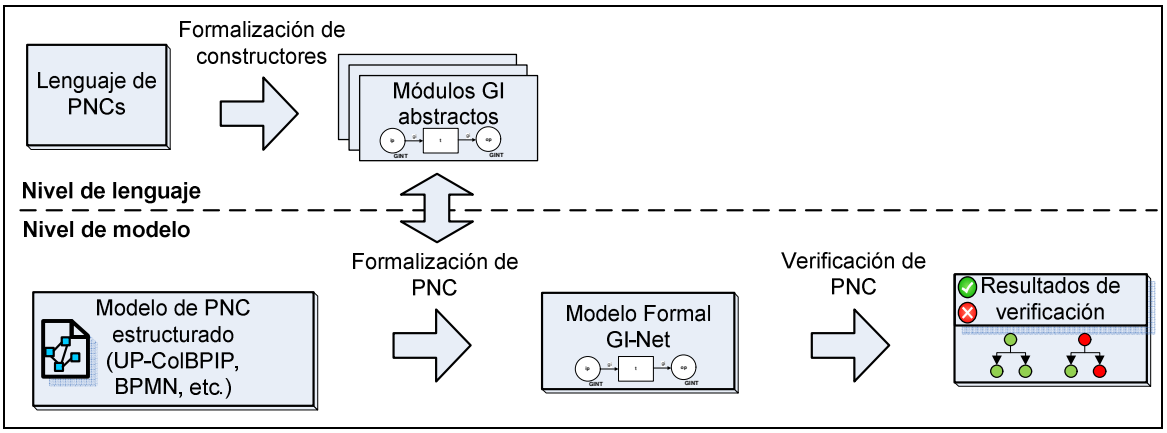


Fig. 4.1. Método de verificación de un PNC

La salida de la etapa de formalización es un modelo formal, definido por una GI-Net, correspondiente al modelo o especificación del PNC de entrada. Para poder verificar una herramienta de redes de Petri, esta debe ser generada en el formato o representación requerido por dicha herramienta. Para ello se aplican dos patrones de transformación. Primero se aplica el patrón de transformación que se muestra en la Figura **4.2**, el cual está basado en el patrón de transformación para GI-Nets (Definición **3.7**). Este patrón toma como entrada el modelo estructurado del PNC, el cual es una instancia del metamodelo del lenguaje de PNC usado, y genera la GI-Net en formato PNML (Petri Net Markup Language) (Billington et al. 2003). PNML es un lenguaje estándar para definir modelos de redes de Petri, el cual es interpretado por varias herramientas disponibles basadas en redes de Petri.

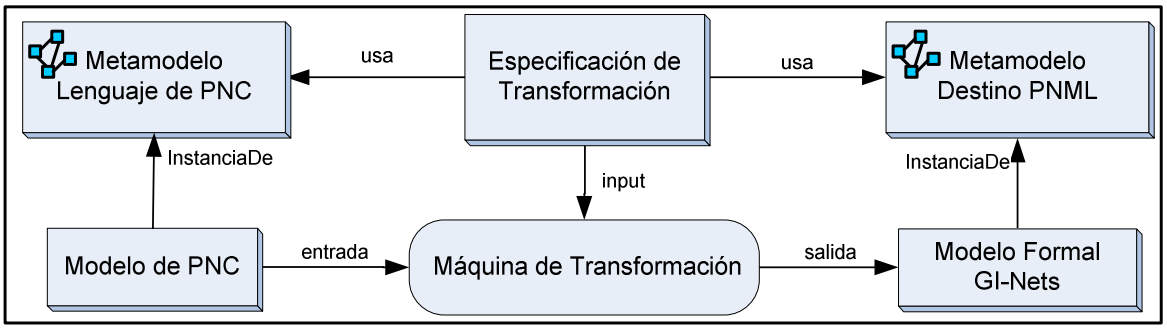


Fig. 4.2. Patrón de transformación para GI-Nets en formato PNML

Debido a que en esta tesis se decidió utilizar la herramienta CPN Tools, cuyo formato es particular, se aplica el patrón de transformación que se muestra en la Figura **4.3**. Este patrón recibe como entrada una GI-Net en formato PNML, y genera documentos XML que representan la GI-Net en formato CPN Tools (CPN Tools 2013). De esta manera, se utilizan GI-Nets basadas en el estándar PNML como modelos intermedios a partir de los cuales es posible generar GI-Nets basadas en CPN Tools o en cualquier otra herramienta específica de redes de Petri.

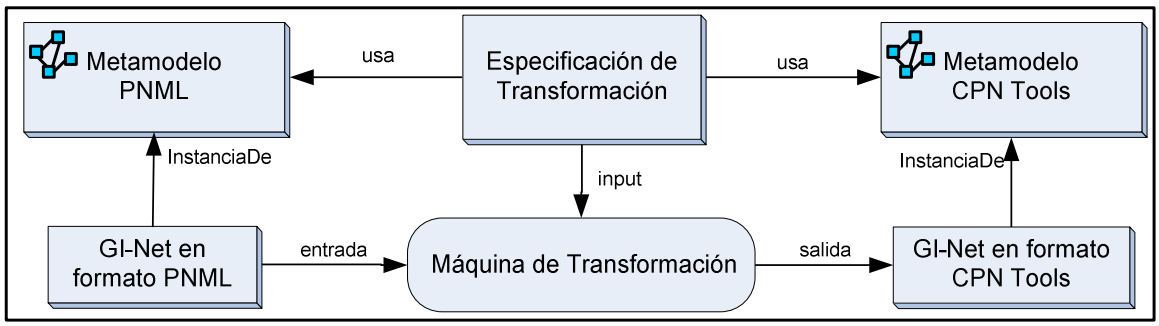


Fig. 4.3. Patrón de transformación para GI-Nets en formato CPN Tools

Etapa de verificación: en esta etapa se analiza la GI-Net a efectos de determinar si satisface las propiedades de comportamiento deseadas. Para ello se define la propiedad *Solidez de Interacción Global* (descripta en la Sección 4.2), la cual permite determinar la ausencia o no de bloqueos en la GI-Net, y por ende, en el modelo del PNC que formaliza. El análisis del comportamiento de una GI-Net con CPN Tools consiste en generar el espacio de estados de la red, a partir del cual se genera información respecto a los estados de la red, transiciones muertas, y estados finales. Esta información permite determinar si la GI-Net satisface la propiedad Solidez de Interacción Global, lo que implica que la GI-Net es sólida. Si la GI-Net es sólida, se concluye que el modelo o especificación de PNC verificado también es sólido. Esto significa que es correcto desde el punto de vista de su comportamiento. De lo contrario, si no se satisface la propiedad Solidez de Interacción Global, se concluye que la GI-Net no es sólida, lo que implica que el comportamiento del modelo o especificación de PNC verificado no es correcto.

Además de permitir concluir acerca del comportamiento de un modelo de PNC, en caso de errores, permite determinar el lugar exacto donde ocurren los errores, ya sea

bloqueos o estados no alcanzables. Para esto, en la salida del método de verificación se genera la interpretación de los resultados devueltos por la herramienta de verificación utilizada.

4.2. Propiedad Solidez de Interacción Global

La propiedad de solidez "clásica" fue definida para verificar modelos de procesos o workflows (van der Aalst 1998). La misma se basa en el uso de redes de Petri clásicas para analizar el comportamiento de un workflow, y establece que: en el estado final sólo debe haber señales en la única posición final de la red. No obstante, la formalización de constructores de flujo de control complejos requiere el uso de redes de Petri coloreadas (CP-Nets) que contengan un conjunto de posiciones, distintas de la posición final, que pueden tener señales en el estado final de una red.

Por ejemplo, la Figura **4.4** muestra una red de Petri clásica que se encuentra en su estado final, la cual formaliza un constructor complejo. De acuerdo a la propiedad clásica de solidez, esta red no es sólida, debido a que en el estado final existe una posición (*p1*), distinta de la posición final (*op*), que tiene una señal. Esto implica que mediante las redes de Petri clásicas no es posible formalizar constructores de flujo de control complejos. Esto también implica que, si se desea utilizar CP-Nets para formalizar constructores de flujo de control complejo, es necesario adaptar la definición clásica de solidez para que acepte en el estado final de una red señales en diferentes posiciones (distintas de la posición final). Es decir, que se permitan estados finales como el que se muestra en la Figura **4.4**.

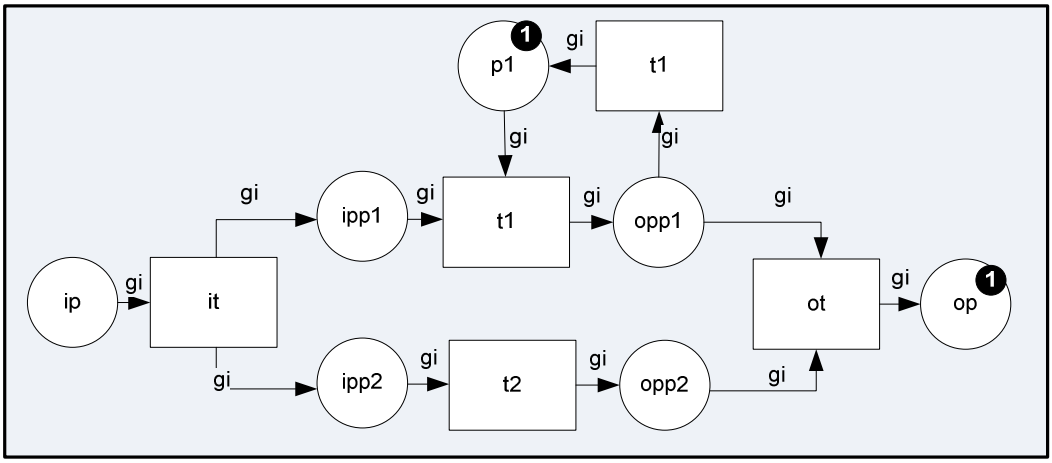


Fig. 4.4. Red de Petri en su estado final

Existen variantes de la propiedad de solidez clásica que relajan la restricción mencionada (van der Aalst et al. 2010), las cuales permiten en el estado final de una red señales en posiciones distintas a la posición final. Sin embargo, estas variantes sólo consideran durante el análisis que la posición final se encuentre en un estado apropiado, pero no tienen en cuenta el estado en el que se encuentran las demás posiciones de la red. Por ejemplo, en la Figura **4.4** sólo se consideraría el estado de la posición *op*, pero se dejaría de lado el estado de la posición *p1*, con lo cual *p1* podría tener o no señales. Esto implica que aunque se pueda afirmar que la posición final *op* se encuentra en un estado apropiado, no se pueden emitir conclusiones respecto de las demás posiciones de la red.

Con el objetivo de permitir la verificación de flujos de control complejos para PNCs, se propone la propiedad Solidez de Interacción Global, la cual adapta la propiedad de solidez clásica de manera tal de permitir que haya señales en posiciones distintas a la posición final de una GI-Net. Para esto se considera que en la evaluación del estado final de una GI-Net es necesario diferenciar entre *posiciones de control* (cuyo color es de tipo $CTRL$) y *posiciones de interacción* (cuyo color es de tipo $GINT$). De esta manera, en el estado final de una GI-Net sólo la posición de interacción final puede tener una señal, y las posiciones de control pueden tener o no señales, de acuerdo al estado en el que se encontraban al inicio. Esto se define de la siguiente manera.

Definición 4.1. ***(Solidez de Interacción Global)*** Sea $GI_N = (S, SM, PS, FS)$ una GI-Net. Sean P_C las posiciones de control y P_I las posiciones de interacción de GI_N, donde $ip, op \in P_I$ son las posiciones de interacción de entrada y salida de GI_N respectivamente. Sean además M_E, M_0 los estados vacío e inicial respectivamente de GI_N, tal que en el estado vacío M_E sólo pueden tener señales las posiciones de control, es decir, $\forall_{p \in P_C} |M_E(p)| \geq 0 \wedge \forall_{p \in P_I} |M_E(p)| = 0$, y el estado inicial M_0 está formado por el estado vacío M_E y el estado en el cual la posición ip tiene una señal ($M(ip)$), es decir, $M_0 = M_E + M(ip)$. Además, sea M_F el estado final de GI_N, tal que el estado final M_F está formado por el estado vacío M_E y el estado en el cual la posición op tiene una señal ($M(op)$), es decir, $M_F = M_E + M(op)$. Entonces, GI_N cumple la propiedad *solidez de interacción global* sii se cumplen los siguientes requerimientos:

1. existe opción para finalizar: $\forall_M \left(M_0 \overset{*}{\rightarrow} M\right) \Rightarrow (M \rightarrow M_F)$;
2. finalización adecuada: $\forall_M \left(M_0 \overset{*}{\rightarrow} M \wedge M \geq M_F\right) \Rightarrow (M = M_F)$;
3. no existen transiciones muertas: $\forall_{t \in T} \exists_{M,M'} M_0 \overset{*}{\rightarrow} M \overset{t}{\rightarrow} M'$.

El primer requerimiento establece que si se comienza en el estado inicial M_0 siempre es posible llegar al estado final M_F, es decir, para todo estado M que es alcanzable desde el estado inicial M_0, se cumple que el estado final M_F es alcanzable desde M. El segundo requerimiento establece que en el momento en que se coloca una señal en la posición op, todas las demás posiciones de interacción deberían estar vacías y las posiciones de control deberían estar en sus respectivos estados iniciales. El tercer requerimiento establece que no debe haber transiciones muertas en el estado inicial M_0. Estos requerimientos permiten determinar la ausencia o no de bloqueos y elementos no alcanzables.

La principal diferencia entre la propiedad de solidez clásica (Van der Aalst 1998) y la propiedad Solidez de Interacción Global propuesta, es que la primera está definida en términos de redes de Petri clásicas, mientras que la segunda está definida en términos de redes de Petri Jerárquica y Coloreadas. Esta diferencia se observa en el estado final M_F de la propiedad de Solidez de Interacción Global, que implica que cuando se coloca una señal en la posición op, todas las demás posiciones de tipo $GINT$ deberían estar vacías, y las posiciones de tipo $CTRL$ deberían estar en sus respectivos estados iniciales, los cuales no

necesariamente deben ser estados vacíos. Esto quiere decir que si una dada posición de tipo $CTRL$ tiene una señal en el estado inicial, para que se cumpla la propiedad Solidez de Interacción Global, esta posición también debería tener una señal en el estado final. Si la misma posición estuviera vacía en el estado inicial, en el estado final también debería estar vacía. De esta manera es posible determinar si en el estado final tanto la posición final de la GI-Net como las demás posiciones finalizan en un estado adecuado.

A partir de aquí y para los restantes capítulos de esta tesis se utiliza el término *solidez* para referirse a la propiedad *Solidez de Interacción Global* de una GI-Net. Se considera que si una GI-Net es *sólida*, entonces dicha GI-Net satisface la propiedad solidez de interacción global. Se considera además que un modelo o especificación de PNC es sólido, si su correspondiente GI-Net es sólida.

4.2.1. Condición Necesaria y Suficiente para Solidez

Para determinar si una dada GI-Net $GI_N = (P, T, F)$ es sólida, se define la GI-Net extendida $\overline{GI}_N = (\overline{P}, \overline{T}, \overline{F})$ agregando la transición t^* que conecta op e ip a GI_N (Figura **4.5**). La GI-Net extendida $\overline{GI}_N = (\overline{P}, \overline{T}, \overline{F})$ se define de la siguiente manera:

- $\overline{P} = P$,
- $\overline{T} = T \cup \{t^*\}$, y
- $\overline{F} = F \cup \{\langle op, t^* \rangle, \langle t^*, ip \rangle\}$.

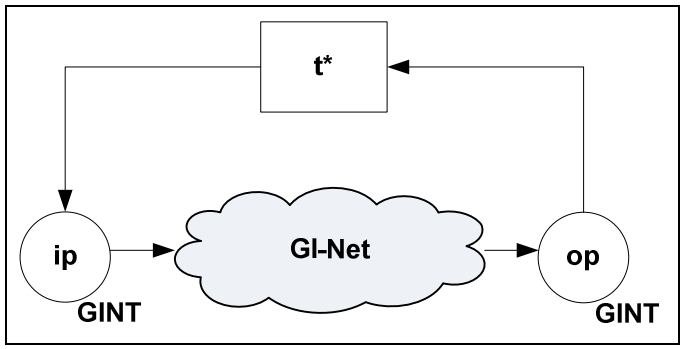


Fig. 4.5. GI-Net extendida $\overline{GI}_N$

Para una dada GI-Net GI_N y su correspondiente red extendida $\overline{GI}_N$ se va a probar que GI_N es sólida sii $(\overline{GI}_N, M_0)$ está *viva* y es *acotada*, las cuales son dos propiedades de las redes de Petri clásicas (según Definición **0**.2) que pueden ser aplicadas a las GI-Nets.

Lema 4.1. Si $(\overline{GI}_N, M_0)$ está *viva* y es acotada, entonces GI_N es sólida.

Demostración.

Asúmase que $(\overline{GI}_N, M_0)$ está viva, es decir, para cada estado alcanzable M existe una secuencia de disparos de transiciones que llevan al estado en el cual la transición t^* esté habilitada. Debido a que op es la posición de entrada de t^*, para cada estado M alcanzable desde el estado M_0, es posible alcanzar un estado con al menos una señal en la posición op. Considere ahora un estado alcanzable arbitrario $M' + M(op)$, es decir, un estado con una señal en la posición op. En este estado t^* está habilitada. Si t^* se dispara, se alcanza el estado $M' + M(ip)$. Debido a que $(\overline{GI}_N, M_0)$ es también acotada, M' debería ser igual al estado vacío M_E. Por lo tanto, los requerimientos 1 y 2 (establecidos en definición **4.1**) se cumplen y se garantiza una terminación apropiada. El requerimiento 3 también se cumple ya que $(\overline{GI}_N, M_0)$ está viva. Por lo tanto, GI_N cumple la propiedad de solidez. ■

Lema 4.2. Si GI_N es sólida, entonces $(\overline{GI}_N, M_0)$ es acotada.

Demostración.

Asuma que GI_N es sólida y (GI_N, M_0) no es acotada. Debido a que GI_N no es acotada existen dos estados M_i y M_j tal que $M_0 \overset{*}{\rightarrow} M_i$, $M_i \overset{*}{\rightarrow} M_j$, y $M_j > M_i$, es decir, existe un estado que es alcanzable desde otro estado, tal que el estado alcanzable tiene más señales que el otro estado (ver por ejemplo la prueba de que el árbol de cobertura (o coverability tree) es finito en (Peterson 1981) (Teorema 4.1)). Sin embargo, debido a que GI_N es sólida, se sabe que existe una secuencia de disparos σ tal que $M_i \overset{\sigma}{\rightarrow} M_F$. Por lo tanto, existe un estado M tal que $M_j \overset{\sigma}{\rightarrow} M$ y $M > M_F$. Esto significa que no es posible que GI_N sea sólida y no sea acotada. Entonces, si GI_N es sólida, (GI_N, M_0) es acotada. A partir del hecho de que GI_N es sólida y que (GI_N, M_0) es acotada, se puede deducir que $(\overline{GI}_N, M_0)$ es acotada. Si la transición t^* en $\overline{GI}_N$ se dispara, la red retorna al estado inicial M_0. ■

Lema 4.3. Si GI_N es sólida, entonces $(\overline{GI}_N, M_0)$ está viva.

Demostración.

Asúmase que GI_N es sólida. Por el Lema **4.2** se sabe que $(\overline{GI}_N, M_0)$ es acotada. Debido a que GI_N es sólida se sabe que M_0 es un estado origen de $\overline{GI}_N$, es decir, para cada estado M' alcanzable desde $(\overline{GI}_N, M_0)$ es posible retornar al estado M_0. En la red original (GI_N, M_0), es posible disparar una transición arbitraria t (requerimiento 3). Esto también se cumple para $(\overline{GI}_N, M_0)$. Por lo tanto, $(\overline{GI}_N, M_0)$ está viva, ya que para cada estado M' alcanzable desde $(\overline{GI}_N, M_0)$ es posible alcanzar un estado que habilita a una transición arbitraria t. ■

Teorema 4.2. Una GI-Net GI_N es sólida sii $(\overline{GI}_N, M_0)$ está viva y es acotada.

Demostración.

Se demuestra directamente a partir de los Lemas **4.1**, **4.2**, **4.3**. ■

4.2.2. Interpretación de Resultados

A continuación se propone cómo interpretar los resultados de la verificación de las GI-Nets para determinar el o los elementos del PNC en donde ocurren los errores. Para ello se hace uso de la propiedad de solidez, junto con la estructura modular de las GI-Nets.

Para una dada GI-Net y un estado inicial (GI_N, M_0) puede haber un conjunto de estados muertos. Un *estado muerto* (dead marking) es un estado en el cual no hay transiciones habilitadas. Con lo cual el estado final M_F utilizado en la propiedad de solidez es un estado muerto. Un *estado origen* (home marking) es un estado que puede ser alcanzado desde cualquier otro estado.

Por lo tanto, si una GI-Net GI_N es sólida, se cumplen las siguientes reglas:

1. Debe haber exactamente un *estado muerto* M_D y un *estado origen* M_H, tal que ambos estados sean iguales, y sean además el *estado final* M_F de GI_N, es decir, $M_D = M_H = M_F$.
2. Para cada transición $t \in T$, t no está *muerta*.

La primer regla tiene que ver con los requerimientos 1 y 2 de la propiedad de solidez. Si esta regla no se cumple, hay al menos un estado diferente del estado final M_F en el cual hay un bloqueo en la GI-Net. La ubicación del bloqueo puede ser determinada chequeando cada estado M' que forma parte del conjunto de todos los estados muertos de la GI-Net, tal que $M' \neq M_F$. Para cada modulo $s \in S$ y posición $p \in P^s$, si $M'(p) \neq M_F(p)$, entonces hay un bloqueo originado en el módulo $s \in S$, y al menos una de las transiciones de salida de p (es decir, $p\bullet$) es el origen del bloqueo. Como cada módulo $s \in S$ está asociado a un elemento del PNC P que se está verificando, esto significa que el elemento del PNC P asociado al módulo s genera un bloqueo en el comportamiento del PNC.

La segunda regla tiene que ver con el requerimiento 3 de la propiedad de solidez. Si esta regla no se cumple, significa que hay al menos una interacción que no va a ser llevada a cabo. El módulo GI que causa el comportamiento no deseado puede ser determinado chequeando el conjunto de *transiciones muertas* T_M de la GI-Net. Como cada transición forma parte de un módulo $s \in S$ y dicho módulo está asociado a un elemento del PNC P que se está verificando, se puede determinar que el elemento del PNC P no es alcanzable.

De esta manera, con estas reglas, se puede determinar el elemento o los elementos del modelo de PNC en donde se encuentra el error. Esto permite al diseñador identificar el lugar en donde el comportamiento del PNC debe ser redefinido para corregir el error.

La herramienta CPN Tools que se utiliza en el método para llevar a cabo la verificación de la red de Petri que representa la GI-Net, genera un reporte con los estados muertos y transiciones muertas de una GI-Net. Este reporte puede ser utilizado para realizar la interpretación de los resultados de verificación, de acuerdo a las reglas descriptas en esta sección.

4.3. Verificación de PNCs

En esta sección se describe la aplicación del método formal de verificación propuesto a dos casos de estudio. El primero consiste en la verificación del PNC *gestión de pronóstico de demanda* definido con UP-ColBPIP y descripto en la Sección 3.5 del Capítulo 3. El segundo caso consiste en la verificación de un PNC de fabricación de hardware a pedido definido con BPMN, según un modelo de negocio Build-to-Order (Holweg & Pil 2001). La aplicación del método en ambos casos implicó dos etapas, la formalización y la verificación. Para la etapa de verificación se utiliza la interpretación de resultados de la propiedad de *solidez* propuesta en la Sección 0 del Capítulo 4.

4.3.1. Caso de Estudio 1: Verificación del PNC Gestión de Pronóstico de Demanda

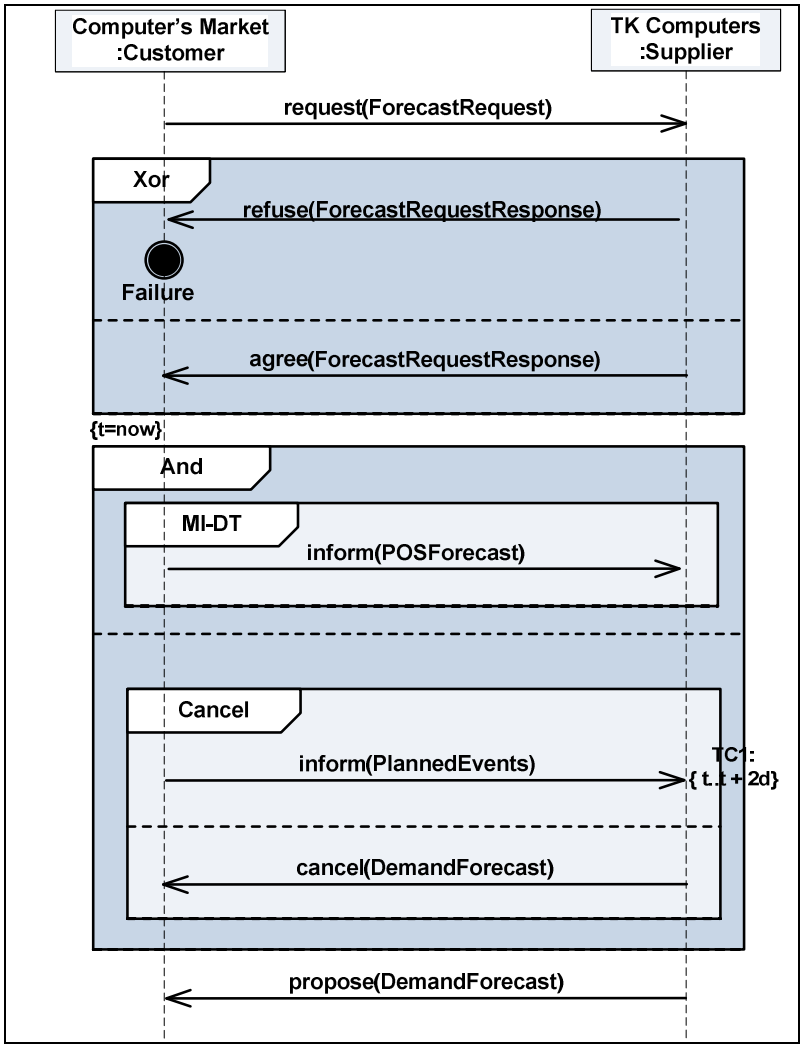


Fig. 4.6. PNC Gestión de Pronóstico de Demanda

Formalización con GI-Nets

Debido a que la formalización de este PNC fue descripta en la Sección 3.5.1 del Capítulo 3, se reutiliza el modelo formal GI-Nets generado en dicho capítulo. La Figura **4.7** muestra la estructura en árbol de la GI-Net resultante que formaliza este PNC. Dicha GI-Net se expresa en un documento XML en formato CPN Tools.

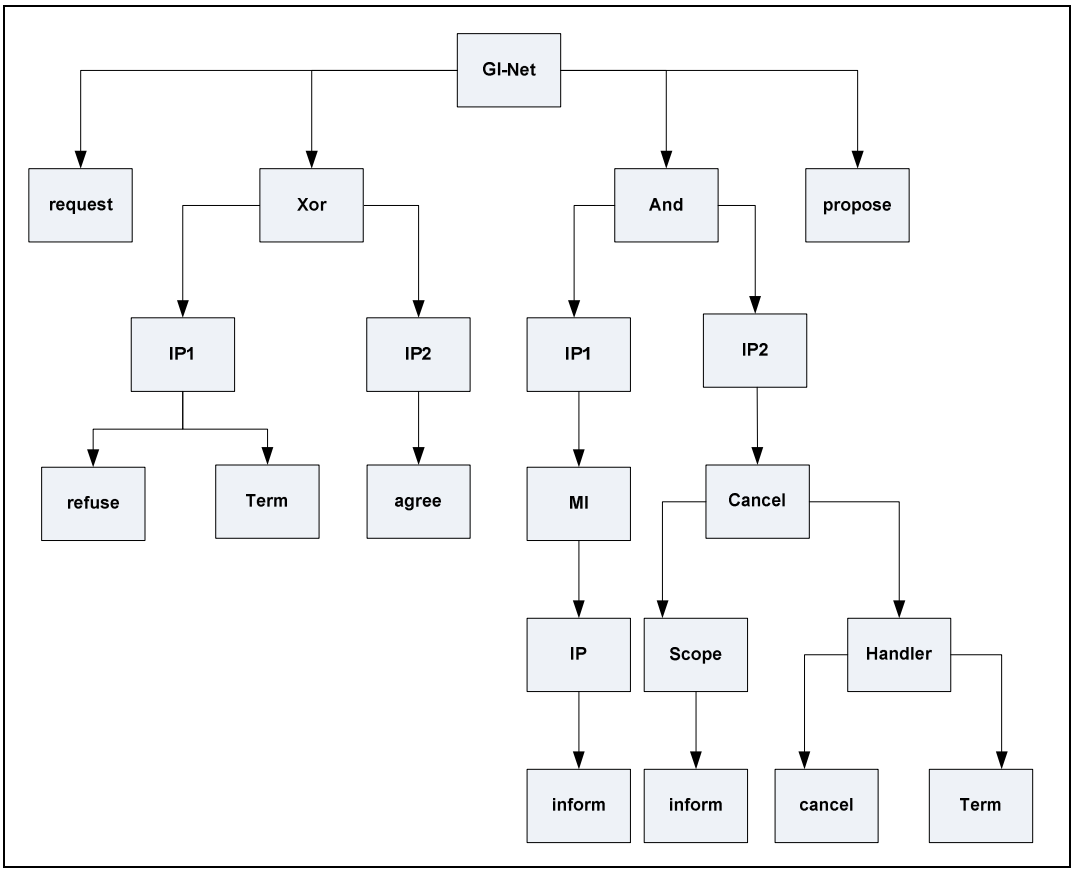


Fig. 4.7. Estructura de la GI-Net del PNC Gestión de Pronóstico de Demanda

Verificación de la solidez del PNC

Para determinar si el PNC gestión de pronóstico de demanda es sólido, es necesario determinar si la GI-Net que formaliza a dicho PNC es sólida. La interpretación de los resultados se obtiene siguiendo la especificación realizada en la Sección 4.2.2. Para verificar y determinar la solidez de la GI-Net se utilizó la herramienta CPN Tools. El reporte generado por dicha herramienta para esta GI-Net (Figura **4.8**) indica dos estados muertos (ítem "Dead Markings"). Los números que se muestran son los identificadores de los nodos del espacio de estado generado por la herramienta CPN Tools que contienen los estados muertos. Por lo tanto, al tener dos estados muertos se determina que la GI-Net no es sólida, ya que no cumple con la regla (1). Uno de estos estados muertos corresponde a la finalización correcta de la GI-Net, es decir, al estado en que todas las posiciones de interacción están vacías, excepto la posición final de la GI-Net, y además las posiciones de

control están en sus respectivos estados iniciales. Por lo tanto, se deduce que, si bien la GI-Net no es sólida, existe al menos un camino de ejecución en el flujo de control que hace que la GI-Net finalice de manera adecuada.

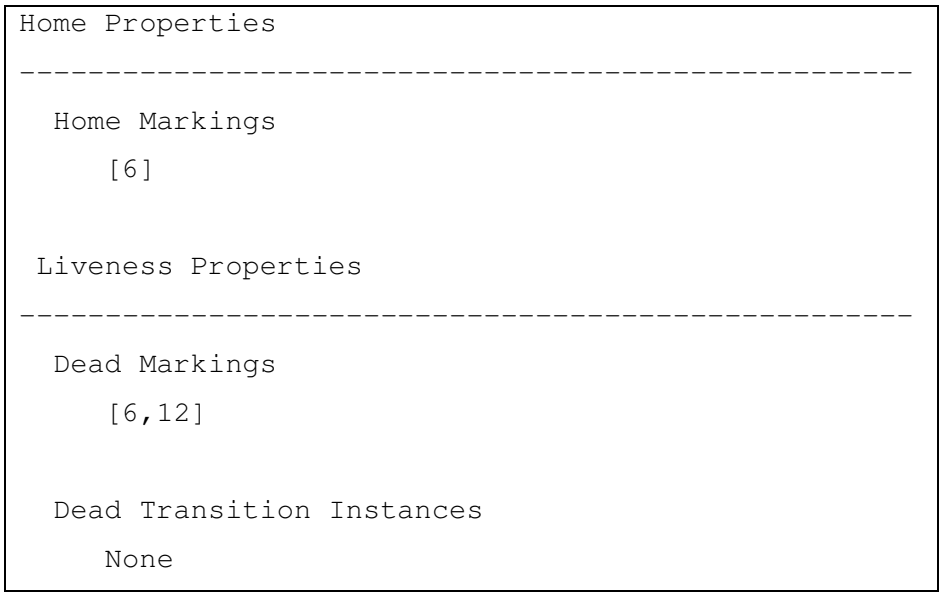


Fig. 4.8. Resultados de verificación devueltos por la herramienta CPN Tools

El otro estado corresponde a un bloqueo causado por el elemento *Cancel* que se encuentra en uno de los caminos del elemento *And*. El problema reside en que el comportamiento formal definido para el constructor *And* establece que todos los caminos paralelos deben ser sincronizados. En esta situación, si ocurre un evento de tiempo y se dispara una excepción, la sincronización nunca se va a llevar a cabo, ya que el elemento *Cancel* finaliza el PNC una vez que termina su ejecución y por lo tanto no se puede realizar la sincronización del *And*. Debido a que los caminos de interacción en el *And* son paralelos, pueden quedar interacciones en ejecución aún cuando el PNC haya llegado a un estado de finalización mediante el elemento *Cancel*.

La solución a este problema se muestra en la Figura **4.9**, en la cual el elemento *And* pasa a estar contenido dentro del alcance del *Cancel*. De esta manera, si ocurre una excepción, el alcance de la misma abarca a todos los caminos del elemento *And*, evitando así el bloqueo en la sincronización de dicho constructor.

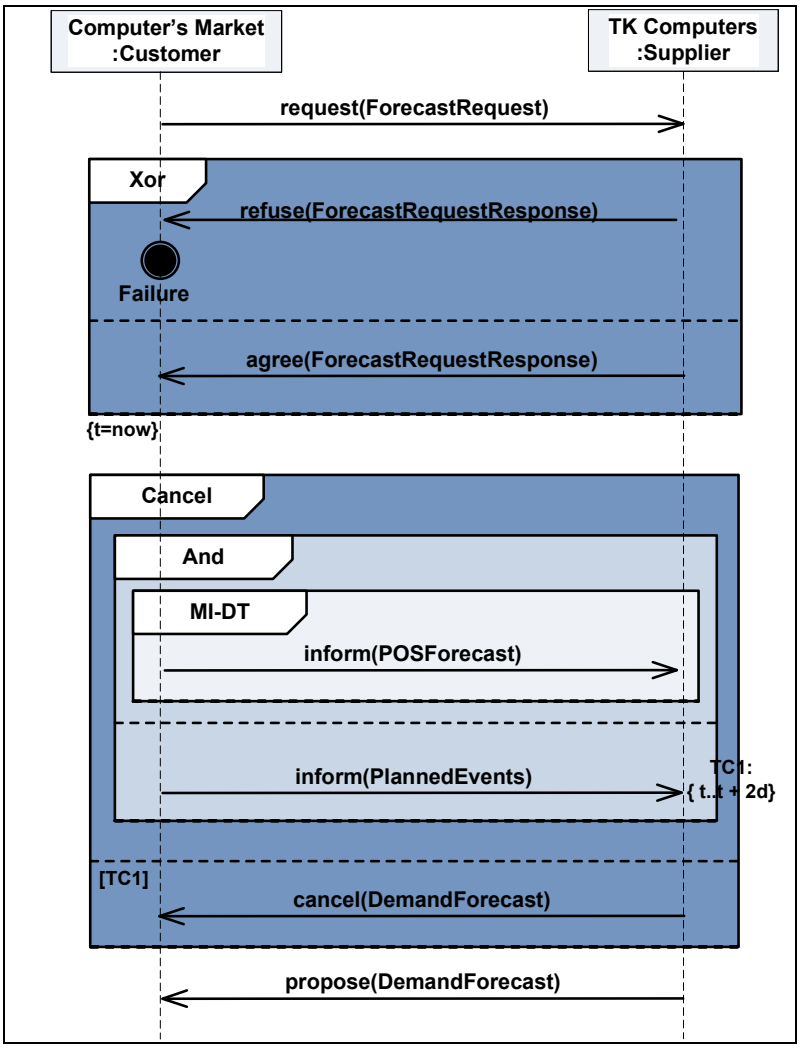


Fig. 4.9. PNC Gestión de Pronóstico de Demanda sin bloqueos

4.3.2. Caso de Estudio 2: Fabricación de Hardware a Pedido

Este caso de estudio refiere a un PNC adaptado de un escenario propuesto en (Rosenberg et al. 2007). El PNC definido en BPMN (Figura **4.10**) consiste de un cliente, un fabricante, y proveedores de CPUs, placas madre, y discos rígidos. El fabricante ofrece equipamiento de hardware ensamblado a sus clientes. Para esto implementó un modelo de negocio de fabricación a pedido BTO (cuya sigla se refiere a Build-to-Order) (Holweg &

Pil 2001). El fabricante dispone de un conjunto de componentes de hardware en su stock y en caso de ser necesario solicita a sus proveedores los componentes faltantes.

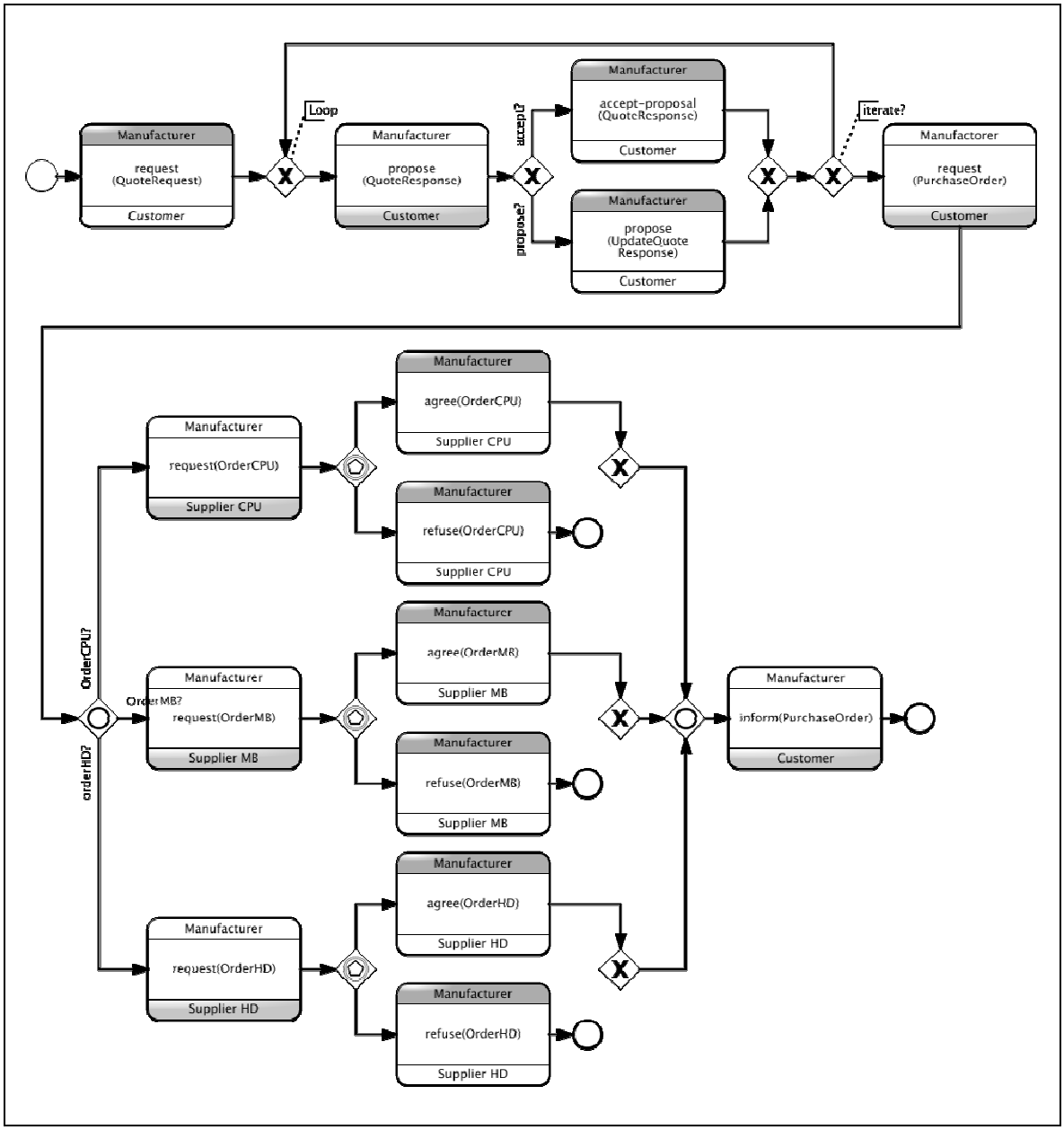


Fig. 4.10. PNC Fabricación de Hardware a Pedido

El PNC comienza cuando el cliente le envía al fabricante una solicitud de cotización *request(QuoteRequest)* con los detalles sobre el equipamiento de hardware requerido. Luego el fabricante le responde al cliente con una cotización propose(*QuoteResponse*). El

cliente puede aceptar o rechazar la propuesta de cotización. Este ciclo se repite hasta que acuerdan la cotización.

Si se llega a un acuerdo el cliente le envía una orden de compra (*PurchaseOrder*) al fabricante. Dependiendo del stock de hardware, el fabricante tiene que realizar un pedido a sus proveedores indicando los componentes de hardware que necesita. En caso que el fabricante necesite obtener componentes de hardware para cumplir con la orden de compra envía la orden de hardware necesaria al respectivo proveedor (*OrderCPU*, *OrderMB*, y *OrderHD*). Luego, el proveedor acuerda (*agree*) el envío del hardware al fabricante, o rechaza (*refuse*) el pedido en caso de no poder satisfacer la solicitud. Finalmente, el fabricante envía una respuesta al cliente indicando si puede satisfacer la orden de compra.

Formalización con GI-Nets

La Figura **4.11** muestra la estructura de la GI-Net del PNC definido con BPMN. Cada nodo del árbol representa un módulo GI concreto que se genera a partir de uno o más elementos del PNC. El módulo que es raíz del árbol formaliza al PNC, el cual es una secuencia de elementos. El módulo *Loop* formaliza al bloque de elementos indicados como *Loop* e *iterate?*. Los módulos *Exc/Exc* formalizan al bloque de elementos de BPMN formado por dos *Exclusive* gateways. Los módulos *EndEv* formalizan a los elementos de BPMN *End Event*. Los módulos *request*, *refuse*, *agree*, *propose*, e *inform* formalizan a las tareas de coreografía. Por último, el módulo *Imp/Imp* formaliza al bloque de elementos de BPMN formado por dos *Implicit* gateways.

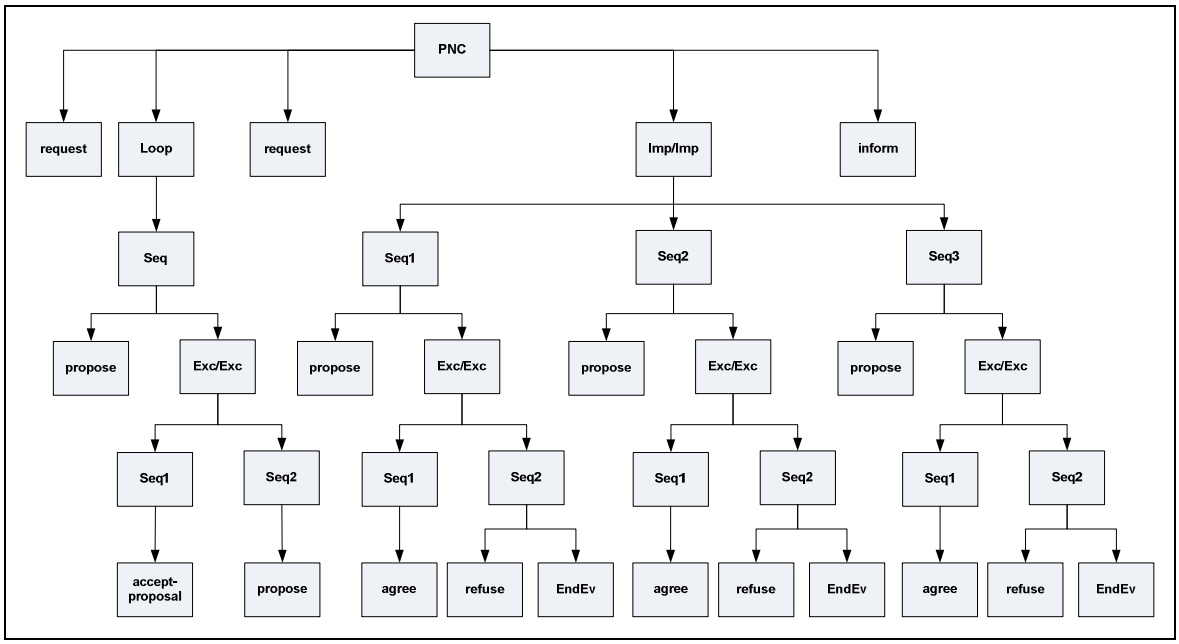


Fig. 4.11. Estructura de la GI-Net del PNC Fabricación de Hardware a Pedido

La Figura **4.12** muestra la definición de los módulos GI concretos de la GI-Net generada. No se muestran módulos repetidos. La generación de cada módulo se llevó a cabo utilizando los módulos abstractos de los constructores de BPMN definidos en la Sección 3.4 del Capítulo 3.

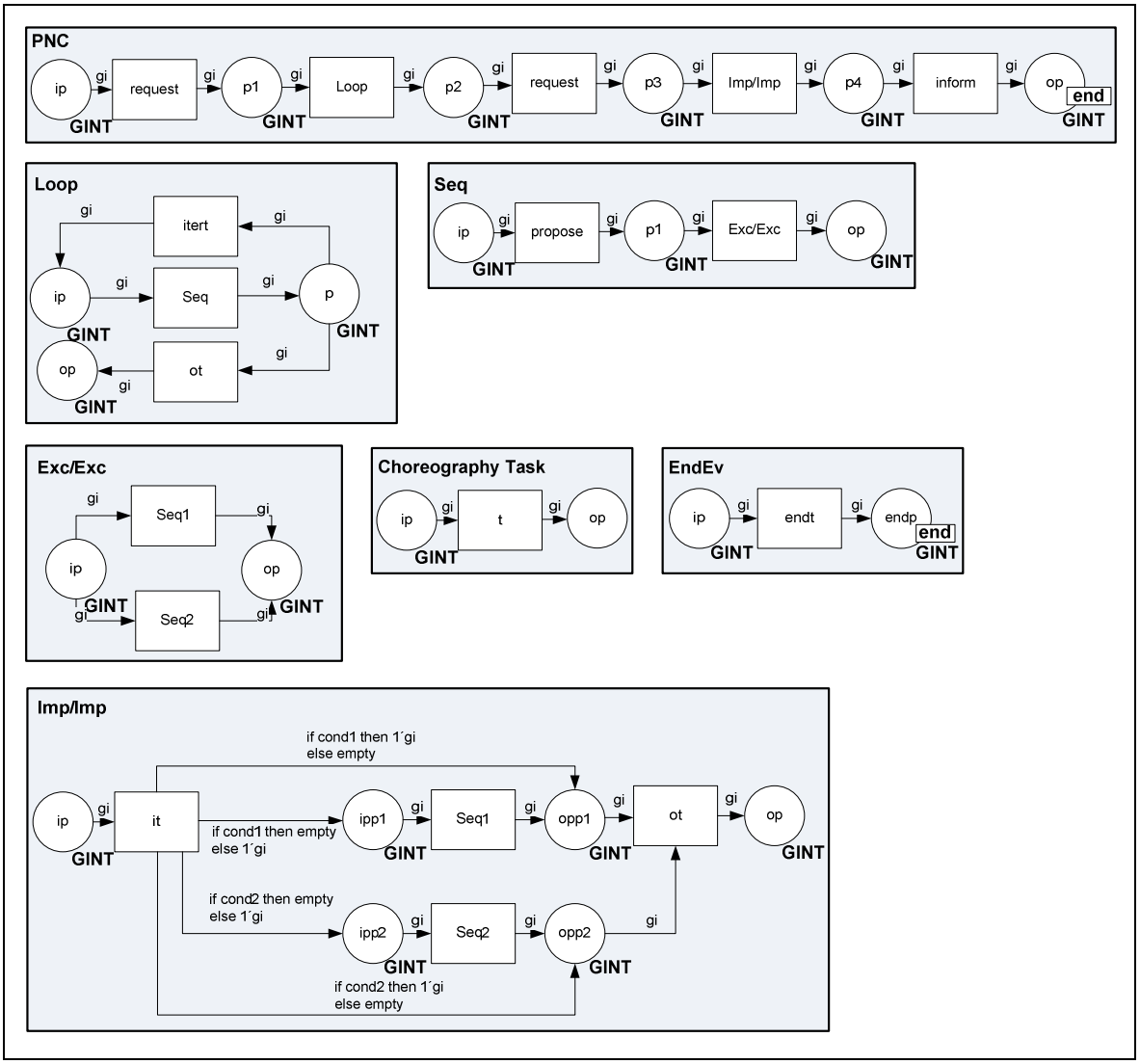


Fig. 4.12. Módulos GI concretos del PNC Fabricación de Hardware a Pedido

Verificación de la solidez del PNC

Para determinar si el PNC Fabricación de Hardware a Pedido es sólido, es necesario determinar si la GI-Net que formaliza dicho PNC es sólida. La interpretación de los resultados se obtiene siguiendo la especificación realizada en la Sección 4.2.2. Para determinar la solidez de la GI-Net se utilizó la herramienta CPN Tools.

El reporte generado por dicha herramienta para esta GI-Net (Figura **4.13**) indica ocho estados muertos (ítem "Dead Markings"). Por lo tanto, se determina que la GI-Net no

es sólida, ya que no cumple con la regla (1). Uno de estos estados muertos corresponde a la finalización correcta de la GI-Net según la propiedad de *solidez*, es decir, al estado en que todas las posiciones de interacción están vacías, excepto la posición final de la GI-Net, y además las posiciones de control están en sus respectivos estados iniciales. Por lo tanto, se deduce que, si bien la GI-Net no es sólida, existe al menos un camino de ejecución en el flujo de control que hace que la GI-Net (y por lo tanto el PNC) finalice de manera adecuada.

```
Home Properties
------------------------------------------------
  Home Markings
     [24]

 Liveness Properties
------------------------------------------------
  Dead Markings
     [24,26,33,48,50,64,72,78]

  Dead Transition Instances
     None
```

Fig. 4.13. Resultados de verificación devueltos por la herramienta CPN Tools

Los otros estados corresponden a bloqueos causados por los elementos *End Event* dentro de los constructores *Implicit/Implicit*, definidos luego de los mensajes *refuse(OrderCPU)*, *refuse(OrderMB)*, y *refuse(OrderHD)*. El problema reside en que el comportamiento formal definido para el constructor *Implicit/Implicit* establece que todos los caminos paralelos que inician su ejecución deben ser sincronizados. En esta situación, si en alguno de los caminos paralelos del *Implicit* gateway se opta por tomar el camino que contiene un *End Event*, los demás caminos en paralelo no van a poder ser sincronizados, con lo cual se produciría un bloqueo. Además, debido a que los caminos de interacción en el *Implicit* son paralelos, se puede dar la situación en la que ocurran interacciones aún

cuando el PNC haya llegado a un estado de finalización mediante un evento de terminación *End Event*.

La solución a este problema se muestra en la Figura **4.14**, en la cual se eliminan los elementos *End Event* mencionados anteriormente. De esta manera, el PNC no finaliza cuando los proveedores rechazan los pedidos del fabricante, ya que el PNC continúa con el fabricante que informa al cliente el estado del pedido.

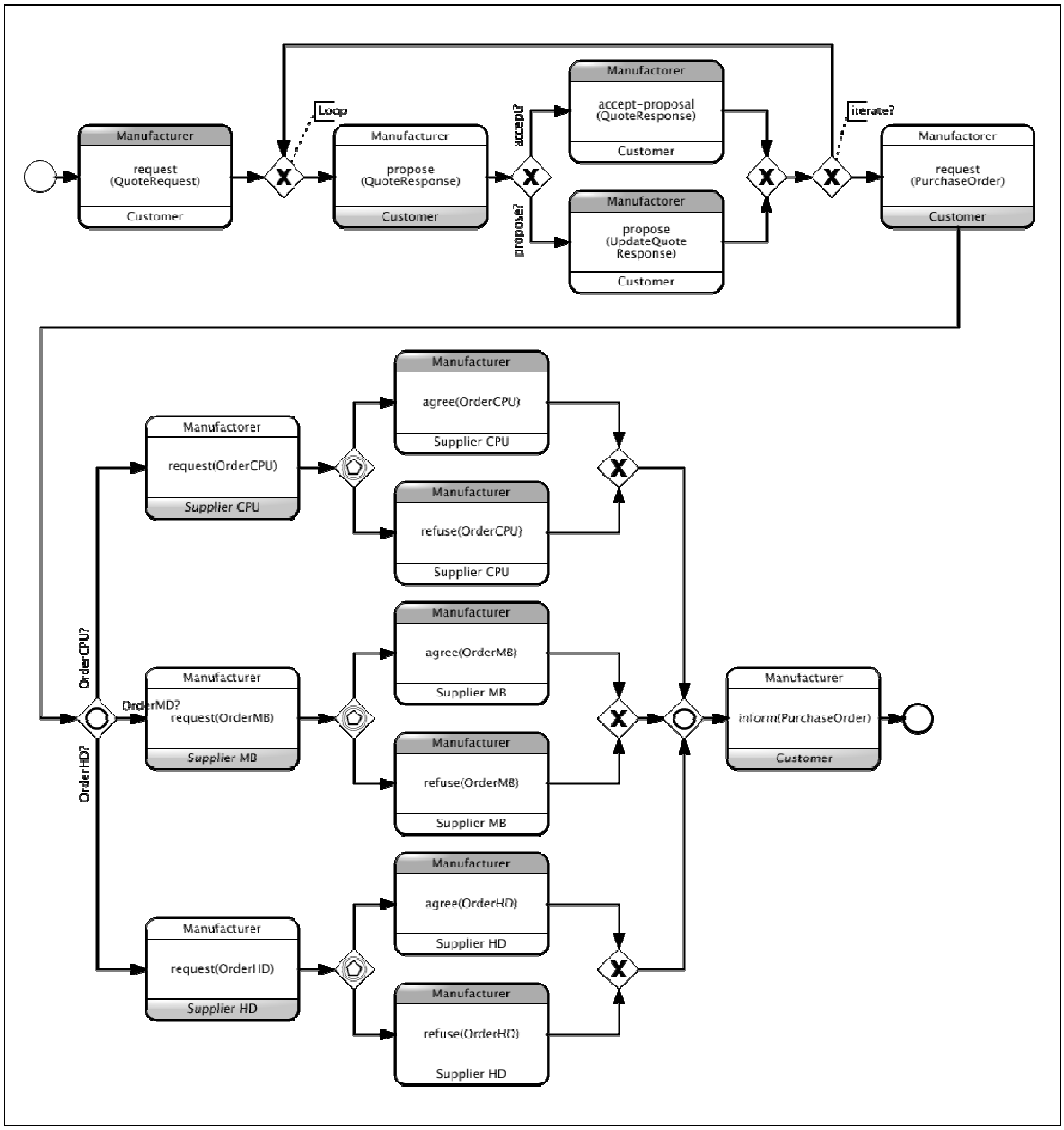


Fig. 4.14. PNC Fabricación de Hardware a Pedido sin bloqueos

4.4. Conclusiones

El método formal de verificación de PNCs propuesto en este capítulo permite en una primer etapa (formalización) generar un modelo formal de PNC a partir de un modelo conceptual o especificación de PNC definido con un lenguaje no formal; y en una segunda etapa (verificación) analizar el modelo formal, determinar si satisface ciertas propiedades e interpretar los resultados.

Mediante el uso del lenguaje formal GI-Nets y el patrón de transformación para GI-Nets, el método permite generar el modelo formal de un PNC como una GI-Net en formato PNML y CPN Tools. En base a la propiedad de *solidez* definida y la interpretación de resultados propuesta, haciendo uso de la herramienta de redes de Petri CPN Tools analiza la GI-Net generada, permitiendo determinar si un PNC tiene bloqueos o elementos no alcanzables.

El uso del lenguaje formal GI-Nets posibilita que el método de verificación sea independiente de la semántica de cualquier lenguaje específico de modelado de PNCs, lo cual lo hace flexible y adaptable a ser utilizado para verificar PNCs definidos con distintos lenguajes y, por lo tanto, en diferentes etapas del desarrollo dirigido por modelos de PNCs.

Haciendo uso de patrones de transformación para generar modelos formales de PNC con GI-Nets, se aplican los principios del desarrollo dirigido por modelos a un método de verificación. El uso de patrones de transformación de modelos posibilita generar automáticamente el modelo formal GI-Net y realizar la verificación automáticamente utilizando la herramienta de redes de Petri CPN Tools. La generación de modelos de GI-Nets en formato PNML permite desacoplar el método de cualquier herramienta de verificación de redes de Petri, ya que hay herramientas que utilizan este estándar, o bien a partir de PNML es posible generar redes de Petri para otros tipos de herramientas, tal como se lo realizo para CPN Tools

La propiedad Solidez de Interacción Global, propuesta para ser satisfecha por una GI-Net, relaja la restricción establecida por la definición clásica de la propiedad de solidez, la cual exige que el estado final de una red de Petri debe tener solo una señal en la posición final. De esta forma, se permite la verificación de flujos de control complejos y se evita un estado final aleatorio, el cual podría resultar de situaciones inesperadas del

comportamiento. Esto se debe a que la propiedad de solidez definida contempla que las posiciones de control de una GI-Net pueden tener un conjunto predefinido de señales, mientras que las posiciones de interacción no deben tener señales, excepto la posición de interacción final. Esta propiedad permite concluir acerca de la ausencia o no de bloqueos y elementos no alcanzables en un PNC.

El enfoque propuesto de interpretación de los resultados del análisis de una GI-Net con herramientas de redes de Petri genera resultados de verificación que permiten determinar no sólo la solidez de la GI-Net, sino también los elementos en donde hay bloqueos o son no alcanzables. Las propiedades clásicas de redes de Petri, tales como estados muertos y estado origen, son interpretadas para determinar los elementos donde hay errores en un PNC. La estructuración y modularidad de las GI-Nets a través de módulos GI que representan los elementos de un PNC, posibilita esta interpretación, facilitando y mejorando la tarea de corrección del PNC por parte del diseñador.

La aplicación del método de verificación propuesto a casos de estudio de modelos conceptuales de PNCs, definidos con los lenguajes UP-ColBPIP y BPMN, permitió la verificación de modelos UP-ColBPIP y BPMN con flujos de control complejos, tales como sincronización avanzada, excepciones, cancelaciones, e instancias múltiples. Además, permitió mostrar el proceso de cómo interpretar los resultados de análisis de las GI-Nets que formalizan dichos PNCs, según la propiedad de solidez propuesta, para detectar los elementos de los modelos que son origen de errores en el comportamiento de los PNCs.

5 Verificación de Procesos de Negocio Colaborativos basada en Anti-Patrones de Comportamiento

En este capítulo se presenta un método de verificación de modelos estructurados de PNCs basado en anti-patrones de comportamiento. Se define el concepto de anti-patrón de comportamiento (Sección 5.1) y se presenta un enfoque que permite especificar los anti-patrones de comportamiento para un lenguaje de PNCs (Sección 5.2). Se describe la especificación de anti-patrones de comportamiento para el lenguaje UP-ColBPIP y se presentan ejemplos de verificación con los mismos (Sección 5.3). Finalmente, se presentan las conclusiones (Sección 5.4).

5.1. Anti-patrones de Comportamiento para PNCs

El concepto de anti-patrón surge a partir del concepto de patrón. Un patrón captura una solución a un problema repetitivo en un contexto particular. El uso de patrones permite capturar diseños y prácticas basadas en experiencias exitosas de manera tal que sea posible llevar a cabo su reutilización, exponiendo problemas comunes y sus respectivas soluciones (Gamma et al. 1995). En el área de diseño de procesos y "workflows" se han propuesto patrones que capturan los diferentes constructores de flujo de control recurrentes en el modelado de procesos de negocio o "workflows" (Russell et al. 2006).

Un anti-patrón, en cambio, captura una solución incorrecta a un problema de diseño, la cual debe ser evitada, y por lo tanto representa diseños y prácticas inadecuadas (Brown, Malveau & Mowbray 1998). Usualmente, capturan también descripciones de soluciones, indicando los cambios que se deben realizar para obtener una solución correcta.

En el área de procesos de negocio se han propuesto anti-patrones para detectar errores en el comportamiento de modelos de procesos (Koehler & Vanhatalo 2007; Awad & Puhlmann 2008; Gruhn & Laue 2009; Kühne et al. 2010; Han et al. 2013). Los anti-patrones definidos en estos trabajos tienen como limitación que no consideran flujos de

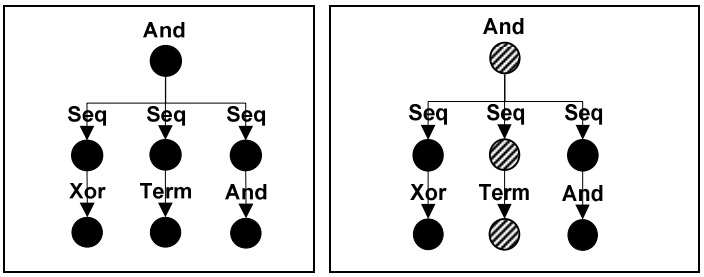
control complejos, y su descubrimiento y especificación generalmente se realiza mediante estudios empíricos que evalúan una determinada cantidad de procesos existentes, obtenidos de casos de estudio o repositorios de procesos. Esta práctica presenta dos inconvenientes principales: en primer lugar, no siempre es posible disponer de un conjunto suficientemente amplio de casos reales de procesos que puedan ser analizados y, en segundo lugar, los procesos seleccionados pueden estar sesgados por los errores de diseño más usuales quedando por descubrir otros problemas menos comunes. Esto puede afectar el criterio de completitud de un método de verificación basado en anti-patrones.

Para evitar estos inconvenientes y proveer un método que satisfaga el criterio de completitud además del criterio de rendimiento, se propone un enfoque sistemático para especificar anti-patrones de comportamiento de un lenguaje estructurado de PNCs. A partir de los anti-patrones que se obtienen de un lenguaje de PNC, el proceso de verificación se limita a una técnica de búsqueda de coincidencias de anti-patrones en los modelos o especificaciones de PNCs definidos con dicho lenguaje.

5.2. Especificación de Anti-Patrones de Comportamiento de PNCs

En esta tesis se define como *anti-patrón de comportamiento de PNCs* a una situación predefinida y conocida de una definición deficiente del flujo de control de PNCs. Estos anti-patrones se definen específicamente para cada lenguaje, debido a que la semántica de comportamiento de un constructor de un lenguaje de PNCs puede diferir de la semántica de comportamiento del mismo constructor de otro lenguaje.

Para definir un anti-patrón de comportamiento de PNCs a partir del análisis de los constructores de un lenguaje, primero es necesario *descubrir* las combinaciones de elementos que pueden generar inconvenientes en el comportamiento de un PNC, y luego *generalizar* las relaciones entre dichos elementos, de manera tal que el anti-patrón pueda ser utilizado para detectar problemas en el comportamiento de distintos PNCs.



(a) (b)

Fig. 5.1. PNC estructurado no sólido

Por ejemplo, la Figura **5.1** a) muestra la vista estructurada de un modelo de PNC utilizando la notación descripta en el Capítulo 2, Sección 2.3.2. Se desea descubrir si este PNC contiene elementos que definen un anti-patrón de comportamiento. Dicho PNC contiene un elemento *And* con tres caminos paralelos, que están compuestos respectivamente de un *Xor*, un *Termination* y un *And*. Este PNC es no sólido ya que el elemento *Termination* dentro del *And* impide la sincronización del mismo, ocasionando un bloqueo. Estos elementos se muestran con una textura de líneas oblicuas en la Figura **5.1** b).

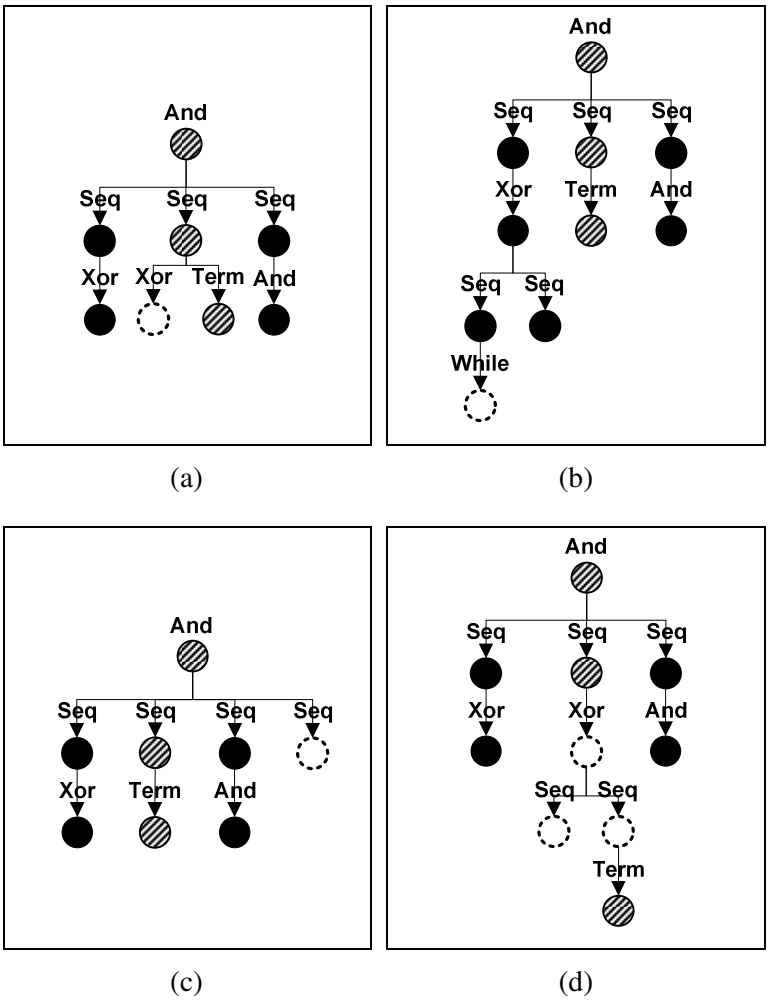


Fig. 5.2. Evolución de un PNC estructurado no sólido

Para descubrir los elementos que forman parte del anti-patrón, se muestran en la Figura **5**.**2** distintos modelos estructurados de PNCs que surgen de adicionar elementos (mostrados con líneas de puntos) al PNC de la Figura **5.1**. Estos PNCs son también no sólidos, debido a que mantienen la misma combinación de elementos *And* y *Termination* del PNC de la Figura **5.1**. En estos ejemplos, la combinación de los elementos *And* y *Termination* (Figura **5.1** b)) es la causa de la presencia de bloqueos en todos los PNCs de la Figura **5**.**2**. Lo mismo sucede con cualquier otro PNC que tenga dicha combinación de elementos. Por lo tanto, esta combinación de elementos es determinante para la presencia de bloqueos en PNCs y da origen a un anti-patrón.

Sin embargo, para definir un anti-patrón de comportamiento no basta con descubrir la combinación de elementos que pueden generar problemas, tal como la combinación de

And con *Termination*, sino que es además necesario *generalizar* la relación entre los mismos. Esto implica determinar el tipo de relación entre dichos elementos en la vista de árbol estructurado del PNC. Esta relación puede ser directa o indirecta. En una *relación directa*, uno de los elementos está anidado dentro del otro. En una *relación indirecta*, un elemento puede ser accedido desde otro elemento recorriendo la estructura de elementos anidados del primero.

En los PNCs mostrados en las Figuras **5**.**2** a), b) y c), los elementos determinantes de la no solidez están conectados de manera directa, el *Termination* está anidado dentro del *And*. En cambio, en el PNC mostrado en la Figura **5**.**2** d) dichos elementos están relacionados de manera indirecta, ya que el *Termination* no está anidado directamente dentro del *And*, sino que está dentro de un *Xor* el cual a su vez está anidado en el *And*. Este PNC contiene la misma combinación de elementos que los anteriores pero relacionados de manera diferente.

Para especificar anti-patrones de comportamiento de un lenguaje de PNCs, esto es, descubrir los elementos que forman parte de los anti-patrones y luego generalizar las relaciones entre estos, se proponen las siguientes tareas:

- *Generar un perfil de no solidez del lenguaje de PNCs seleccionado*. Para ello, primero se generan todos los *PNCs mínimos* que pueden ser definidos con el lenguaje de PNC. Éstos representan todas las combinaciones mínimas de elementos posibles de ser generadas con dicho lenguaje. Luego, se determina la solidez de cada PNC mínimo aplicando un método formal de verificación de PNC. El conjunto de PNCs mínimos no sólidos de un lenguaje define el *perfil de no solidez* para dicho lenguaje.
- *Obtener los PNC mínimos determinantes a partir del perfil de no solidez del lenguaje*. En un PNC mínimo no sólido que forma parte del perfil de no solidez del lenguaje, puede haber elementos que conducen a la no solidez, mientras que otros no causan problemas. Además, varios PNCs mínimos pueden compartir elementos que conducen a la no solidez. Esto da lugar a la obtención de *PNC mínimos determinantes*, que son aquellos que sólo tienen elementos que conducen a la no solidez. De esta manera se reduce el número de PNCs del

perfil de no solidez. A partir de estos PNCs mínimos determinantes es posible descubrir los constructores que conforman los anti-patrones de comportamiento.

- *Definir los anti-patrones*. Por cada PNC mínimo determinante se define un anti-patrón, indicándose los elementos del mismo, donde cada elemento referencia a un constructor de un lenguaje. Luego se define la *regla* del anti-patrón que establece las relaciones entre los elementos del anti-patrón que dan lugar a inconvenientes en el comportamiento de un PNC. Esto permite que el anti-patrón pueda ser aplicado a cualquier PNC definido con el lenguaje.

A continuación se introduce el concepto de PNC mínimo, y luego se describen cada una de las tareas mencionadas.

5.2.1. PNCs Mínimos y No-Mínimos

Un modelo estructurado de un PNC puede evolucionar en distintas dimensiones con la adición de nuevos elementos. La evolución se puede estudiar a partir de las posibles combinaciones entre los elementos de un proceso. En modelo estructurado de un PNC, los elementos se combinan en una relación padre/hijo, donde existe un conjunto de elementos C (los hijos) que están directamente conectados (anidados) a un elemento e (el padre), es decir, $C = Post(e)$, donde $Post(e)$ refiere al conjunto de sucesores directos del elemento e. La combinación puede ser secuencial o jerárquica.

Definición 5.1. ***(Combinación secuencial)*** Dado un elemento e y un conjunto ordenado de elementos C, se dice que e y los elementos de C establecen una combinación secuencial de elementos si $C = Post(e)$ y se cumple que: (1) e es un elemento *Sequence* y (2) para cada $c \in C$, c es distinto de *Sequence*.

En una *combinación secuencial*, e es un elemento de tipo *Sequence* y C es un conjunto ordenado de elementos distinto de *Sequence* y corresponden a los hijos de e.

Definición 5.2. ***(Combinación jerárquica)*** Dado un elemento e y un conjunto de elementos C, se dice que e y los elementos de C establecen una combinación jerárquica de elementos si $C = Post(e)$ y se cumple que: (1) e es distinto de *Sequence*; (2) para cada elemento $s \in C$, se cumple que s es *Sequence*; (3) existe al menos un elemento *Sequence* $s \in C$ y al menos un elemento $e_1 \in Post(s)$, tal que e_1 es distinto de *Sequence*.

En una *combinación jerárquica*, el padre no es *Sequence* y C es un conjunto de elementos *Sequence* (hijos de *e*), donde al menos uno de los elementos *Sequence* contiene al menos un elemento de flujo de control que no es *Sequence*. Esto define combinaciones de anidamiento entre elementos de flujo de control, tal como un *Xor* con un *And* anidado.

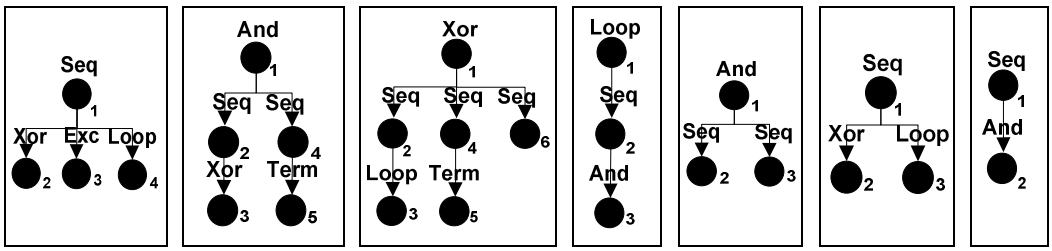
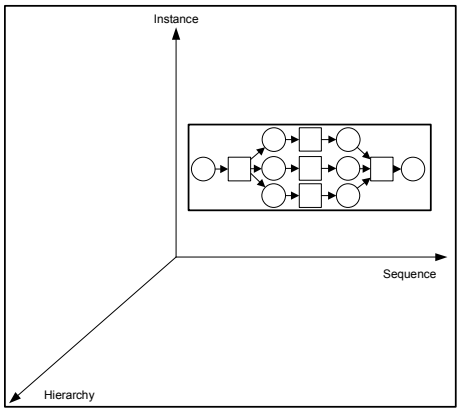


Fig. 5.3. Combinaciones de elementos

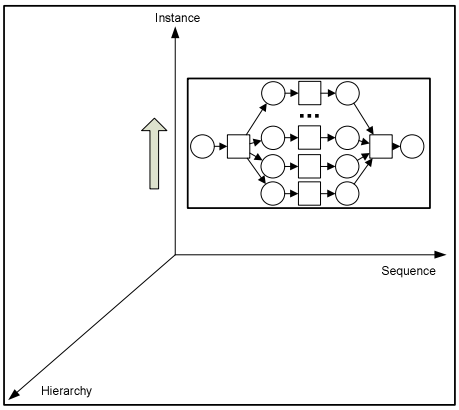
La Figura **5.3** muestra combinaciones de elementos de modelos estructurados de PNCs basados en constructores utilizados comúnmente en diferentes lenguajes: *Seq (Sequence)*, *Xor*, *And*, *Exception (Exc)*, *Loop*, y *Termination (Term)*. Las Figuras **5.3** a) y f) muestran combinaciones secuenciales, y las Figuras **5.3** b), c), y d) muestran combinaciones jerárquicas. La Figura **5.3** a) muestra una combinación secuencial de elementos *Xor*, *Exception* y *Loop* que se ejecutan en secuencia uno luego de otro. La Figura **5.3** b) muestra una combinación jerárquica en la cual los elementos *Xor* y *Termination* están anidados en los caminos de un *And* representados por una secuencia cada uno. Las Figuras **5.3** c) y d) muestran combinaciones jerárquicas en las cuales se combinan en la primera un *Xor* con un *Loop* y un *Termination*, y en la segunda un *Loop* con un *And*. La Figura **5.3** e) no representa una combinación secuencial ni una jerárquica debido a que cada secuencia o camino del *And* no contiene elementos. Los elementos de la Figura **5.3** g) no

representan una combinación secuencial, debido a que hay un sólo elemento dentro de la secuencia, por lo cual, el elemento *And* no puede ser combinado en secuencia con algún otro elemento.

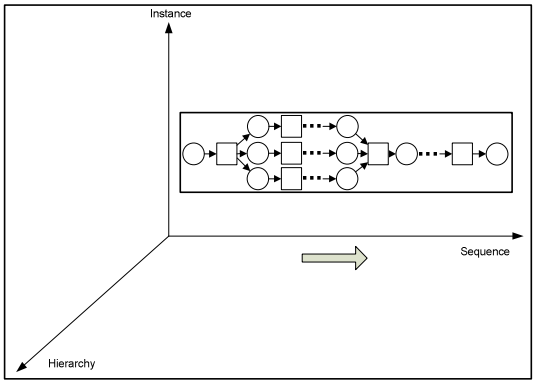
Considerando estos tipos de combinaciones secuenciales y jerárquicas de elementos, se llevó a cabo un análisis de la estructura de modelos estructurados de PNCs cuando crecen mediante la adición de nuevos elementos. El análisis se asemeja al estudio del límite de una función matemática. Si se considera a un PNC como la imagen de una función matemática cuyo dominio se representa por una o más dimensiones de PNCs, sería posible estudiar la estructura de un modelo de PNC cuando el límite de dicha función tiende a infinito.



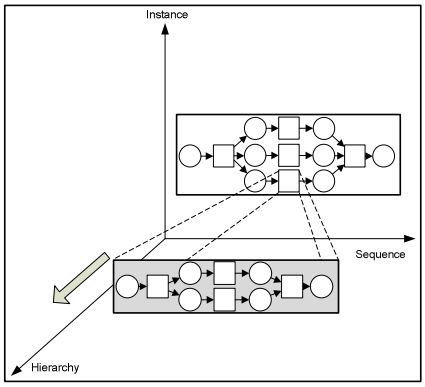
(a) PNC Original



(b) Evolución de la dimensión de instancia



(c) Evolución de la dimensión de secuencia



(d) Evolución de la dimensión de jerarquía

Fig. 5.4. Dimensiones de un PNC estructurado

Se identificaron tres dimensiones en las cuales la estructura de un modelo de proceso puede evolucionar con la adición de nuevos elementos (Figura **5.4**): instancia, secuencia, y jerarquía. La Figura **5.4** a) muestra un sistema de ejes coordenados en el cual se encuentra ubicado un elemento *And* definido con redes de Petri. Cada uno de los ejes representa una dimensión. La *dimensión de instancia* determina la evolución de un PNC en base a cada posible instancia de los constructores que componen al mismo. Por ejemplo, la Figura **5**.**4** b) muestra una nueva instancia del constructor *And* mostrado en la Figura **5.4** a), en la cual se agrega una secuencia que representa un nuevo camino del *And*.

La *dimensión de secuencia* determina cómo un PNC evoluciona cuando se agregan nuevos elementos en forma secuencial (Figura **5.4** c)). Considera combinaciones secuenciales de elementos.

La *dimensión de jerarquía* determina cómo un proceso evoluciona cuando se agregan nuevos elementos en forma jerárquica (Figura **5.4** d)). Considera combinaciones jerárquicas de anidamiento entre elementos, tal como un *Xor* anidado dentro de un *And*.

Debido a que un constructor usualmente tiene infinitas instancias (dimensión de instancias), y la combinación entre instancias de diferentes constructores (dimensiones secuencial y jerárquica) no está limitada a ningún número, el dominio de estas tres dimensiones es infinito, es decir, cualquier modelo estructurado de un PNC puede crecer de manera infinita mediante la adición de nuevos elementos a lo largo de estas dimensiones.

Las dimensiones tienen límites superiores e inferiores que son importantes para estudiar tanto la estructura como el comportamiento de modelos estructurados de PNCs. En el límite superior, los modelos pueden crecer de manera infinita, ya que existen infinitas combinaciones de elementos, mientras que en el límite inferior, los modelos están limitados a un número dado de elementos, debido a que existe un número limitado de elementos a combinar.

Teniendo en consideración los límites mencionados, un elemento de flujo de control puede ser definido como mínimo o no-mínimo. Un elemento de un modelo de PNC es mínimo si el mismo se encuentra en el límite inferior de la dimensión de instancia. Para generar un elemento mínimo es necesario conocer las cardinalidades y restricciones de las relaciones que existen entre los elementos del metamodelo de un lenguaje. Por ejemplo, el constructor *And* puede estar restringido a tener al menos dos caminos paralelos. Por lo

tanto, un *And* mínimo está compuesto de dos caminos paralelos. En este caso, el *And* con dos caminos se encuentra en el límite inferior de la dimensión de instancia, ya que no existe ningún otro elemento *And* que pueda tener una cantidad menor de caminos paralelos.

Definición 5.3. ***(Elemento mínimo)*** Dado un constructor C y un elemento de flujo de control e que es instancia de C, se dice que e es un *elemento mínimo* si se encuentra en el límite inferior de la dimensión de instancia. De otra manera, e es un *elemento no-mínimo*.

Un *PNC mínimo* es un modelo estructurado de PNC compuesto de una combinación de elementos mínimos, y puede ser secuencial o jerárquico.

Definición 5.4. ***(PNC secuencial mínimo)*** P_M es un *PNC secuencial mínimo* si está compuesto solamente de una combinación secuencial de dos elementos mínimos de flujo de control e_1 y e_2, y se denota como $P_M = \{r, e_1, e_2\}$, donde r es la raíz de P_M.

Definición 5.5. ***(PNC jerárquico mínimo)*** P_M es un *PNC jerárquico mínimo* si está compuesto solamente de una combinación jerárquica de tres elementos mínimos de flujo de control r, e_1, y e_2, y se denota como $P_M = \{r, e_1, e_2\}$, o una combinación jerárquica de dos elementos mínimos, y se denota como $P_M = \{r, e_1\}$, donde r es la raíz de P_M.

Definición 5.6. ***(PNC mínimo)*** *Sea* P_M un PNC estructurado. P_M es un *PNC mínimo* si está compuesto solamente de un elemento mínimo, o si es un PNC secuencial o jerárquico mínimo. De otra manera, P_M es un *PNC no-mínimo*.

Como ejemplos, las combinaciones de elementos de las Figuras **5.3** b), d), y e) definen PNCs jerárquicos mínimos y la combinación de elementos de la Figura **5.3** f) define un PNC secuencial mínimo. En cambio, los elementos de la Figura **5.3** a) no representan un PNC secuencial mínimo, ya que la secuencia tiene más de dos elementos.

Los elementos de la Figura **5.3** c) tampoco representan un PNC jerárquico mínimo, ya que el *Xor* está compuesto de más de dos caminos. Las Figuras **5.3** b), d), e), y f) representan PNCs mínimos, mientras que las Figuras **5.3** a), c), y g) representan PNCs no-mínimos.

5.2.2. Generación de un Perfil de No Solidez

Debido a que los lenguajes de PNCs tienen un conjunto finito de constructores, existe un número finito de formas de combinarlos, y por lo tanto, el conjunto de todos los posibles PNCs mínimos que pueden ser definidos a partir de un lenguaje de PNCs es también finito, a pesar que la combinación entre instancias de diferentes constructores es infinita, como se discutió anteriormente. Conociendo los PNCs mínimos de un lenguaje de PNCs, los mismos pueden ser verificados para determinar su solidez. Un *perfil de no solidez* de un lenguaje de PNCs contiene todos los posibles PNCs mínimos no sólidos que pueden ser definidos a partir de dicho lenguaje, es decir, todas las combinaciones de elementos que pueden determinar la no solidez de un PNC.

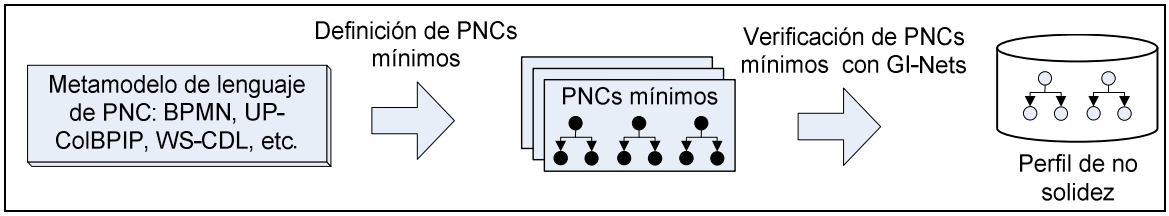


Fig. 5.5. Definición de un perfil de no solidez

La Figura **5.5** expresa los pasos para generar un perfil de no solidez. Dado un conjunto finito de constructores de un lenguaje de PNCs (como por ejemplo, BPMN o UP-ColBPIP), se generan todas las combinaciones de constructores de acuerdo al metamodelo de dicho lenguaje. En base a estas combinaciones, usando los conceptos de PNCs mínimos, se genera un conjunto finito de PNCs mínimos que están compuestos de instancias de estas combinaciones de constructores. Una vez generados los PNCs mínimos, estos son verificados utilizando cualquier método formal de verificación de PNCs, tal como el método basado en GI-Nets propuesto en el Capítulo 4. Más aún, cada PNC mínimo puede ser verificado independientemente de los demás, y por lo tanto se pueden utilizar diferentes

métodos de verificación. Aquellos PNCs cuyos resultados de verificación indican que son no sólidos, se agregan al perfil de no solidez.

5.2.3. Obtención de los PNCs Mínimos Determinantes

Un PNC mínimo de un perfil de no solidez puede tener elementos que causan no solidez, como así también elementos que no son causantes de la no solidez. Además, en el perfil puede haber distintos PNCs mínimos no sólidos que den origen a la definición de un mismo anti-patrón. Esto se debe a que varios PNC mínimos pueden tener el mismo conjunto de elementos que causan la no solidez. Por ejemplo, la Figura **5.3** b) muestra un PNC mínimo cuya raíz es un *And*, compuesto de un *Xor* y un *Termination*. Este PNC mínimo es no sólido ya que el *Termination* impide la sincronización del *And*. Sin embargo, si se elimina el elemento *Xor* o se lo reemplaza por cualquier otro elemento, el PNC mínimo continua siendo no sólido, ya que el elemento *Termination* sigue impidiendo la sincronización del *And*. En cambio, si se elimina el *Termination*, el PNC resultante se convierte en sólido, ya que la sincronización del *And* puede llevarse a cabo. Esto significa que los elementos determinantes de la no solidez del PNC mínimo son el *And* y el *Termination*.

De esta manera, los elementos *no determinantes* de la no solidez de un PNC son aquellos que pueden ser eliminados sin que cambie el resultado de la verificación. En cambio, los elementos *determinantes* de la no solidez son aquellos que al ser eliminados cambia el resultado de la verificación. Un PNC mínimo que contiene sólo elementos determinantes se lo denomina *PNC mínimo determinante.*

Definición 5.7. ***(PNC mínimo determinante de no solidez)*** Sea SP un modelo estructurado de un PNC compuesto de un conjunto de elementos E. Se dice que SP es un *PNC mínimo determinante de no solidez* (o *PNC mínimo determinante*) si: (1) SP es un PNC mínimo, y (2) la eliminación de cualquier elemento $e \in E$ causa que el SP se transforme en un PNC sólido o en un PNC estructurado no válido. Este último ocurre cuando deja de ser estructurado o no respeta las restricciones definidas en el metamodelo del lenguaje con el cual fue definido.

Para obtener los PNCs mínimos determinantes de no solidez de un lenguaje de PNCs a partir de su perfil de no solidez se propone el Algoritmo **5.1**. La función *isDeterminant(process)* recibe como entrada un PNC mínimo *process*. El algoritmo primero determina si *process* es un PNC mínimo no sólido mediante las funciones *isMinimalProcess(process)* e *isSound(process)*. Luego, itera sobre cada elemento del PNC mínimo. Si el elemento es un nodo hoja, lo elimina mediante la invocación de la función *removeElement*(*process*, *element)*. Esta función devuelve un nuevo PNC basado en el PNC pasado como parámetro, pero sin el elemento *element*. Si el PNC resultante de dicha eliminación no es sólido y es válido (satisface las restricciones del metamodelo), la función retorna *false*, indicando que el PNC *process* recibido como parámetro no es determinante. Si se recorren todos los elementos del PNC y finaliza el ciclo, la función retorna *true*, indicando que el PNC recibido como parámetro es determinante.

Algoritmo 5.1. Pseudocódigo para obtener PNCs mínimos determinantes.

```
Function isDeterminant (process)
if (isMinimalProcess(process) and ¬isSound(process))
  for all element such that element ∈ process do
    if (isLeaf(element)) then
      newProcess = removeElement(process, element)
      if (isValid(newProcess) and ¬isSound(newProcess)) then
        return false
      end if
    end if
  end
return true
end if
```

5.2.4. Definición de Anti-Patrones de Comportamiento de PNCs

A partir de un PNC mínimo determinante es posible definir un anti-patrón, ya que el mismo determina la combinación de constructores del anti-patrón de comportamiento.

Definición 5.8. ***(Anti-patrón de comportamiento de PNCs estructurados)*** Sea C un conjunto de constructores de un lenguaje, y SP un PNC mínimo determinante de no solidez que está compuesto de un conjunto de elementos E, que son instancias de algunos de los constructores de C. Un anti-patrón de comportamiento de PNCs estructurados para el conjunto de constructores C es una tupla $BAP = (SP, E_{AP}, R)$, donde:

— $E_{AP} \subseteq E_c \cup E_u$ representa al conjunto de todos los elementos del anti-patrón de comportamiento tal que cada elemento $e_{AP} \in E_{AP}$ representa a un elemento $e \in E$ y cada elemento $e \in E$ es representado por un elemento $e_{AP} \in E_{AP}$, donde:

- E_c es el conjunto de elementos del anti-patrón, tal que cada elemento $e \in E_c$ está asociado a un constructor $c \in C$,
- E_u es un conjunto de elementos del anti-patrón, tal que para cada elemento $e \in E_u$, no se requiere indicar con qué constructor $c \in C$ se asocia el elemento e,

— R es una regla que especifica las relaciones entre los elementos E_{AP} del anti-patrón.

La Definición **5.8** establece que un anti-patrón de comportamiento de modelos estructurados de PNCs se especifica a partir de un PNC mínimo determinante de no solidez. E_{AP} es el conjunto de elementos que define al anti-patrón y hay una relación biyectiva entre cada elemento de E_{AP} y de E. El conjunto de elementos E_{AP} puede estar compuesto de dos tipos de elementos representados por los subconjuntos E_c y E_u. E_c son los elementos determinantes del anti-patrón que se relacionan cada uno con un constructor específico, mientras que E_u representa a los elementos determinantes del anti-patrón para los cuales no se requiere conocer cuál es su constructor, de manera tal de representar a cualquier tipo de elemento de un modelo estructurado de PNC.

Un anti-patrón es utilizado para concluir sobre la no solidez de cualquier modelo estructurado de PNC, con lo cual el mismo debe ser definido de manera general sin considerar a ningún PNC específico. Esto se logra mediante los dos tipos de elementos que definen a un anti-patrón (E_c y E_u) y mediante la definición de una regla R que especifica cómo se relacionan dichos elementos para que se cumpla el anti-patrón. La regla debe indicar cómo sus elementos pueden estar relacionados (conectados), de manera directa o indirecta.

De esta manera, un anti-patrón de comportamiento puede ser visto como una regla que define un conjunto de restricciones estructurales en un lenguaje de PNCs para determinar especificaciones deficientes del comportamiento de modelos estructurados de PNCs. Dado un anti-patrón y un modelo estructurado de PNC, se dice que el anti-patrón se cumple sobre el modelo de PNC (o detecta un problema en un modelo de PNC) si tanto la combinación de elementos del modelo de PNC como las relaciones entre los mismos satisfacen la regla definida en el anti-patrón.

Tabla 5.1. Notación gráfica de anti-patrones de comportamiento de modelos estructurados de PNCs

Elemento	Notación Gráfica
Elemento de anti-patrón	
Elemento de anti-patrón no especificado	
Relación de sucesión directa entre elementos	
Relación de accesibilidad indirecta entre elementos	

La Tabla **5.1** muestra la notación gráfica utilizada para definir estos anti-patrones. Los elementos del conjunto E_c se definen con un círculo blanco con una etiqueta que indica el nombre del constructor, mientras que los elementos del conjunto E_u se definen con un círculo blanco sin etiqueta, ya que no están asociados a un constructor específico. Para indicar que dos elementos están conectados de manera directa se utiliza una flecha sólida, mientras que para una conexión indirecta se utiliza una flecha con línea de puntos.

La definición de la regla de un anti-patrón se puede realizar analizando cómo evoluciona el comportamiento de dicho PNC mínimo cuando se le adicionan nuevos elementos considerando las dimensiones de los PNCs.

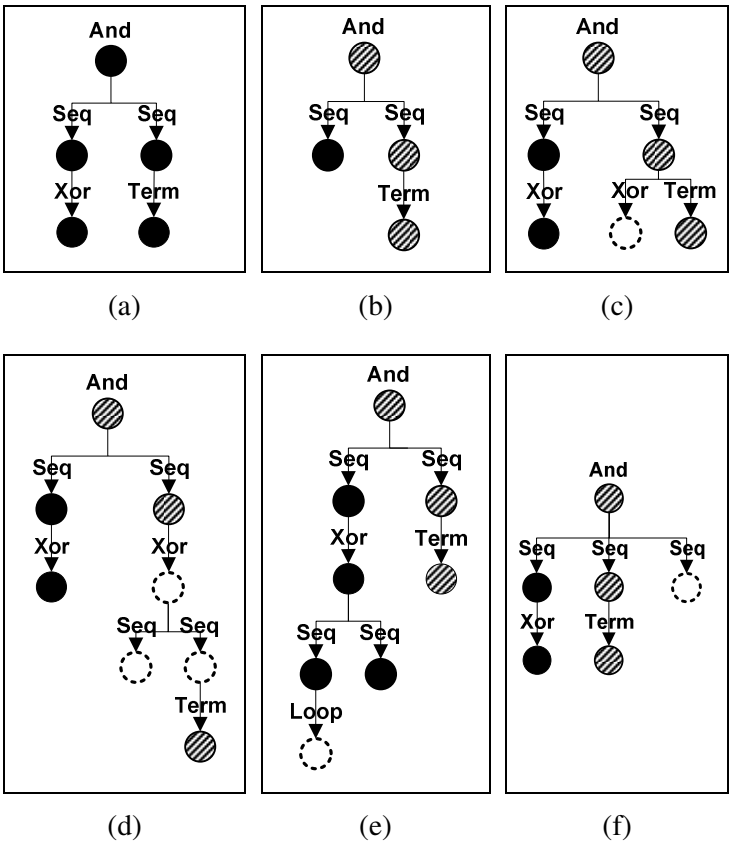


Fig. 5.6. Evolución de un PNC mínimo determinante de no solidez

La Figura **5.6** permite ejemplificar la especificación de una regla de un anti-patrón. La Figura **5.6** a) muestra un PNC mínimo no sólido, a partir del cual se generó un PNC mínimo determinante (Figura **5.6** b)), donde los elementos determinantes son el *And* y el *Termination*. El PNC mínimo determinante contiene elementos que están conectados de manera directa. Para definir la regla es necesario establecer qué sucede con el comportamiento del modelo de PNC si esta relación fuese indirecta, es decir, si los elementos determinantes no estuvieran directamente conectados. Para esto se estudia empíricamente cómo evoluciona el PNC mínimo determinante cuando se agregan nuevos elementos intermedios entre los elementos determinantes. Esto se muestra en la Figura **5.6** c), en la cual se agrega en la dimensión secuencia un *Xor* previo al *Termination*; en la Figura **5**.**6** d), en la cual se agrega en la dimensión jerarquía un *Xor* que anida al *Termination* ; en la Figura **5**.**6** e), en la cual se agrega también en la dimensión jerarquía un

Loop dentro del *Xor*; y en la Figura **5.6** f), en la cual se agrega en la dimensión instancia un nuevo camino paralelo al *And*. Los elementos agregados se muestran con línea de puntos.

En cualquiera de estos casos el modelo de PNC resultante sigue siendo no sólido, independientemente de los elementos intermedios que se agreguen y el nivel de anidamiento. Con lo cual el anti-patrón se va a cumplir si los elementos *And* y *Termination* están conectados de manera directa o indirecta, es decir, que el anti-patrón se cumple si existe en un modelo de PNC un elemento *And* a través del cual se puede alcanzar un *Termination* en la estructura de la vista de árbol del modelo de PNC.

Esto permite definir la regla R del anti-patrón de comportamiento para el caso que se encuentre la combinación de elementos *And* y *Termination*, la cual contempla estas relaciones entre los elementos (Figura **5.6**). Dicha regla se enuncia de la siguiente manera:

Dado un modelo estructurado de PNC SP compuesto del conjunto de elementos E, el anti-patrón se cumple si existe un par de elementos $e_1, e_2 \in E$ *donde* $e_1 = And$ *y* $e_2 = Termination$ *tal que* $e_1 \overset{*}{\rightarrow} e_2$*, es decir,* e_2 *puede ser alcanzado desde* e_1.

De la misma manera se debe analizar cada PNC mínimo determinante de un anti-patrón y definir su regla. Para definir gráficamente las reglas se utilizan las notaciones de modelos estructurados de PNCs: $\rightarrow$ indica una relación de sucesión entre elementos directamente conectados; $\overset{*}{\rightarrow}$ indica una relación de sucesión de elementos que no están directamente conectados; $Parent(e)$ indica al padre del elemento e en la estructura de árbol de un PNC; $Ancestors(e)$ indica al conjunto de elementos que se encuentran en el camino desde el elemento e hacia la raíz del árbol; *Successor(e)* retorna al sucesor del elemento *e* en la secuencia.

5.3. Especificación de Anti-Patrones de Comportamiento para UP-ColBPIP

Aplicando el enfoque descripto en la Sección 5.2, se especifican los anti-patrones de comportamiento para el lenguaje UP-ColBPIP. Primero se genera el perfil de no solidez y

los PNCs mínimos determinantes de UP-ColBPIP. Luego, en base a estos PNCs mínimos determinantes se especifican los anti-patrones de comportamiento de UP-ColBPIP con sus reglas.

5.3.1. Perfil de No Solidez de UP-ColBPIP

Para generar el perfil de no solidez se consideró el conjunto de constructores de UP-ColBPIP: *Interaction Path (Seq)*, *Xor*, *And*, *Or-SyncMerge (Or)*, *Loop-While (While)*, *Loop-Until (Until)*, *Cancel*, *Exception (Exc)*, *Multiple Instances (MI)* y *Termination (Term)*. En los PNCs mínimos obtenidos se asume que si contienen un *Interaction Path* o secuencia sin elementos hijos, se entiende que al menos un mensaje forma parte de dicha secuencia. El mensaje no se indica dado que no es un constructor de flujo de control. Los PNCs mínimos fueron generados manualmente en base a las combinaciones de constructores permitidas por UP-ColBPIP, las que dan lugar a la generación de 342 PNCs mínimos. La solidez de los mismos se determinó con el método de verificación basado en GI-Nets presentado en el Capítulo 4, Sección 4.1.

En total se obtuvieron 72 PNCs mínimos no sólidos que conforman el perfil de no solidez de UP-ColBPIP. La Tabla **5.2** muestra los resultados obtenidos. La columna *Raíz* indica al elemento raíz del PNC mínimo, y las columnas *E1* y *E2* indican los elementos anidados o hijos de la raíz. Cada fila define un PNC mínimo. Por ejemplo, el PNC mínimo N° 1 es un elemento *Sequence* que contiene un *Termination* y un *Xor*. El N° 25 es un elemento *And* que contiene un *Cancel* y un *While*.

Tabla 5.2. Perfil de no solidez del lenguaje UP-ColBPIP

N°	Raíz	E1	E2
1	*Seq*	*Term*	*Xor*
2	*Seq*	*Term*	*And*
3	*Seq*	*Term*	*Or*
4	*Seq*	*Term*	*While*
5	*Seq*	*Term*	*Until*
6	*Seq*	*Term*	*MI*
7	*Seq*	*Term*	*Cancel*
8	*Seq*	*Term*	*Exc*
9	*Seq*	*Term*	*Term*
10	*Xor*	*Term*	*Term*
11	*And*	*Term*	*Seq*
12	*And*	*Term*	*Xor*
13	*And*	*Term*	*And*
14	*And*	*Term*	*Or*
15	*And*	*Term*	*While*
16	*And*	*Term*	*Until*
17	*And*	*Term*	*MI*
18	*And*	*Term*	*Cancel*
19	*And*	*Term*	*Exc*
20	*And*	*Term*	*Term*
21	*And*	*Cancel*	*Seq*
22	*And*	*Cancel*	*Xor*
23	*And*	*Cancel*	*And*
24	*And*	*Cancel*	*Or*
25	*And*	*Cancel*	*While*
26	*And*	*Cancel*	*Until*
27	*And*	*Cancel*	*MI*
28	*And*	*Cancel*	*Cancel*
29	*And*	*Cancel*	*Exc*
30	*And*	*Cancel*	*Term*
31	*Or*	*Term*	*Seq*
32	*Or*	*Term*	*Xor*
33	*Or*	*Term*	*And*
34	*Or*	*Term*	*Or*
35	*Or*	*Term*	*While*
36	*Or*	*Term*	*Until*
37	*Or*	*Term*	*MI*
38	*Or*	*Term*	*Cancel*
39	*Or*	*Term*	*Exc*
40	*Or*	*Term*	*Term*
41	*Or*	*Cancel*	*Seq*
42	*Or*	*Cancel*	*Xor*
43	*Or*	*Cancel*	*And*
44	*Or*	*Cancel*	*Or*
45	*Or*	*Cancel*	*While*
46	*Or*	*Cancel*	*Until*
47	*Or*	*Cancel*	*MI*
48	*Or*	*Cancel*	*Cancel*
49	*Or*	*Cancel*	*Exc*
50	*Or*	*Cancel*	*Term*
51	*While*	*Term*	
52	*Until*	*Term*	
53	*MI*	*Term*	
54	*MI*	*Cancel*	
55	*Cancel*	*Seq*	*Term*
56	*Cancel*	*Xor*	*Term*
57	*Cancel*	*And*	*Term*
58	*Cancel*	*Or*	*Term*
59	*Cancel*	*While*	*Term*
60	*Cancel*	*Until*	*Term*
61	*Cancel*	*MI*	*Term*
62	*Cancel*	*Cancel*	*Term*
63	*Cancel*	*Exc*	*Term*
64	*Exc*	*Seq*	*Term*
65	*Exc*	*Xor*	*Term*
66	*Exc*	*And*	*Term*
67	*Exc*	*Or*	*Term*
68	*Exc*	*While*	*Term*
69	*Exc*	*Until*	*Term*
70	*Exc*	*MI*	*Term*
71	*Exc*	*Cancel*	*Term*
72	*Exc*	*Exc*	*Term*

Se aplicó la función *isDeterminant()* del Algoritmo **5.1** a cada PNC mínimo de este perfil de no solidez para obtener los PNCs mínimos determinantes (Tabla **5.3**). Por ejemplo, dicha función devuelve verdadero al evaluarla con el PNC mínimo número 11 de la Tabla **5.2**, con lo cual dicho PNC mínimo es determinante. La aplicación del Algoritmo **5.1** a los PNCs mínimos 12 a 20 determina que los mismos son no determinantes. Esto se debe a que

si se elimina el segundo elemento de cada uno de estos PNCs mínimos (del 12 al 20) dichos PNCs seguen siendo no sólidos. De manera similar se obtuvieron los restantes PNCs mínimos determinantes de la Tabla **5.3**.

Tabla 5.3. PNCs mínimos determinantes para el lenguaje UP-ColBPIP

N°	Raíz	E1	E2
1	*Seq*	*Term*	*Xor*
2	*Seq*	*Term*	*And*
3	*Seq*	*Term*	*Or*
4	*Seq*	*Term*	*While*
5	*Seq*	*Term*	*Until*
6	*Seq*	*Term*	*MI*
7	*Seq*	*Term*	*Cancel*

N°	Raíz	E1	E2
8	*Seq*	*Term*	*Exc*
9	*Seq*	*Term*	*Term*
10	*Xor*	*Term*	*Term*
11	*And*	*Term*	*Seq*
12	*And*	*Cancel*	*Seq*
13	*Or*	*Term*	*Seq*
14	*Or*	*Cancel*	*Seq*

N°	Raíz	E1	E2
15	*While*	*Term*	
16	*Until*	*Term*	
17	*MI*	*Term*	
18	*MI*	*Cancel*	
19	*Cancel*	*Seq*	*Term*
20	*Exc*	*Seq*	*Term*

En base al conjunto de PNCs mínimos determinantes de UP-ColBPIP se especificaron doce anti-patrones los cuales se muestran en la Figura **5.7**. Los algoritmos que implementan estos anti-patrones se encuentran en el Anexo B. En las siguientes secciones se describen estos anti-patrones agrupados en cinco categorías distintas.

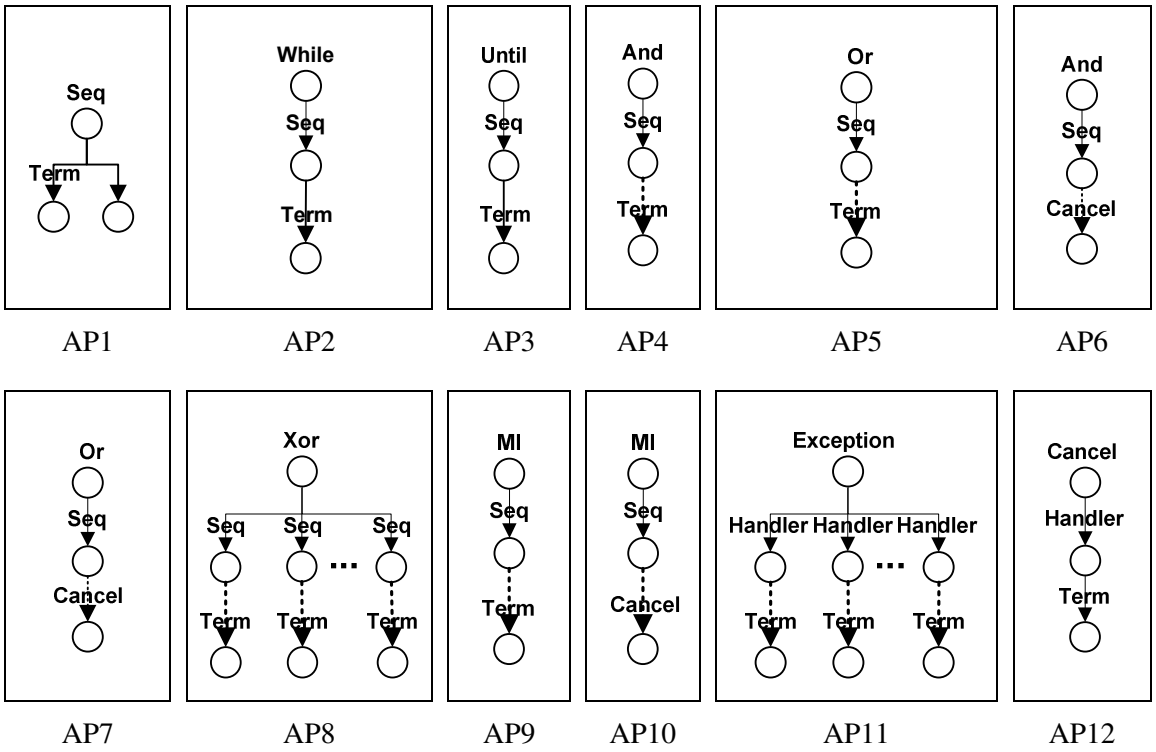


Fig. 5.7. Anti-patrones de comportamiento de UP-ColBPIP

5.3.2. Bloqueo en Secuencias

Anti-Patrón AP1 - Elementos en Secuencia No Accesibles

Descripción

De los elementos 1 a 9 de la Tabla **5.3**, se puede observar que para todo PNC mínimo *P* cuya raíz sea *Sequence* y que esté compuesto de un elemento *Termination* sucedido por cualquier elemento, se cumple que *P* no es sólido. Esto da lugar al anti-patrón de comportamiento *AP1 - Elementos en Secuencia No Accesibles* que se muestra en la Figura **5.8**. El *AP1* se cumple si se combinan los elementos *Sequence*, con un elemento *Termination*, el cual está seguido de cualquier otro tipo de elemento en la secuencia. La regla que define al anti-patrón *AP1* es la siguiente:

Dado un modelo estructurado de PNC SP compuesto del conjunto de elementos E, AP1 se cumple si existe un par de elementos $e_1, e_2 \in E$ *donde* $e_1 = Termination$, $Parent(e_1) = Sequence$ y $Parent(e_2) = Sequence$ *tal que* e_1 *precede en la secuencia a* e_2.

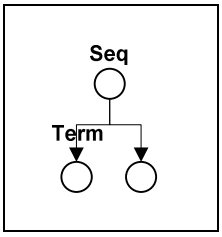
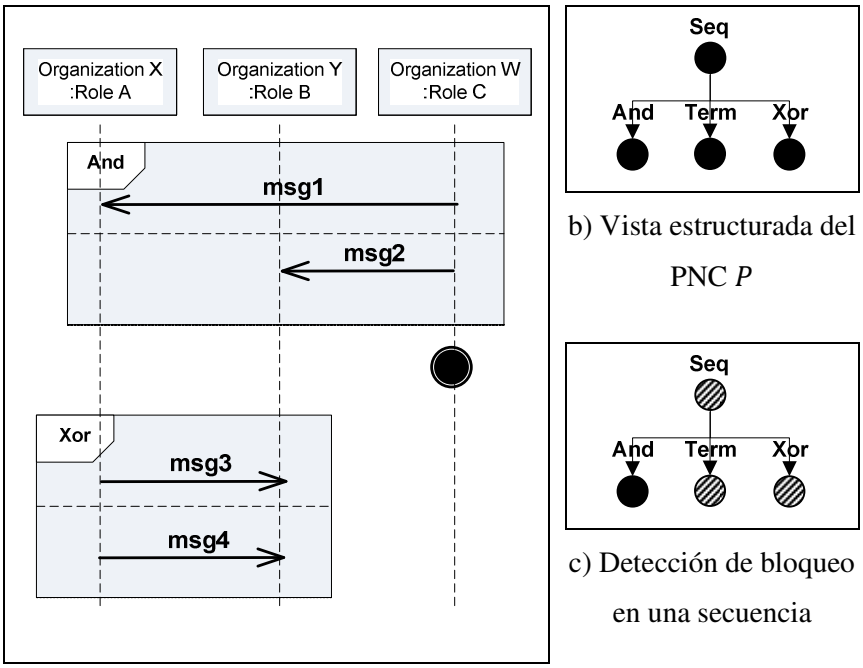


Fig. 5.8. Anti-patrón AP1

Ejemplo del Anti-Patrón AP1 en UP-ColBPIP

La Figura **5.9** a) muestra un protocolo de interacción definido con UP-ColBPIP que modela un PNC en el cual participan las organizaciones X, Y, y W a través de los roles A, B, y C respectivamente. El PNC inicia cuando C le envía simultáneamente los mensajes *msg1* y *msg2* a A y B. Luego de enviar los mensajes, C finaliza su participación en el proceso, ya que no interviene en otros mensajes. Finalmente, A envía a B de manera alternativa el mensaje *msg3* o el mensaje *msg4*, y el protocolo finaliza.



a) PNC *P* definido con UP-ColBPIP

b) Vista estructurada del PNC *P*

c) Detección de bloqueo en una secuencia

Fig. 5.9. PNC que ejemplifica el anti-patrón AP1

La Figura **5.9** b) muestra la vista estructurada de este PNC, el cual es una secuencia con los elementos *And*, *Term*, y *Xor*, mientras que la Figura **5.9** c) muestra la detección del anti-patrón *AP1* en dicho modelo de PNC, donde los nodos con textura de líneas oblicuas determinan los elementos que forman parte del anti-patrón. El anti-patrón *AP1* se cumple en este modelo de PNC debido a que luego del elemento *Termination* la secuencia sigue con otro elemento, en este caso un *Xor*. El problema que ocurre en este ejemplo es que se asume que la terminación explícita finaliza la ejecución del rol C, sin embargo, esta terminación explícita finaliza el PNC, con lo cual A no va a poder enviar los mensajes *msg3* y *msg4* a la organización B. Desde el punto de vista lógico, el PNC no debería finalizar luego que C envía los mensajes.

Solución para el Anti-Patrón AP1

Existen dos posibles soluciones para eliminar el anti-patrón *AP1*: eliminar el elemento *Termination* que causa el bloqueo, o cambiar el orden de los elementos y pasar el *Termination* al final de la secuencia. Ambas opciones son válidas para el ejemplo mostrado.

La definición de un elemento *Termination* precediendo cualquier otro elemento parece una situación que se puede detectar trivialmente. Sin embargo, puede ocurrir que la semántica de dicho elemento no esté bien comprendida por el diseñador y llevar a situaciones como las planteadas en el ejemplo. Lenguajes estructurados como UP-ColBPIP permiten estas situaciones, ya que los elementos que forman parte de un camino están conectados de manera secuencial. Esto permite que un *Termination* pueda ser definido previo a cualquier otro elemento. En otros lenguajes como BPMN esto no es posible, debido a que un elemento *evento de terminación* sólo puede tener flujos de secuencia de entrada, pero no puede tener flujos de secuencia de salida.

5.3.3. Bloqueo en Ciclos

En UP-ColBPIP hay dos tipos de ciclos: *Loop-While* y *Loop-Until*. Ambos están compuestos por una secuencia de elementos. La semántica de comportamiento determina que la ejecución de esa secuencia de elementos se puede repetir un número determinado de veces, cero o más veces para el *Loop-While* y una o más veces para el *Loop-Until*.

De la Tabla **5.3** se puede observar que para todo PNC mínimo *P* cuya raíz sea *Loop-While* o *Loop-Until* y esté compuesto de un elemento *Termination*, se cumple que *P* no es sólido. Esto se debe a que una vez que se alcanza el *Termination* la condición para determinar si finaliza o continúa la ejecución del ciclo ya no se puede alcanzar, ocasionando que el ciclo se ejecute a lo sumo una vez, lo cual no es el comportamiento esperado para un ciclo. A partir de esto se definen los siguientes anti-patrones.

Anti-Patrones AP2 - Bloqueo en ciclo Loop-While y AP3 - Bloqueo en ciclo Loop-Until

Descripción

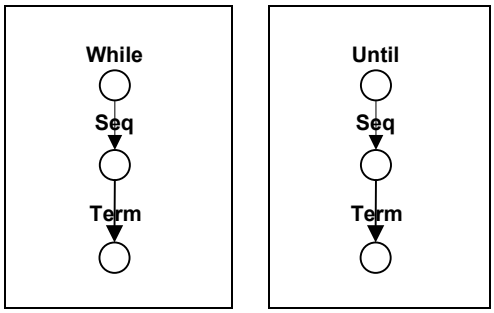
Los anti-patrones *AP2* y *AP3* se obtienen a partir de los PNCs mínimos determinantes N° 15 y 16 de la Tabla **5.3**. El anti-patrón *AP2 - Bloqueo en ciclo Loop-While* se muestra en la Figura **5.10** a) que expresa que *AP2* se cumple si se combina un elemento *Loop-While* con un elemento *Termination*. No importan los demás elementos que

estén contenidos dentro del *Loop-While*, con lo cual podría haber otros elementos formando parte del mismo. La regla que define al anti-patrón *AP2* es la siguiente:

Dado un modelo estructurado de PNC SP compuesto del conjunto de elementos E, AP2 se cumple si existen tres elementos $e_1, e_2, e_3 \in E$ *donde* $e_1 = Termination$, $e_2 = Sequence$, $Parent(e_1) = e_2$, *y* $Parent(e_2) = While$.

El anti-patrón *AP3 - Bloqueo en ciclo Loop-Until* (Figura **5.10** b)) es análogo al anti-patrón *AP2*, pero difiere en que la raíz del mismo es un *Loop-Until* en lugar de un *Loop-While*. La regla que define al anti-patrón *AP3* es la siguiente:

Dado un modelo estructurado de PNC SP compuesto del conjunto de elementos E, AP3 se cumple si existen tres elementos $e_1, e_2, e_3 \in E$ *donde* $e_1 = Termination$, $e_2 = Sequence$, $Parent(e_1) = e_2$, *y* $Parent(e_2) = Until$.

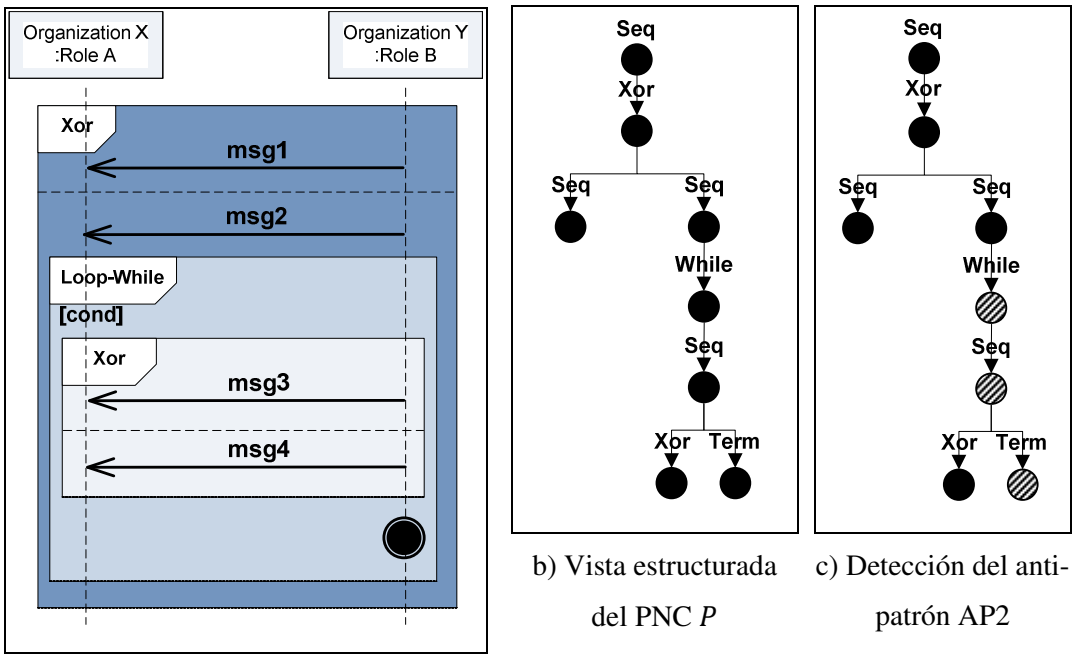


a) AP2 b) AP3

Fig. 5.10. Anti-patrones AP2 y AP3

Ejemplo

La Figura **5.11** a) muestra el protocolo de interacción definido con UP-ColBPIP, el cual inicia cuando B envía el mensaje *msg1* o el mensaje *msg2* a A. En caso de enviarse el mensaje *msg2*, el proceso continúa con un ciclo de tipo *Loop-While* que se ejecuta mientras que se cumpla la condición *cond*. En este ciclo B puede enviar de manera alternativa los mensajes *msg4* y *msg5* y luego el protocolo finaliza su ejecución de manera explícita.



a) PNC *P* definido con UP-ColBPIP

b) Vista estructurada del PNC *P*

c) Detección del anti-patrón AP2

Fig. 5.11. PNC que ejemplifica el anti-patrón AP2

Las Figuras **5.11** b) y c) muestran la vista estructurada de este PNC y la detección del anti-patrón *AP2* en el mismo. El problema que ocurre en este ejemplo es que cuando el protocolo alcanza el elemento *Termination* en la primera iteración del ciclo *Loop-While*, el ciclo no va a poder continuar con su ejecución, con lo cual ocurre un bloqueo. Si se tuviera un ciclo *Loop-Until* ocurriría exactamente el mismo bloqueo.

Solución a los Anti-Patrones AP2 y AP3

Las soluciones para estos anti-patrones: eliminar el elemento *Termination* del ciclo, o agregar dos caminos alternativos tal que uno incluya al elemento *Termination* y el otro permita continuar con la ejecución normal del ciclo.

5.3.4. Bloqueo en Caminos Paralelos

UP-ColBPIP provee dos constructores para representar paralelismo: *And y Or*. Ambos están compuestos por secuencias de elementos que se ejecutan en paralelo, aunque la semántica de comportamiento de bifurcación y sincronización de cada uno varía.

De la Tabla **5.3** se puede observar que todo PNC mínimo *P* cuya raíz sea *And* u *Or* y que esté compuesto de un elemento *Termination* o de un elemento *Cancel*, se cumple que *P* no es sólido. Se puede inferir entonces que todo modelo de PNC que tenga un elemento que representa paralelismo y que desde dicho elemento sea posible alcanzar un elemento *Termination* o *Cancel* en uno de los caminos en paralelo, la sincronización de los caminos en paralelo no se podrá realizar, ocasionando un bloqueo. A partir de este análisis se definen cuatro anti-patrones de comportamiento que causan bloqueos en caminos paralelos.

Anti-Patrones AP4 - Bloqueo And-Termination y AP5 - Bloqueo Or-Termination

Descripción

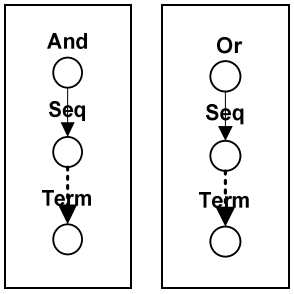
Los anti-patrones *AP4* y *AP5* se especifican a partir de los PNCs mínimos determinantes N° 11 y 13 de la Tabla **5.3**.

El anti-patrón *AP4 - Bloqueo And-Termination* (Figura **5.12** a)) se cumple si existe un *And* que tiene un elemento *Sequence* que representa un camino del *And*, a través del cual es posible acceder (directa o indirectamente) a un elemento *Termination*. La relación entre el elemento *Sequence* y el *Termination* se muestra con una flecha con línea de puntos, lo que indica que el *Termination* puede ser accesible de manera directa o indirecta. Los demás caminos o secuencias que forman parte del *And* no son determinantes para detectar el anti-patrón. La regla que define al anti-patrón *AP4* es la siguiente:

Dado un modelo estructurado de PNC SP compuesto del conjunto de elementos E, AP4 se cumple si existen tres elementos $e_1, e_2, e_3 \in E$ *donde* $e_1 = Termination$, $e_2 = Sequence$, $e_2 \in Ancestors(e_1)$ *y* $Parent(e_2) = And$.

El anti-patrón *AP5 - Bloqueo Or-Termination* (Figura **5.12** b)) se cumple si existe *Or* con sincronización *Synchronizing Merge* y alguno de los posibles caminos paralelos del *Or* alcanza un elemento *Termination* directa o indirectamente. La descripción de este anti-patrón es análoga a la del anti-patrón *AP4*, pero difiere en que la raíz del mismo es un *Or* en lugar de un *And*. La regla que define al anti-patrón *AP5* es la siguiente:

Dado un modelo estructurado de PNC SP compuesto del conjunto de elementos E, AP5 se cumple si existen tres elementos $e_1, e_2, e_3 \in E$ *donde* $e_1 = Termination$, $e_2 = Sequence$, $e_2 \in Ancestors(e_1)$ *y* $Parent(e_2) = Or$.

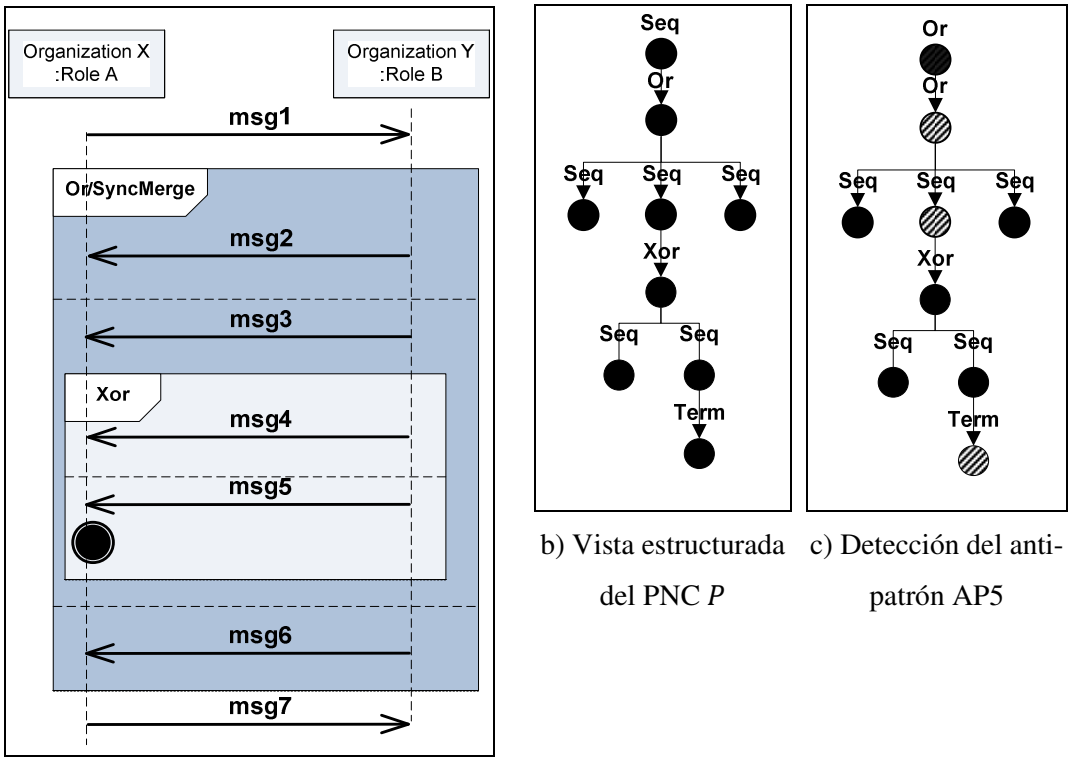


a) AP4 b) AP5

Fig. 5.12. Anti-patrones AP4 y AP5

Ejemplo

La Figura **5.13** a) muestra un protocolo de interacción definido con UP-ColBPIP, el cual inicia cuando A envía el mensaje *msg1* a B. Luego, pueden ocurrir los siguientes caminos de interacciones alternativos: (1) B le envía el mensaje *msg2* a A; (2) B le envía el mensaje *msg3* a A, y luego puede optar por enviar el mensaje *msg4* o el mensaje *msg5* y finaliza la ejecución; (3) B le envía el mensaje *msg6* a A. Finalmente, A le envía el mensaje *msg7* a B, y el protocolo finaliza.



a) PNC *P* definido con UP-ColBPIP

b) Vista estructurada del PNC *P*

c) Detección del anti-patrón AP5

Fig. 5.13. PNC que ejemplifica el anti-patrón AP5

Las Figuras **5.13** b) y c) muestran la vista estructurada del PNC y la detección del anti-patrón *AP5* respectivamente. El problema que ocurre en este ejemplo es que si el segundo camino de interacción del constructor *Or* se ejecuta, se puede llegar a un punto en el cual el elemento *Termination* que está definido dentro del *Xor* impida que se lleve a cabo la sincronización de los caminos del *Or* que se ejecutan en paralelo. En este caso, el *Termination* puede causar un bloqueo en la sincronización del *Or* y es posible que el mensaje *msg7* no se ejecute debido a que no se puede llevar a cabo dicha sincronización.

Anti-Patrones AP6 - Bloqueo And-Cancel y AP7 - Bloqueo Or/SyncMerge-Cancel

Descripción

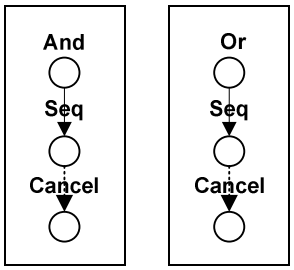
Para especificar los anti-patrones *AP6* y *AP7* se utilizaron los PNCs mínimos determinantes N° 12 y 14 de la Tabla **5.3**.

El anti-patrón *AP6 - Bloqueo And-Cancel* (Figura **5.14** a)) indica que se cumple si existe un *And* con una secuencia (o camino) en la cual es posible alcanzar directa o indirectamente un elemento *Cancel*. Los demás caminos que forman parte del *And* no son determinantes para el anti-patrón. La regla que define al anti-patrón *AP6* es la siguiente:

Dado un modelo estructurado de PNC SP compuesto del conjunto de elementos E, AP6 se cumple si existen tres elementos $e_1, e_2, e_3 \in E$ *donde* $e_1 = Cancel$, $e_2 = Sequence$, $e_2 \in Ancestors(e_1)$ *y* $Parent(e_2) = And$.

El anti-patrón *AP7 - Bloqueo Or-Cancel* (Figura **5.14** b)) se cumple si existe un *Or* con sincronización *Synchronizing Merge* y alguno de los posibles caminos paralelos del *Or* alcanza un elemento *Cancel*. La regla que define al anti-patrón *AP7* es la siguiente:

Dado un modelo estructurado de PNC SP compuesto del conjunto de elementos E, AP7 se cumple si existen tres elementos $e_1, e_2, e_3 \in E$ *donde* $e_1 = Cancel$, $e_2 = Sequence$, $e_2 \in Ancestor(e_1)$ *y* $Parent(e_2) = Or$.



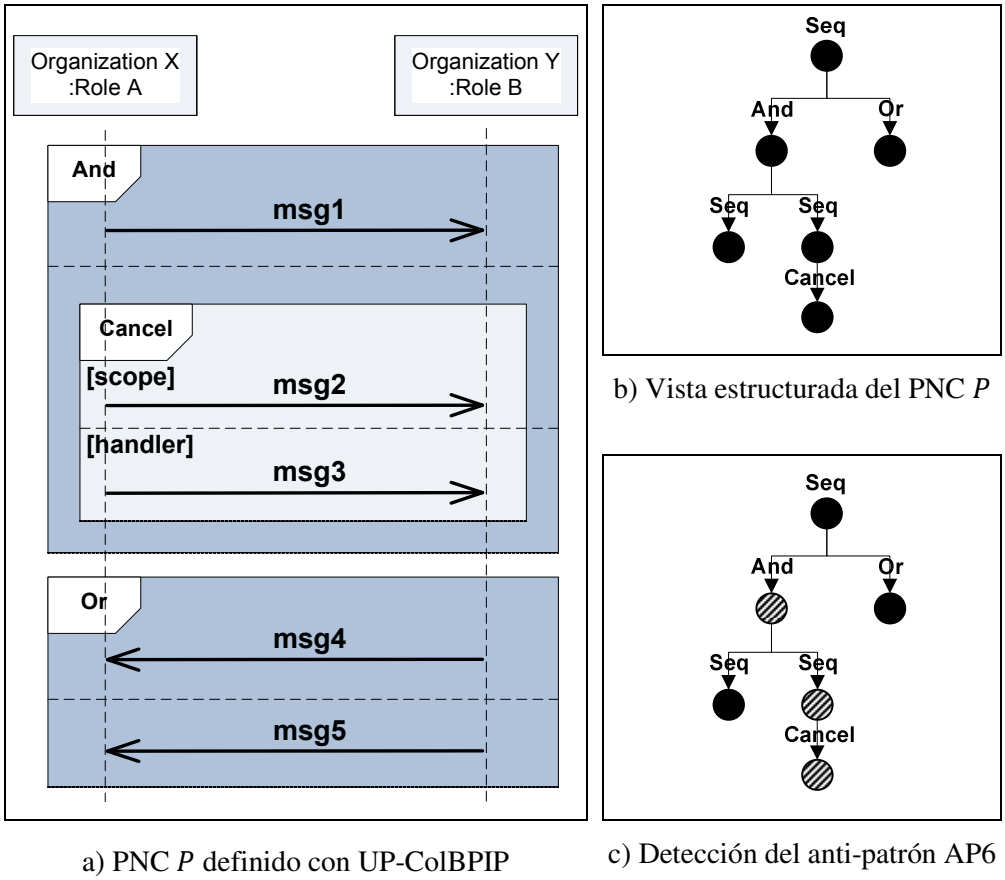
a) AP6 b) AP7

Fig. 5.14. Anti-patrones AP6 y AP7

Ejemplo

La Figura **5.15** a) muestra un protocolo de interacción definido con UP-ColBPIP. El PNC inicia cuando A envía simultáneamente a B los mensajes *msg1* y *msg2*. Si este último mensaje no llega a tiempo a destino ocurre una excepción y se cancela la ejecución del

PNC luego que A le envíe el mensaje *msg3* a B. En caso que el PNC no sea cancelado, B le envía de manera alternativa a A los mensajes *msg4* y *msg5* y el protocolo finaliza.



a) PNC *P* definido con UP-ColBPIP

b) Vista estructurada del PNC *P*

c) Detección del anti-patrón AP6

Fig. 5.15. PNC que ejemplifica el anti-patrón AP6

Las Figuras **5.15** b) y c) muestran la vista estructurada del PNC y la detección del anti-patrón *AP6* respectivamente. El problema en este modelo de PNC es que cuando se ejecuta el segundo camino de interacción del *And*, se puede ejecutar el *Cancel* que está definido dentro del *And* e impida que se lleve a cabo la sincronización de los caminos que se ejecutan en paralelo. En este caso, los mensajes *msg4* y *msg5* no se ejecutarán.

Solución a los Anti-Patrones AP4 a AP7

Existen dos posibles soluciones a estos anti-patrones: eliminar el elemento *Termination* o *Cancel*; o en el caso de un *Cancel*, definir al mismo tal que su *Scope* incluya

al elemento *And* u *Or*, y por ende a todos sus elementos. De esta manera, si se ejecuta una excepción se cancelan todos los caminos paralelos simultáneamente.

5.3.5. Bloqueo en Caminos Mutuamente Excluyentes

El constructor *Xor* de UP-ColBPIP representa la exclusión mutua de dos o más caminos, cada uno representado por una secuencia de elementos, donde dada una condición de entrada sólo un camino (una de las secuencias) se ejecutará.

De la Tabla **5.3** se puede observar que todo PNC mínimo *P* cuya raíz sea *Xor* y donde todas las secuencias que los componen tengan un elemento *Termination*, se cumple que *P* no es sólido. Esto se debe a que la semántica del Xor establece que al menos una de las secuencias debe ejecutarse. Si todos los posibles caminos alcanzan un *Termination*, entonces se produce un bloqueo y todos los elementos que vienen a continuación del *Xor* nunca puedan ejecutarse. A partir de este análisis se define el siguiente anti-patrón.

Anti-Patrón AP8 - Bloqueo Xor-Termination

Descripción

El anti-patrón *AP8* se especificó a partir del PNC mínimo determinante N° 10 de la Tabla **5.3**. El anti-patrón *AP8 - Bloqueo Xor-Termination* (Figura **5.16**) indica que se cumple si existe un *Xor* en donde en cada una de sus secuencias o caminos es posible alcanzar un elemento *Termination*. La regla que define al anti-patrón *AP8* es:

Dado un modelo estructurado de PNC SP compuesto del conjunto de elementos E, AP8 se cumple si existe un elemento $e_1 \in E$*, donde* $e_1 = Xor$*, tal que para cada elemento* s_i*, donde* $Parent(s_i) = e_1$ *y* $s_i = Sequence$*, existe un elemento* $e_i = Termination$ *tal que* $s_i \in Ancestors(e_i)$

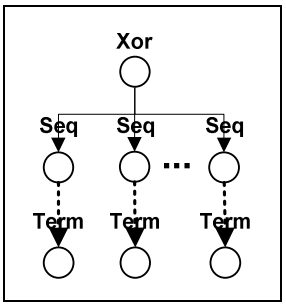
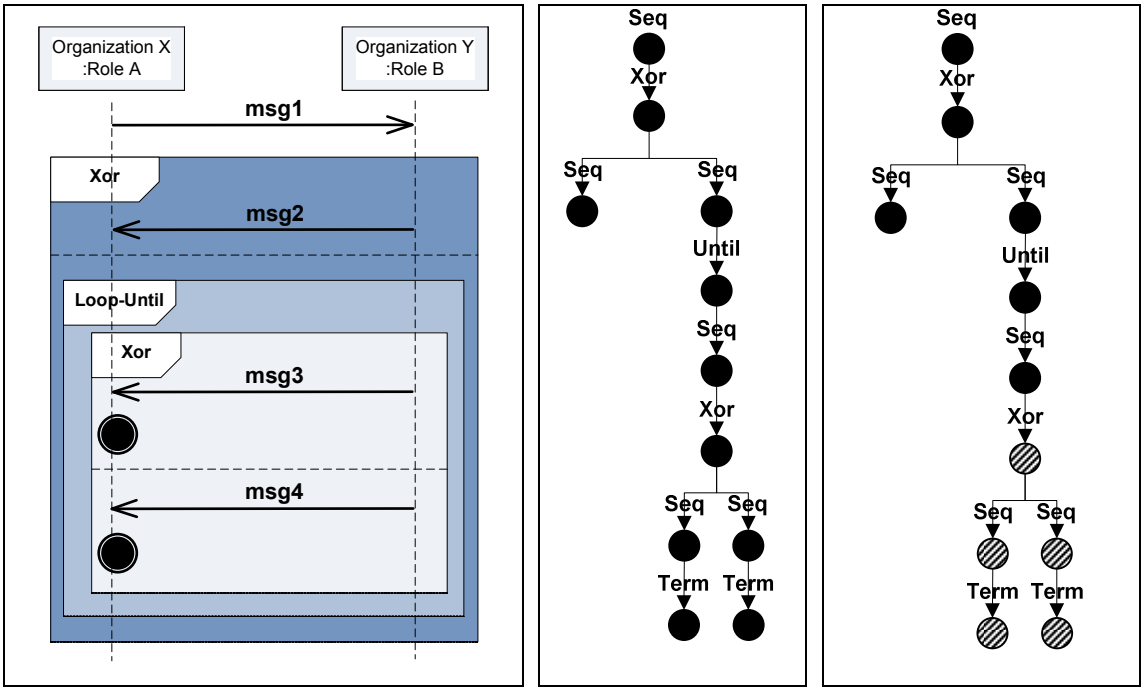


Fig. 5.16. Anti-patrón AP8

Ejemplo

La Figura **5.17** a) muestra un protocolo de interacción definido con UP-ColBPIP, el cual inicia cuando A le envía a B el mensaje *msg1*. Luego, B puede enviar el mensaje *msg2* a A o iniciar un ciclo en el cual puede optar por enviar el mensaje *msg3* y finalizar la ejecución o enviar el mensaje *msg4* y finalizar la ejecución. Luego del ciclo el protocolo finaliza.



a) PNC *P* definido con UP-ColBPIP

b) Vista estructurada del PNC *P*

c) Detección del anti-patrón AP8

Fig. 5.17. PNC que ejemplifica el anti-patrón AP8

La Figura **5.17** b) muestra la vista estructurada del PNC. La Figura **5.17** c) muestra la detección del anti-patrón *AP8*. El problema que ocurre en este ejemplo es que si se ejecuta el *Xor* anidado en el *Loop-Until*, se llega a un punto en el cual ya no se puede avanzar, y se impide que se lleve a cabo la unión de los caminos del *Xor*. En este caso, los elementos *Termination* ocasionan un bloqueo dentro del *Xor* y el *Loop-Until* puede ejecutarse a lo sumo una vez, lo cual no es el comportamiento esperado para un ciclo.

Solución

Existen dos posibles soluciones para eliminar este anti-patrón: eliminar un elemento *Termination* o agregar un nuevo camino que no contenga un elemento *Termination* al *Xor*. En el modelo de PNC de la Figura **5.17** lo correcto es agregar un nuevo camino de interacción que permita continuar con el PNC.

Un caso especial de este anti-patrón ocurre si el elemento *Xor* con un *Termination* en todos sus caminos es el último elemento de un modelo de PNC, es decir, luego del *Xor* el PNC finaliza su ejecución. En este caso se podría interpretar que el PNC no tiene un bloqueo ya que no existe ningún elemento a ejecutar luego del *Xor*. Sin embargo, en un modelo estructurado de PNC, para un *Xor* se debe representar la unión de los caminos (*Xor Join*). Si el *Xor* contiene elementos *Termination* en todos sus caminos, el *Xor Join* nunca se va a alcanzar. La solución en este caso particular es eliminar los *Termination* de los caminos del *Xor* y agregar un único *Termination* a continuación del *Xor Join*.

5.3.6. Bloqueo en Caminos con Instancias Múltiples

El constructor *Multiple Instances* de UP-ColBPIP permite representar instancias múltiples de mensajes (interacciones). Este constructor está compuesto por una secuencia de elementos, donde puede haber múltiples instancias de ejecución en paralelo de dicha secuencia. Su ejecución finaliza cuando todas las instancias finalizaron su ejecución.

De la Tabla **5.3** se puede observar que todo PNC mínimo *P* cuya raíz sea *Multiple Instances* y que esté compuesto de un elemento *Termination* o de un elemento *Cancel*, se cumple que *P* no es sólido. Esto se debe a que una vez que se alcanza un *Termination* o *Cancel* la instancia que contiene a dichos constructores ya no podrá ser sincronizada,

ocasionando que las demás instancias en paralelo se bloqueen esperando la sincronización. A partir de esto se definen los siguientes anti-patrones.

Anti-Patrón AP9 - Bloqueo Multiple Instances-Termination

Descripción

El anti-patrón *AP9* se especifica a partir del PNC mínimo determinante N° 17 de la Tabla **5.3**. El anti-patrón *AP9 - Bloqueo Multiple Instances-Termination* (Figura **5.18**) se cumple si existe un *Multiple Instances (MI)* que contiene una secuencia en la cual es posible alcanzar un elemento *Termination*. La regla que lo define es la siguiente:

Dado un modelo estructurado de PNC SP compuesto del conjunto de elementos E, AP9 se cumple si existen tres elementos $e_1, e_2, e_3 \in E$ *donde* $e_1 = Termination$, $e_2 = Sequence$, $e_2 \in Ancestors(e_1)$ *y* $Parent(e_2) = MI$.

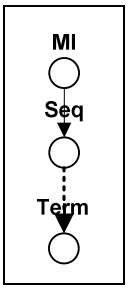
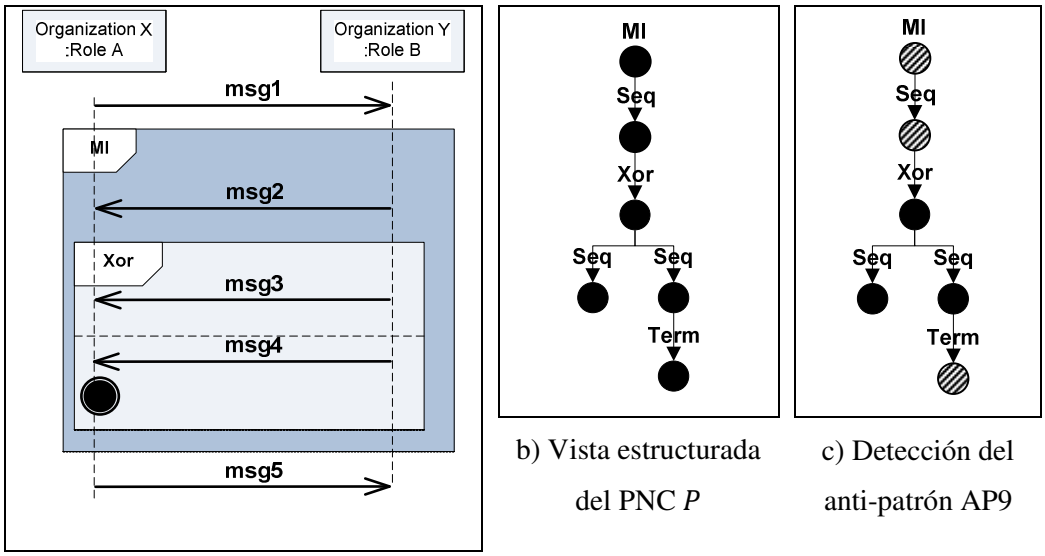


Fig. 5.18. Anti-patrón AP9

Ejemplo

La Figura **5.19** a) muestra un protocolo de interacción UP-ColBPIP. El PNC inicia cuando A le envía a B el mensaje *msg1*. Luego, se ejecutan instancias múltiples del mensaje *msg2* enviado de B a A, seguido por el envío del mensaje *msg3* o por el envío de *msg4* y una terminación explícita. Finalmente, A le envía a B el mensaje *msg5*.



a) PNC *P* definido con UP-ColBPIP

b) Vista estructurada del PNC *P*

c) Detección del anti-patrón AP9

Fig. 5.19. PNC que ejemplifica el anti-patrón AP9

Las Figuras **5.19** b) y c) muestran la vista estructurada del PNC y la detección del anti-patrón *AP9* respectivamente. El problema de este ejemplo es que si el segundo camino de interacción del constructor *Xor* se ejecuta, se puede llegar a un punto en el cual el elemento *Termination* definido dentro del *Xor* impida que se realice la sincronización de las instancias múltiples en paralelo. En este caso, el *Termination* puede causar un bloqueo en la sincronización del *Multiple Instances* y es posible que el mensaje *msg5* no se ejecute debido a que no se puede llevar a cabo dicha sincronización.

Anti-Patrón AP10 - Bloqueo Multiple Instances-Cancel

Descripción

El anti-patrón *AP10* se especificó a partir del PNC mínimo determinante N° 18 de la Tabla **5.3**. El anti-patrón *AP10 - Bloqueo Multiple Instances-Cancel* (Figura **5.20**) se cumple si existe un *Multiple Instances* que contiene una secuencia en la cual es posible alcanzar un elemento *Cancel*. La regla que lo define es la siguiente:

Dado un modelo estructurado de PNC SP compuesto del conjunto de elementos E, AP10 se cumple si existen tres elementos $e_1, e_2, e_3 \in E$ *donde* $e_1 = Cancel$, $e_2 = Sequence$, $Ancestor(e_1) = e_2$ *y* $Parent(e_2) = MI$.

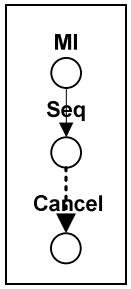
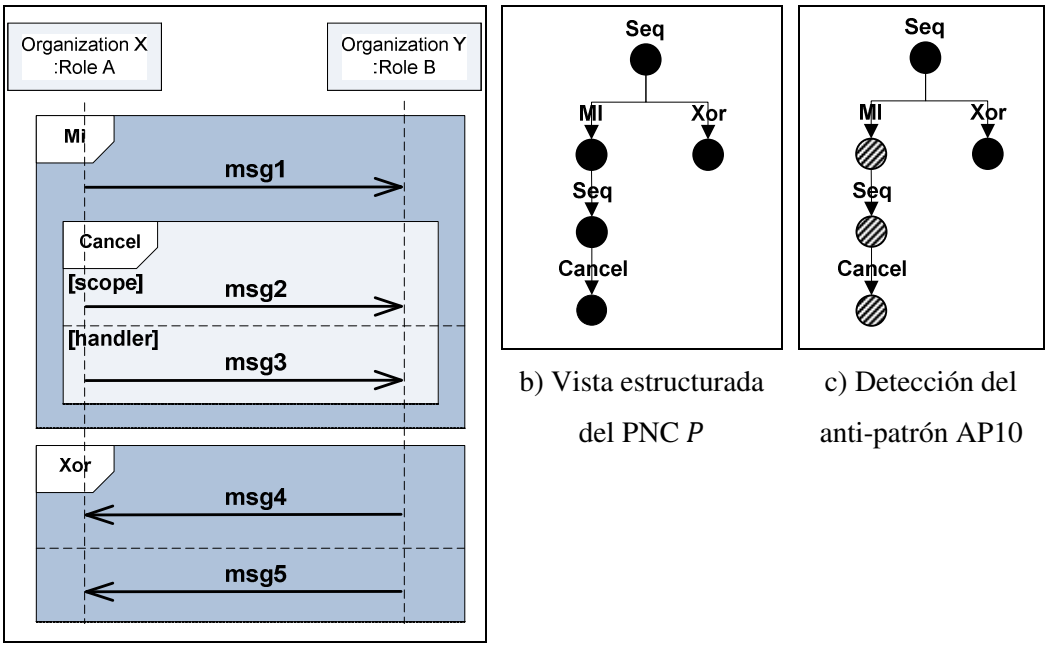


Fig. 5.20. Anti-patrón AP10

Ejemplo

La Figura **5.21** a) muestra un protocolo de interacción definido con UP-ColBPIP, el cual inicia cuando A envía a B instancias múltiples de los mensajes *msg1* y *msg2*. Si *msg2* no llega a tiempo a destino ocurre una excepción y se cancela la ejecución de todos los elementos dentro del *Scope*, y luego que A le envía a B el mensaje *msg3* que es parte del manejo de la excepción. Si no se ejecuta la excepción, B le envía a A de manera alternativa el mensaje *msg3* o el mensaje *msg4* y el protocolo finaliza.



a) PNC *P* definido con UP-ColBPIP

b) Vista estructurada del PNC *P*

c) Detección del anti-patrón AP10

Fig. 5.21. PNC que ejemplifica el anti-patrón AP10

Las Figuras **5.21** b) y c) muestran la vista estructurada del PNC y la detección del anti-patrón *AP10* respectivamente. El problema de este ejemplo es que si se ejecuta la excepción definida por el elemento *Cancel*, se impide la sincronización de las instancias múltiples en paralelo, causando un bloqueo en la sincronización del *Multiple Instances*. Los mensajes *msg4* y msg5 pueden no ejecutarse debido a que no se puede llevar a cabo dicha sincronización.

Solución a los Anti-Patrones AP9 y AP10

Existen dos posibles soluciones para eliminar estos anti-patrones: eliminar el elemento *Cancel* o definir el *Cancel* tal que su *Scope* incluya a todos los elementos del PNC con instancias múltiples. En el modelo de PNC de la Figura **5.21** lo correcto es definir el *Cancel* de manera tal que el *Scope* del mismo incluya al *Multiple Instances*, con lo cual si se ejecuta una excepción se cancelen todas las instancias simultáneamente.

5.3.7. Bloqueo en la Gestión de Excepciones

UP-ColBPIP provee dos constructores para la gestión de excepciones: *Exception y Cancel*. Estos constructores están compuestos de una secuencia que determina el alcance del gestor de excepciones y por una o más secuencias, donde cada secuencia representa a un gestor de una determinada excepción.

De la Tabla **5.3** se puede observar que todo PNC mínimo *P* cuya raíz sea *Exception* o *Cancel*, donde las secuencias que representan a un gestor de excepción tengan un elemento *Termination*, se cumple que *P* no es sólido. En el caso del *Exception*, esto se debe a que de acuerdo a su semántica de comportamiento, una vez ejecutado el gestor de excepciones, el flujo de control continúa con el siguiente elemento al *Exception*. En el caso del *Cancel*, se debe a que no es necesario agregar un *Termination*, ya que el *Cancel* finaliza el PNC cuando termina la ejecución del gestor de excepciones. A partir de este análisis se especificaron los siguientes anti-patrones.

Anti-Patrones AP11 y AP12

Descripción

Los anti-patrones *AP11* y *AP12* se especificaron a partir de los PNCs mínimos determinantes N° 19 y 20 de la Tabla **5.3**.

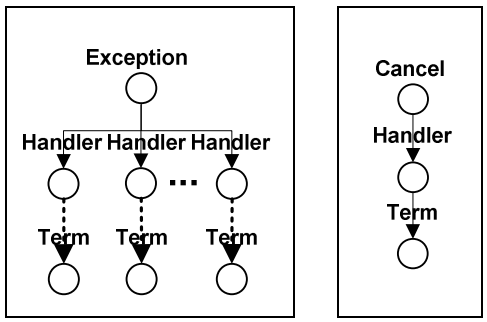
El anti-patrón *AP11* (Figura **5.22** a)) indica que se cumple si existe un *Exception* donde todos sus *Handlers* contienen un elemento *Termination*. No es determinante para el anti-patrón el alcance del *Exception*. La regla que lo define es:

Dado un modelo estructurado de PNC SP compuesto del conjunto de elementos E, AP11 se cumple si existe un elemento $e_1 \in E$*, donde* $e_1 = \mathit{Exception}$*, tal que para cada elemento* s_i*, donde* $\mathit{Parent}(s_i) = e_1$ *y* $s_i = \mathit{Handler}$*, existe un elemento* $e_i = \mathit{Termination}$ *tal que* $s_i \in \mathit{Ancestors}(e_i)$

El anti-patrón *AP12* (Figura **5.22** b)) ocurre cuando un modelo estructurado de PNC estructurado está compuesto de un constructor *Cancel* y alguno de los *Handlers* contiene un elemento *Termination*. La descripción de este anti-patrón es análoga a la del anti-patrón

AP11, difiere en que la raíz es un *Cancel* en lugar de un *Exception*. La regla que lo define es la siguiente:

Dado un modelo estructurado de PNC SP compuesto del conjunto de elementos E, AP12 se cumple si existen tres elementos $e_1, e_2, e_3 \in E$ *donde* $e_1 = Termination$, $e_2 = Handler$, $Parent(e_1) = e_2$ *y* $Parent(e_2) = Cancel$.

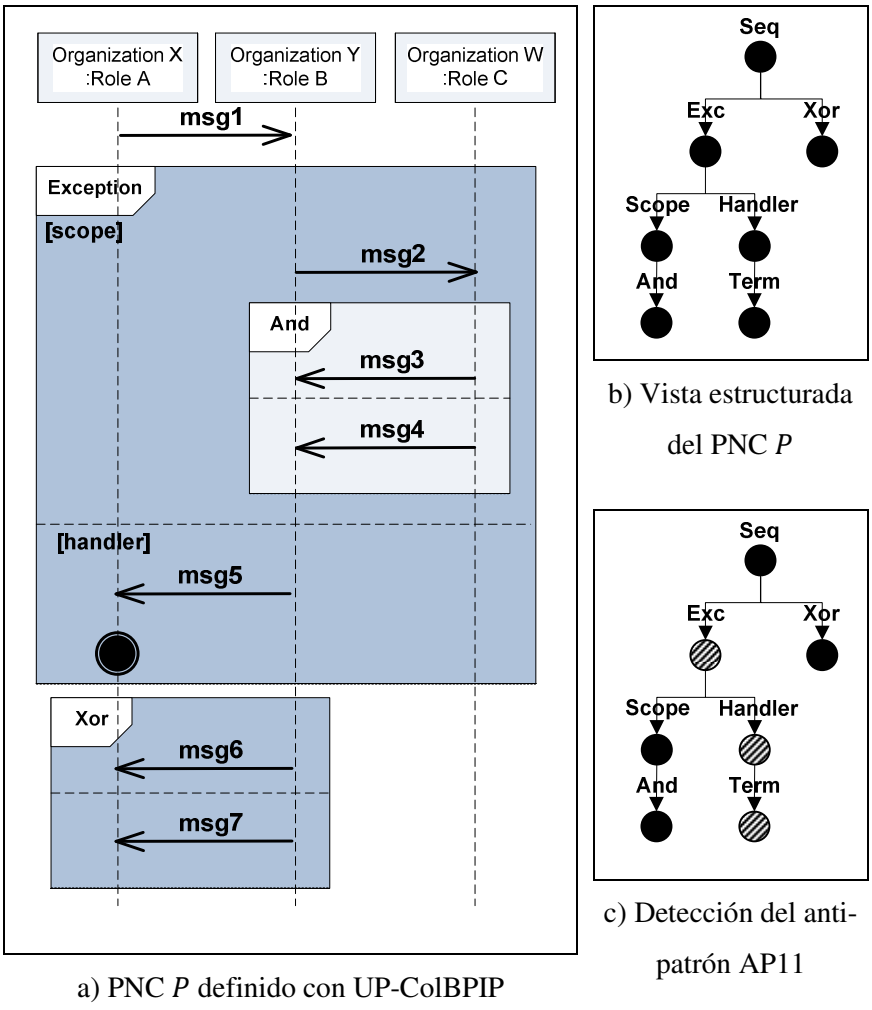


a) AP11 b) AP12

Fig. 5.22. Anti-patrones AP11 y AP12

Ejemplo

El protocolo de interacción UP-ColBPIP de la Figura **5.23** a) se inicia cuando A envía a B el mensaje *msg1*. Luego, B envía el mensaje *msg2* a C, y C responde de manera simultánea con los mensajes *msg3* y *msg4*. Estas interacciones entre B y C no deben superar un determinado tiempo. Caso contrario, ocurre una excepción en la cual B le envía a A el mensaje *msg5*, y luego finaliza el PNC. Si no ocurre ninguna excepción, B opta por enviar a A el mensaje *msg6* o el mensaje *msg7*, y el protocolo finaliza.



a) PNC *P* definido con UP-ColBPIP

b) Vista estructurada del PNC *P*

c) Detección del anti-patrón AP11

Fig. 5.23. PNC que ejemplifica el anti-patrón AP11

Las Figuras **5.23** b) y c) muestran la vista estructurada del PNC y la detección del anti-patrón *AP11* respectivamente. El problema en este ejemplo es que en caso que se ejecute el gestor de excepciones *Handler*, su elemento *Termination* impide que se continúe con la ejecución del PNC donde finaliza el elemento *Exception*. El elemento *Termination* ocasiona un bloqueo y el *Xor* que sigue a continuación del *Exception* no se va a ejecutar. Este comportamiento no es adecuado para un elemento *Exception*, debido a que su semántica indica que una vez finalizado el gestor de excepciones, el protocolo continúa con su ejecución.

Este comportamiento tampoco es adecuado para un elemento *Cancel*, ya que el mismo finaliza la ejecución del PNC una vez que se ejecuta el *Handler*, con lo cual agregar un elemento *Termination* es redundante.

Solución

La solución para eliminar el bloqueo en la gestión de excepciones es eliminar los elementos *Termination* dentro del gestor de excepciones.

5.4. Conclusiones

El método de verificación de modelos de PNCs basado en anti-patrones de comportamiento de un lenguaje de PNCs propuesto en este capítulo implica realizar la búsqueda de anti-patrones en un modelo o especificación de PNC definido con dicho lenguaje. Para ello, el nuevo enfoque presentado en este capítulo permite llevar a cabo de manera sistemática la especificación de anti-patrones de comportamiento de modelos estructurados de PNCs para un lenguaje de PNC dado a partir de un análisis de los constructores de dicho lenguaje.

El enfoque propuesto es independiente de la semántica de cualquier lenguaje específico de modelado de PNCs, lo que implica que sea flexible y adaptable para definir anti-patrones de distintos lenguajes de PNCs y, por lo tanto, puede ser usado para definir métodos de verificación aplicables a diferentes etapas del desarrollo, tanto de modelos como especificaciones de PNCs.

Este enfoque difiere de los enfoques existentes para definir anti-patrones en dos aspectos. Por un lado, es posible definir anti-patrones que soporten flujos de control complejos, lo cual no es contemplado por los enfoques existentes. Por otro lado, en estos enfoques los anti-patrones son definidos mediante una inspección manual de repositorios de procesos de negocio o de casos de estudio. Aunque se utilizan casos reales, este tipo de análisis lleva a definir anti-patrones que contemplan problemas comunes a los diseñadores que modelaron dichos procesos. Sin embargo, podrían existir otros tipos de errores que no son posibles de detectar por estos enfoques. En cambio, el enfoque propuesto analiza combinaciones de todos los constructores de un lenguaje de modelos de PNC, lo que permite verificar un conjunto más amplio de PNCs definidos con dicho lenguaje.

El enfoque propuesto se basa en los conceptos de PNC mínimo y perfil de no solidez de un lenguaje para definir anti-patrones. La importancia de los PNCs mínimos es que todo modelo de PNC va a estar conformado por alguna de las combinaciones de elementos definidas en los mismos. Luego, a partir de los PNCs mínimos no sólidos que conforman el perfil de no solidez de un lenguaje, es posible obtener todas las combinaciones de constructores de flujo de control que pueden conducir a la no solidez de un PNC, lo que es clave para la especificación de anti-patrones de comportamiento.

Como parte del enfoque se consideró la heterogeneidad de los lenguajes de PNCs, ya que las semánticas de los constructores de los lenguajes en muchos casos no son similares o bien tienen un conjunto diferente de constructores. Esto significa que las combinaciones de constructores de flujo de control que llevan a que un modelo de PNC sea no sólido depende de los constructores del lenguaje utilizado, por lo que se propuso que la especificación de anti-patrones se realice por cada lenguaje de PNCs.

Se demostró la aplicación del enfoque propuesto para especificar anti-patrones de comportamiento para el lenguaje UP-ColBPIP. Los anti-patrones especificados incluyen constructores de flujo de control simples y complejos. En particular, se identificaron anti-patrones que determinan bloqueos en secuencias, ciclos, caminos paralelos con sincronización avanzada, caminos mutuamente excluyentes, caminos con instancias múltiples, y en el manejo de excepciones y cancelaciones. Para cada anti-patrón se identificó el problema que representa y una posible solución al mismo. La notación gráfica propuesta facilita el entendimiento de las reglas que definen a cada anti-patrón. Si bien otros autores propusieron notaciones gráficas para especificar anti-patrones de comportamiento (Laue & Awad 2010), éstas no han sido consideradas debido a que son dependiente del lenguaje BPMN y dificulta su aplicación a otros lenguajes.

Si bien para verificar los PNCs mínimos de UP-ColBPIP se utilizó el método de verificación basado en GI-Nets propuesto en el Capítulo 4, los PNCs mínimos de un lenguaje pueden ser verificados con cualquier método de verificación de modelos de PNCs. De esta manera para aquellos PNCs mínimos que contienen elementos de flujo de control complejos, se puede utilizar un método de verificación que soporte dichos constructores, tal como el método basado en GI-Nets, y otro método distinto para aquellos PNCs mínimos con constructores simples. Además, un PNC mínimo podría ser verificado, por ejemplo,

utilizando un método basado en álgebra de procesos y otro PNC mínimo podría ser verificado con un método basado en redes de Petri. Esto facilita la detección de anti-patrones para todo tipo de constructores de flujo de control.

La verificación formal de los PNCs mínimos de un lenguaje para determinar si son o no sólidos se realiza una sola vez. Esto permite que el rendimiento de los métodos formales de verificación que se utilizan para determinar la solidez de los PNCs mínimos deje de ser un aspecto importante, ya que se aplican a modelos de PNCs con muy pocos elementos y una sola vez. Además, el resultado de verificación que se obtiene para un PNC mínimo puede ser generalizado y reutilizado a través de un anti-patrón de comportamiento.

Finalmente, debido a que los anti-patrones son definidos para modelos estructurados, los modelos de PNCs a verificar deben ser estructurados. Cuando se utilizan lenguajes estructurados que sólo generan modelos estructurados esto es directo. Sin embargo, para aplicar el método de verificación basado en anti-patrones a modelos no estructurados de PNCs, como los que se pueden definir con BPMN, los mismos tienen que ser previamente transformados a modelos estructurados. Esto se puede lograr mediante métodos de estructuración de procesos como los presentados en (Vanhatalo, Völzer & Koehler 2009).

6 Generación de Procesos de Negocio Colaborativos con Alineación de Comportamiento

En este capítulo se presenta un método de transformación que permite generar, a partir de un modelo de PNC, una especificación de PNC alineada, desde el punto de vista del comportamiento, con dicho modelo. Se describe el concepto de alineación de comportamiento (Sección 6.1) y se presenta un método de transformación para PNCs (Sección 6.2), que permite generar especificaciones a partir de modelos de PNCs garantizando alineación de comportamiento entre ellos. Se muestra un caso de estudio en el cual se utiliza el patrón de transformación para generar especificaciones de PNCs en WS-CDL a partir de modelos de PNCs definidos con el lenguaje UP-ColBPIP (Sección 6.3). Finalmente, se presentan las conclusiones (Sección 6.4).

6.1. Alineación de Comportamiento

En una metodología de desarrollo dirigido por modelos de sistemas de información orientados a procesos, se espera que el comportamiento definido en un modelo de PNC, expresado con un lenguaje independiente de la plataforma, se mantenga en la especificación generada a partir de dicho PNC, definida con un lenguaje específico de la plataforma. Esto es conocido como *alineación de comportamiento* (Weidlich et al. 2009).

Existen diferentes tipos de alineación de comportamiento (Weidlich et al. 2009). la cual puede ser parcial o total. La alineación parcial ocurre, por ejemplo, entre un proceso de integración (privado) y un proceso de interfaz (público), en donde ambos representan las tareas a realizar por una de las organizaciones involucradas en la colaboración. Si bien el proceso de integración debe satisfacer el comportamiento de su respectivo proceso de interfaz, debido a que puede tener tareas privadas que no forman parte del proceso de interfaz (Basten & van der Aalst 2001), el proceso de integración puede estar parcialmente alineado al comportamiento de éste último.

La alineación es total cuando ambos procesos tienen exactamente el mismo comportamiento. Este caso ocurre entre un modelo conceptual de PNC y su respectiva especificación tecnológica, debido a que ambos representan al mismo PNC. Por lo cual, en esta tesis, el concepto de *alineación de comportamiento* refiere al de alineación total.

A efectos de evitar las posibles ambigüedades de los lenguajes de modelado y especificación de PNCs, para determinar alineación de comportamiento se recurre al uso de modelos formales. En particular se utiliza el lenguaje formal GI-Nets definido en el Capítulo 3, y se establece que existe alineación de comportamiento entre un modelo de PNC ϕ_i y su correspondiente especificación ϕ_o, si ambos tienen el mismo modelo de comportamiento formal, es decir, si las GI-Nets GI_{N_i} y GI_{N_o} que formalizan el modelo y la especificación respectivamente son iguales.

Definición 6.1. (***Alineación de comportamiento de PNCs***) Sean ϕ_i y ϕ_o un modelo y una especificación de PNC respectivamente, y sean GI_{N_i} y GI_{N_o} sus respectivas representaciones formales basadas en GI-Nets. Si $GI_{N_i} = GI_{N_o}$, entonces ϕ_i y ϕ_o están alineados desde el punto de vista del comportamiento.

Para generar especificaciones de PNCs a partir de modelos de PNCs es necesario definir una especificación de transformación de modelos de acuerdo a la Definición **3.6**, donde intervienen un lenguaje de origen y otro de destino. En general, los métodos de transformación de modelos existentes definen un mapeo entre los constructores de dichos lenguajes, de acuerdo a lo que el diseñador del método considera adecuado como transformación. El mapeo define que el comportamiento de un constructor del lenguaje de origen se transforma en un conjunto de constructores del lenguaje destino. No obstante, en estos casos no se lleva a cabo una evaluación formal que determine que el comportamiento de estos constructores está realmente alineado, es decir, que tienen la misma semántica de comportamiento. Esto implica que, cuando se ejecuta una máquina de transformación, no haya garantía de que la especificación de PNC de salida mantenga el comportamiento definido en el modelo de PNC de entrada.

Para permitir la generación de modelos y especificaciones de PNCs alineados, se debe cumplir que para cada par de constructores de los lenguajes de origen y destino utilizados en la especificación de transformación, la representación formal del constructor del lenguaje de origen debe ser igual a la representación formal del constructor asociado en el lenguaje de destino. Esto implica una alineación de comportamiento entre un conjunto de constructores de dos lenguajes de PNCs, y por lo tanto, una alineación entre ambos lenguajes.

Definición 6.2. (*Alineación entre lenguajes de PNCs*) Sea $TE_{s-t} = (\lambda_s, \lambda_t, cc_{s-t})$ una especificación de transformación para los conjuntos de constructores λ_s y λ_t que pertenecen a dos lenguajes de PNCs. Además, sean $TE_s = (\lambda_s, GI^a_{M_s}, cc_s)$ y $TE_t = (\lambda_t, GI^a_{M_t}, cc_t)$ dos especificaciones de transformaciones para GI-Nets (basado en Definición **3.6**), donde $GI^a_{M_s}$ y $GI^a_{M_t}$ son dos conjuntos finitos de módulos GI abstractos de las especificaciones de transformación TE_s y TE_t tal que cada elemento $AM_s \in GI^a_{M_s}$ y $AM_t \in GI^a_{M_t}$ formaliza un constructor $C_s \in \lambda_s$ y $C_t \in \lambda_t$ respectivamente. Se dice que hay alineación de comportamiento entre dos lenguajes de PNCs que tienen los conjuntos de constructores λ_s y λ_t respectivamente si para cada constructor $C_s \in \lambda_s$ y $C_t \in \lambda_t$, donde $cc_{s-t}(C_s) = C_t$, hay un módulo GI abstracto $AM_s \in GI^a_{M_s}$ y un módulo GI abstracto $AM_t \in GI^a_{M_t}$ tal que $cc_s(C_s) = AM_s$ y $cc_t(C_t) = AM_t$, y $AM_s = AM_t$.

Tabla 6.1. Ejemplo de alineación de comportamiento entre dos lenguajes de PNCs

Constructores A	Constructores B	Módulo GI abstracto
C_{A_1}	C_{B_1}	$GI^a_{M_1}$
C_{A_2}	C_{B_2}	$GI^a_{M_2}$
…	…	…
C_{A_n}	C_{B_n}	$GI^a_{M_n}$

Por ejemplo, dos lenguajes de PNCs que tienen los conjuntos de constructores A y B están alineados si existe una especificación de transformación TE_{s-t} que asocia cada constructor de A con un constructor de B, tal que cada par de constructores asociados tienen el mismo modelo formal GI-Nets. En la Tabla **6.1**, se muestra la asociación de

constructores de a pares, donde cada par está compuesto de un constructor del lenguaje A y otro del lenguaje B, y además está asociado a un único módulo GI abstracto que representa el comportamiento formal de ambos constructores.

6.2. Método de Transformación para PNCs que Garantiza Alineación de Comportamiento

6.2.1. Patrón de Transformación para PNCs

Para la generación de modelos y especificaciones de PNCs alineados se propone determinar si ambos tienen el mismo modelo formal. La Figura **6.1** muestra las transformaciones necesarias para generar modelos y especificaciones alineados. La Figura **6.1** a) muestra un *PNC Origen*, el modelo, y un *PNC Destino*, la especificación. La especificación de transformación para PNCs TE_{s-t} se utiliza para generar el *PNC Destino* a partir del *PNC Origen*, mientras que las especificaciones de transformación TE_s y TE_t permiten generar las GI-Nets que formalizan los *PNC Origen* y *PNC Destino* respectivamente. Cada PNC tiene su propia GI-Net donde GI_{N_i} se genera a partir del *PNC Origen* y GI_{N_o} se genera a partir del *PNC Destino*. Si al aplicar dichas transformaciones, las GI-Nets resultantes GI_{N_i} y GI_{N_o}son iguales, entonces, según Definición **6.1**, existe alineación de comportamiento entre el modelo y la especificación del PNC.

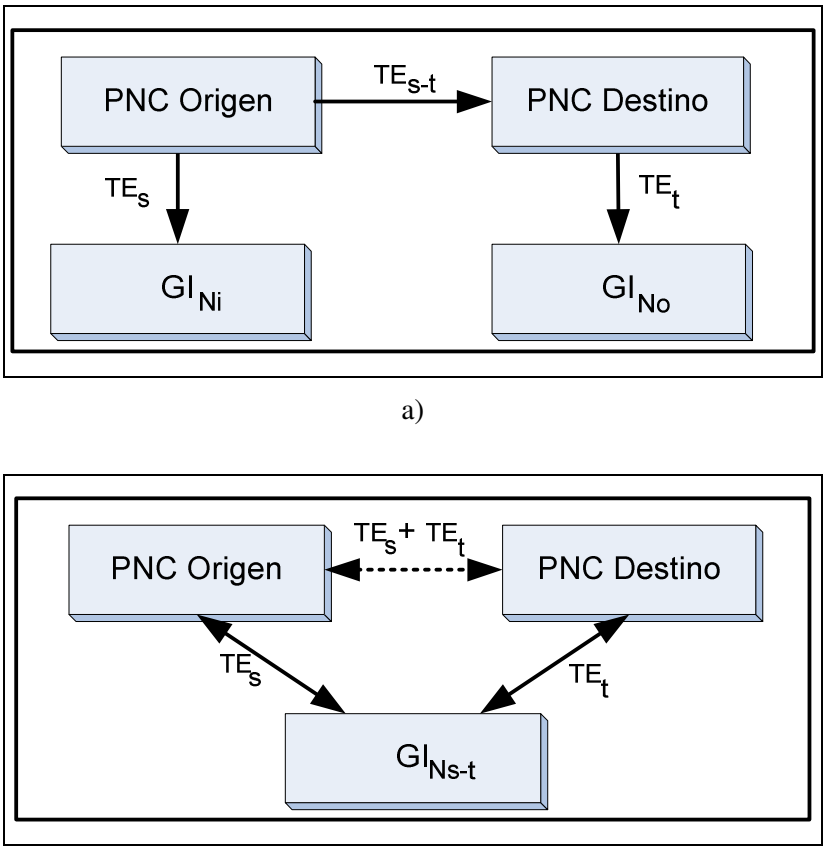


Fig. 6.1. Alineación de comportamiento entre PNCs

Como una especificación de transformación establece una función biyectiva entre los constructores de los lenguajes de origen y destino, dicha especificación puede ser utilizada de manera bidireccional tal que el *PNC Origen* se transforme en destino y el *PNC Destino* en origen. De esta manera se puede obtener una configuración como la mostrada en la Figura **6.1** b), donde las especificaciones de transformación TE_s y TE_t permiten generar una única GI-Net que formaliza tanto al *PNC Origen* como al *PNC Destino*. Ambas especificaciones de transformación pueden ser utilizadas en conjunto para generar el *PNC Destino* a partir del *PNC Origen* y viceversa. Esto es posible, debido a que si ambos PNCs tienen alineación de comportamiento, entonces también tienen la misma especificación formal. De esta manera, para generar un PNC a partir de otro, se utiliza la combinación de las especificaciones de transformación TE_s y TE_t, para generar un modelo formal GI-Nets

intermedio que formaliza a ambos PNCs y luego se utiliza dicho modelo formal para generar el *PNC Destino*.

Por lo tanto, se propone un patrón de transformaciones para PNCs basado en el uso de modelos formales de GI-Nets, que genera modelos y especificaciones de PNCs alineados. En este patrón de transformaciones, las GI-Nets son utilizadas como representaciones formales intermedias de PNCs a partir de las cuales es posible generar modelos y/o especificaciones de PNCs. La Figura **6.2** muestra este patrón de transformaciones, en el cual las especificaciones de transformación TE_s y TE_t permiten generar un único modelo formal GI-Net $GI_{N_{s-t}}$ que formaliza tanto al modelo como a la especificación de un mismo PNC. Estas especificaciones de transformación son también utilizadas para generar modelos y especificaciones de PNCs a partir de dicho modelo formal.

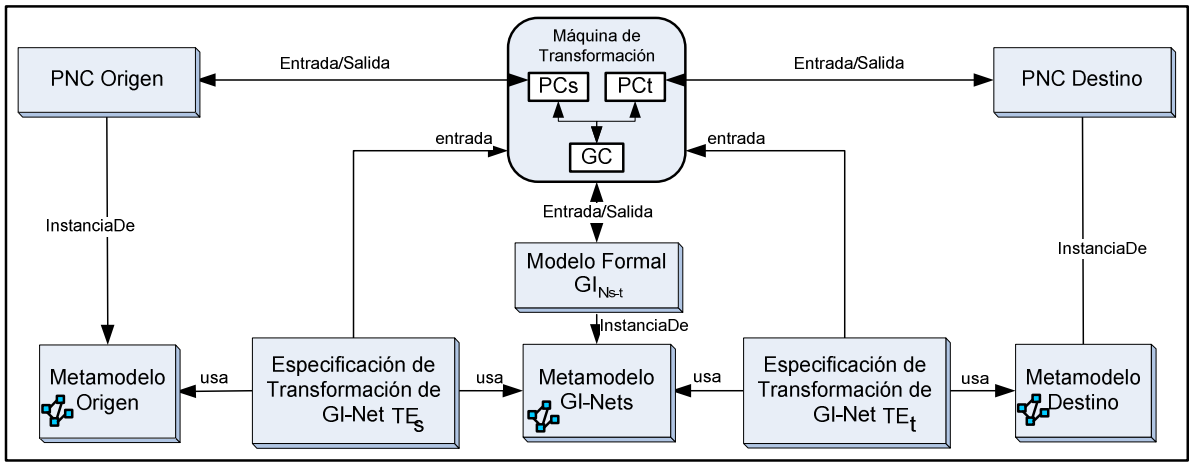


Fig. 6.2. Patrón de transformación para PNCs

Este patrón de transformaciones está compuesto de dos patrones de transformación para GI-Nets que comparten entre sí la máquina de transformación y el metamodelo de GI-Nets. El primer patrón está compuesto por un *Metamodelo Origen*, un *PNC Origen* instancia de dicho metamodelo, y una especificación de transformación para GI-Nets TE_s. El segundo patrón está compuesto por un *Metamodelo Destino*, un *PNC Destino* instancia de dicho metamodelo, y una especificación de transformación para GI-Nets TE_t. La máquina de transformación (compartida por ambos patrones) está compuesta de dos

componentes de transformación PC_s y PC_t que permiten la lectura y generación del *PNC Origen* y el *PNC Destino* respectivamente, y de un componente de transformación para GI-Net GC, que permite la lectura y generación de GI-Nets.

El patrón de transformación para PNCs soporta una transformación bidireccional. Si bien este patrón de transformación indica que existe un *PNC Origen* y un *PNC Destino* (y sus correspondientes metamodelos), es posible utilizar el *PNC Destino* para generar el *PNC Origen*. De esta forma se puede generar una especificación de PNC a partir de un modelo y viceversa. Esto da lugar a dos tipos de transformaciones.

Para el primer tipo de transformación, la entrada del patrón es un modelo de PNC, el *PNC Origen*. El componente PC_s interpreta el *PNC Origen* y el componente de GI-Net GC genera un modelo formal GI-Net del *PNC Origen* (el modelo). Esta GI-Net es utilizada como entrada para el componente PC_t, el cual genera el *PNC Destino*, la especificación de PNC, que es la salida del patrón de transformación.

Para el segundo tipo de transformación, la entrada del patrón es una especificación de PNC, el *PNC Destino*. El componente interpreta el *PNC Destino* PC_t y el componente de GI-Net GC genera un modelo formal de GI-Net del *PNC Destino* (la especificación de PNC). Esta GI-Net es utilizada como entrada para el componente de PNC PC_s el cual genera el *PNC Origen*, que es la salida del patrón de transformación.

El patrón de transformación para PNCs se describe formalmente en la Definición **6.3**, donde las funciones ec_i y cc_s son parte del componente PC_s, las funciones ec_o y cc_t son parte del componente PC_t , y la función ce es parte del componente de PNC G.

Definición 6.3. (***Patrón de transformación para PNCs***) Un patrón de transformación para PNCs es una tulpa $T = (TE_s, TE_t, T_s, T_t, t)$, donde:

1. $TE_s = (\lambda_s, GI_M^a, cc_s)$ y $TE_t = (\lambda_t, GI_M^a, cc_t)$ son las especificaciones de transformación para GI-Nets de los lenguajes de origen y destino respectivamente, donde:
   a. λ_s y λ_t son dos conjuntos de constructores de dos lenguajes de PNCs, y
   b. como TE_s y TE_t comparten el mismo conjunto de módulos GI abstractos que formalizan λ_s y λ_t, los lenguajes de origen y destino están alineados según Definición **6.2**,
2. $T_s = (TE_s, \Phi_\mathrm{i}, \phi_i, GI_{M_i}^c, GI_N^*, ec_i, ce, GI_{N_i}, t_s)$ es un patrón de transformación para GI-Nets, donde TE_s es la especificación de transformación para GI-Net de T_s y ϕ_i es el PNC de entrada de T_s,
3. $T_t = (TE_t, \Phi_\mathrm{o}, \phi_o, GI_{M_o}^c, GI_N^*, ec_o, ce, GI_{N_o}, t_t)$ es un patrón de transformación para GI-Nets, donde TE_t es la especificación de transformación para GI-Net de T_t y ϕ_o es el PNC de entrada de T_t,
4. t: $\Phi_\mathrm{i} \rightarrow \Phi_\mathrm{o}$ es una función de mapeo biyectiva que define una transformación desde un PNC $\phi_i \in \Phi_\mathrm{i}$ a un PNC $\phi_o \in \Phi_\mathrm{o}$ tal que $t = {t_t}^{-1} \circ t_s$.
5. t^{-1} es la función inversa de t, y define una transformación desde un PNC $\phi_o \in \Phi_\mathrm{o}$ a un PNC $\phi_i \in \Phi_\mathrm{i}$ tal que $t^{-1} = ({t_t}^{-1} \circ t_s)^{-1} = {t_s}^{-1} \circ t_t$.

(1) El patrón de transformación para PNCs está compuesto de las especificaciones de transformación para GI-Nets TE_s y TE_t de los lenguajes de origen y destino, donde dichos lenguajes están alineados, con lo cual sus conjuntos de constructores λ_s y λ_t están formalizados por un mismo conjunto de módulos GI abstractos GI_M^a.

(2 y 3) Las especificaciones de transformación para GI-Nets TE_s y TE_t forman parte de los patrones de transformación para GI-Nets T_s y T_t respectivamente. Estos patrones permiten generar modelos formales de PNCs definidos con los lenguajes de origen y destino respectivamente.

(4) El patrón de transformación para PNCs define la función de mapeo t que permite generar el PNC de destino a partir del PNC de origen. La función t es una función compuesta por las funciones ${t_t}^{-1}$ y t_s. t_s forma parte del patrón de transformación para GI-Nets T_s y permite obtener el modelo formal del PNC de origen. t_t forma parte del patrón de

transformación para GI-Nets T_t y permite obtener el modelo formal del PNC de destino. Como t_t es una función biyectiva, ${t_t}^{-1}$ permite obtener el PNC de destino a partir de un modelo formal. En consecuencia, la función t es la composición de las funciones ${t_t}^{-1}$ y t_s, lo cual implica que a través de t_s, que tiene como entrada el PNC de origen, se obtiene como salida una GI-Net, la cual es entrada de la función ${t_t}^{-1}$ y genera el PNC de destino.

(5) t^{-1} es la función inversa de t, que permite generar un modelo a partir de una especificación de PNC. La descripción es análoga a la realizada para t.

6.2.2. Condición Suficiente para Alineación de Comportamiento

Dado un patrón de transformaciones para PNCs $T = (TE_s, TE_t, T_s, T_t, t)$, y dos patrones de transformación para GI-Nets $T_s = (TE_s, \Phi_i, \phi_i, GI^c_{M_i}, GI^*_N, ec_i, ce, GI_{N_i}, t_s)$, y $T_t = (TE_t, \Phi_o, \phi_o, GI^c_{M_o}, GI^*_N, ec_o, ce, GI_{N_o}, t_t)$, se desea determinar si los PNCs ϕ_i y ϕ_o presentan alineación de comportamiento.

Lema 6.1. Si un dado PNC ϕ_o es generado a partir de un PNC ϕ_i por medio del patrón de transformación para PNCs T, entonces ϕ_i y ϕ_o presentan alineación de comportamiento.

Demostración.

Asúmase que el PNC ϕ_o fue generado a partir del PNC ϕ_i por medio del patrón de transformación para PNCs T. Entonces, por Definición **6.3**, se sabe que existen:

(1) dos patrones de transformación para GI-Nets $T_s \in T$ y $T_t \in T$,

(2) una función biyectiva $t \in T$ que genera el PNC ϕ_o a partir del PNC ϕ_i, donde:

(3) $t = {t_t}^{-1} \circ t_s$ y

(4) $t^{-1} = {t_s}^{-1} \circ t_t$, y

(5) $t_s \in T_s$ y $t_t \in T_t$ son dos funciones biyectivas que generan las GI-Nets de ϕ_i y ϕ_o respectivamente, tal que:

(6) $t_s(\phi_i) = GI_{N_i}$ y $t_t(\phi_o) = GI_{N_o}$.

Luego, por (2), (3), y (4), se sabe que:

(7) $t(\phi_i) = {t_t}^{-1}\big(t_s(\phi_i)\big) = \phi_o$ y

(8) $t^{-1}(\phi_o) = {t_s}^{-1}\big(t_t(\phi_o)\big) = \phi_i$.

Además, por (3), (4), y (6) se tiene que:

(9) $t(\phi_i) = {t_t}^{-1}\big(GI_{N_i}\big) = \phi_o$ y

(10) $t^{-1}(\phi_o) = {t_s}^{-1}\big(GI_{N_o}\big) = \phi_i$.

Finalmente, por (5) se sabe que t_s y t_t son funciones biyectivas, entonces por (6) se deben cumplir también las siguientes igualdades:

(11) ${t_s}^{-1}\big(GI_{N_i}\big) = \phi_i$ y

(12) ${t_t}^{-1}\big(GI_{N_o}\big) = \phi_o$.

Debido a que t_s y t_t son funciones biyectivas por (9) y (12) se obtiene que ${t_t}^{-1}\big(GI_{N_i}\big) = {t_t}^{-1}\big(GI_{N_o}\big) = \phi_o$, y por (10) y (11) ${t_s}^{-1}\big(GI_{N_o}\big) = {t_s}^{-1}\big(GI_{N_i}\big) = \phi_i$. Por lo tanto, como t_s y t_t son funciones biyectivas, se concluye que las GI-Nets GI_{N_i} y GI_{N_o} que formalizan los PNCs ϕ_i y ϕ_o tienen que ser iguales, es decir, $GI_{N_i} = GI_{N_o}$, que por Definición **6.1** implica que ϕ_i y ϕ_o presentan *alineación de comportamiento*. ■

La utilización del patrón de transformación para PNCs propuesto en la Definición **6.3** es una condición suficiente, pero no necesaria para generar PNCs alineados, ya que es

posible obtener el mismo resultado relajando alguna de las reglas de la definición de dicho patrón.

Por ejemplo, el patrón de transformación para PNCs propuesto está basado en la idea de que ambos lenguajes de PNCs deben tener la misma definición formal de sus constructores de acuerdo a la Definición **6.2**. Sin embargo, supóngase que se relaja la restricción 1 de la Definición **6.3** estableciendo que sólo un subconjunto de los constructores definidos en la especificación de transformación de ambos lenguajes de PNCs tienen la misma representación formal. Supóngase además que los conjuntos de constructores λ_1 y λ_2 están compuestos cada uno de cuatro constructores: *Message*, *Xor*, *And*, y *Or*, donde los primeros tres constructores tienen la misma representación formal con GI-Nets, mientras que la representación formal del constructor *Or* del conjunto λ_1 difiere de la del constructor *Or* del conjunto λ_2.

Supóngase ahora que hay un PNC ϕ_1 definido con los constructores de λ_1, donde ϕ_1 está compuesto solamente por elementos que son instancias de los constructores *Message*, *Xor*, y *And*. Si se utiliza el patrón de transformaciones para PNCs y se genera un PNC ϕ_o, el cual es instancia de λ_2, ambos PNC ϕ_i y ϕ_o van a estar alineados. Esto ocurre a pesar de que los constructores de PNCs λ_1 y λ_2 no están alineados de acuerdo a la Definición **6.2**. El motivo reside en que los PNC ϕ_i y ϕ_o solo están compuestos de elementos que tienen la misma representación formal. Pero, si se agregara a ϕ_i un elemento instancia del constructor *Or*, entonces ϕ_i y ϕ_o dejarían de estar alineados. Esto significa que relajar la definición del patrón de transformación para PNCs podría llevar a la generación de PNCs que no estén alineados.

Con este ejemplo, se muestra que relajar una restricción del patrón de transformación puede llevar a generar PNCs que pueden estar o no alineados. Por lo tanto, si bien no es necesario utilizar el patrón de transformación propuesto en la Definición **6.3** para generar dos PNCs alineados, por el Lema **6.1**, se sabe que la utilización de dicho patrón de transformación es suficiente para garantizar que ambos PNCs presenten alineación de comportamiento.

6.3. Caso de Estudio

En esta sección se describe un caso de estudio cuyo propósito es mostrar la aplicación del método de transformación propuesto en este capítulo. Las transformaciones son aplicadas a un modelo de PNC definido con UP-ColBPIP (Capítulo 2, Sección 2.2.1) para generar una especificación del PNC definida en WS-CDL (Capítulo 2, Sección 2.2.3) alineada con el modelo.

El caso de estudio consiste en aplicar el método de transformación al PNC *Gestión de Pronóstico de Demanda* (Figura **6.3**) presentado en el Capítulo 3, y verificado y corregido en el Capítulo 4. Para que la especificación pueda ser definida en WS-CDL, en el modelo definido usando UP-ColBPIP se reemplazó el elemento *Multiple Instances* por un *Loop*, ya que WS-CDL no provee constructores para instancias múltiples.

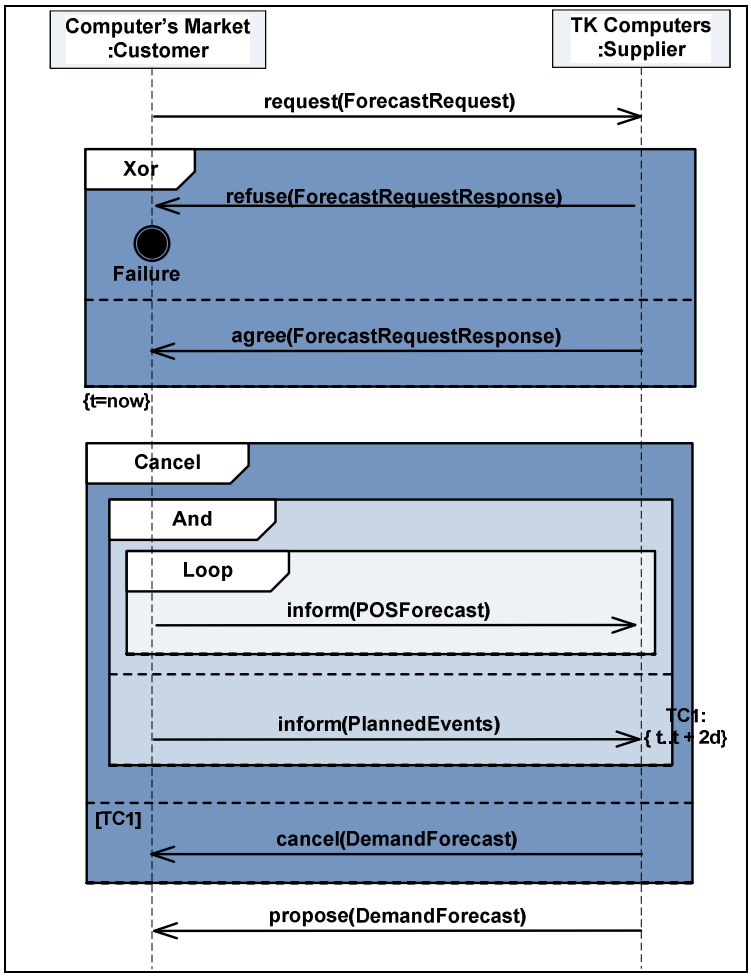


Fig. 6.3. PNC Gestión de Pronóstico de Demanda

6.3.1. Generación de especificaciones WS-CDL

Para aplicar el método de transformación propuesto a modelos UP-ColBPIP y especificaciones WS-CDL se materializa el patrón de transformación para PNCs definido en la Sección 6.2.1. El patrón de transformación materializado es $T = (TE_s, TE_t, T_s, T_t, t)$, siendo $TE_s = (\lambda_s, GI_M^a, cc_s)$ y $TE_t = (\lambda_t, GI_M^a, cc_t)$ las transformaciones de especificación de los lenguajes de origen y destino respectivamente, y $T_s = (TE_s, \Phi_i, \phi_i, GI_{M_i}^c, GI_N^*, ec_i, ce, GI_{N_i}, t_s)$ y

$T_t = (TE_t, \Phi_o, \phi_o, GI^c_{M_o}, GI^*_N, ec_o, ce, GI_{N_o}, t_t)$ los patrones de transformación para GI-Nets de los lenguajes de origen y destino respectivamente, donde:

- λ_s es el conjunto de constructores de UP-ColBPIP,
- λ_t es el conjunto de constructores de WS-CDL,
- GI^a_M son los módulos GI abstractos que formalizan a los constructores de λ_s y λ_t,
- ϕ_i es el modelo UP-ColBPIP mostrado en la Figura **6.3**,
- ϕ_o es la especificación WS-CDL que se genera a partir del modelo UP-ColBPIP ϕ_i,
- t es la máquina de transformación de un modelo UP-ColBPIP a una especificación WS-CDL

Para el modelo de PNC ϕ_i (Figura **6.3**) se aplicó el patrón de transformación T explicado. Se llevó a cabo el mapeo de los constructores de ambos lenguajes (Tabla **6.2**) de acuerdo a la similitud de sus semánticas de comportamiento y a la especificación de transformación informal propuesta en (Villarreal, Salomone & Chiotti 2006).

Tabla 6.2. Mapeo de constructores para UP-ColBPIP y WS-CDL

UP-ColBPIP	WS-CDL
Business Message	Interaction
Xor (data-driven)	A Choice and a Sequence activity for each interaction path
Xor (event-driven)	A Choice and a WorkUnit activity for each interaction path
And	Parallel activity
Loop While	WorkUnit. Guard="Repetition condition". Repeat="True"
Loop Until	WorkUnit. Guard="True". Repeat="Repetition condition"
Cancel	Exception WorkUnit
Termination	Condition in the "complete" attribute of the choreography

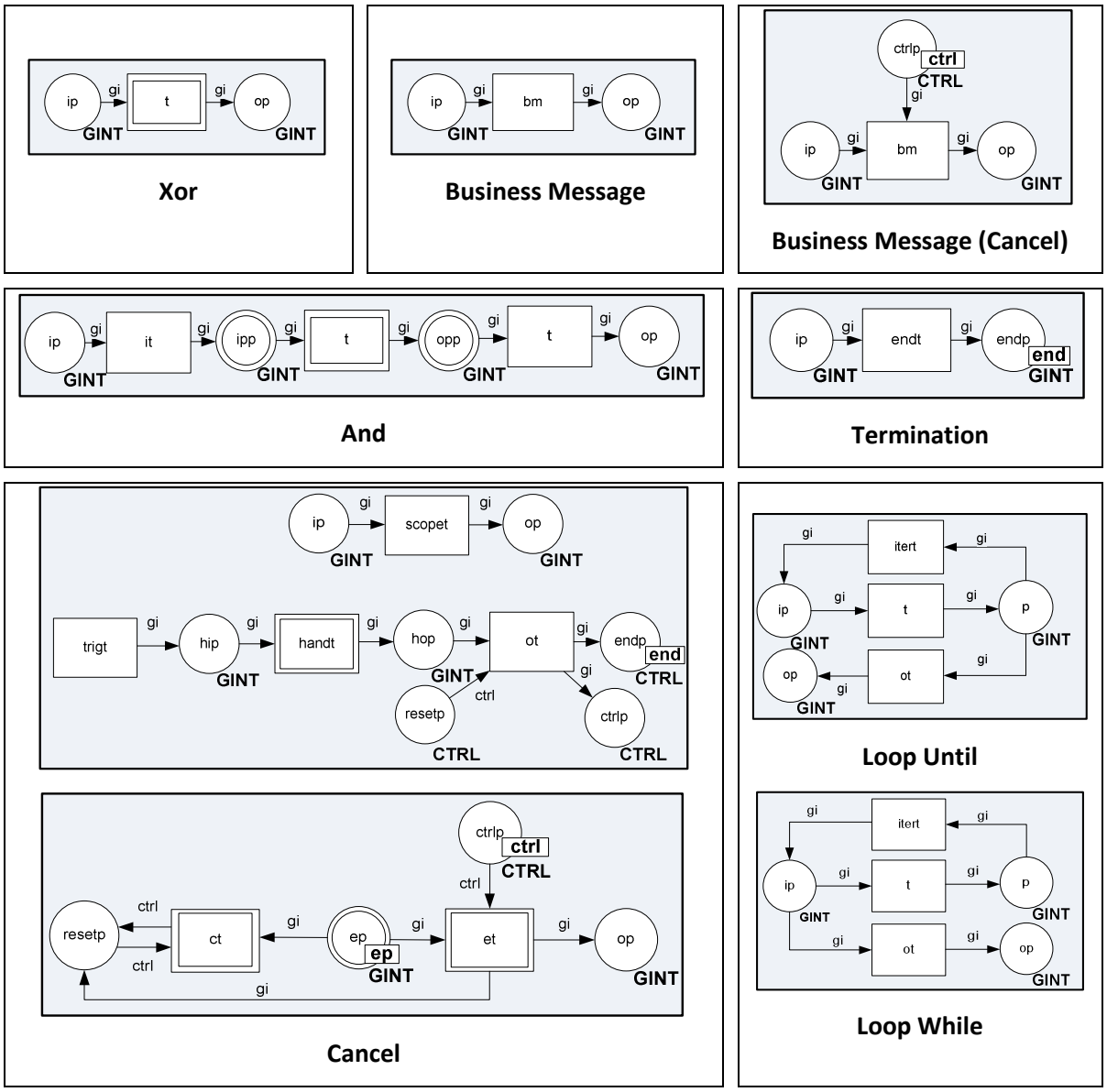


Fig. 6.4. Módulos GI abstractos para los constructores de UP-ColBPIP y WS-CDL

Para cada par de constructores del mapeo entre lenguajes, se definió un módulo GI abstracto. Para esto se reutilizaron los módulos GI abstractos definidos en el Capítulo 3, Sección 3.3 para el lenguaje UP-ColBPIP. La Figura **6.4** muestra los módulos GI abstractos utilizados que formalizan a cada par de constructores de la Tabla **6.2**. Estos módulos son utilizados por los patrones de transformación para generar modelos formales de PNCs definidos con UP-ColBPIP y WS-CDL.

```
1     <?xml version="1.0" encoding="ASCII"?>
2     <package xmlns:cdl="http://www.w3.org/2005/10/cdl" name="giNetUpcolbpip"
      targetNamespace="giNetUpcolbpip">
3       <choreography complete="globalizedTrigger('termination')" isolation="false"
      name="interactionprotocol">
4         <variableDefinitions>
5           <variable name="Termination"/>
6         </variableDefinitions>
7         <sequence>
8           <interaction name="BUSINESSMESSAGErequest(ForecastRequest)1" operation="request"/>
9           <choice>
10            <sequence>
11              <interaction name="BUSINESSMESSAGErefuse(FRR)1" operation="request"/>
12              <assign>
13                <copy>
14                  <target variable="getVariable('Termination')"/>
15                </copy>
16              </assign>
17            </sequence>
18            <sequence>
19              <interaction name="BUSINESSMESSAGEagree(FRR)1" operation="request"/>
20            </sequence>
21          </choice>
22          <workunit name="cancel">
23            <parallel>
24              <sequence>
25                <workunit guard="true" name="loopUntil1" repeat="cond">
26                  <interaction name="BUSINESSMESSAGEinform(POSForecast)1" operation="inform"/>
27                </workunit>
28              </sequence>
29              <sequence>
30                <interaction name="BUSINESSMESSAGEinform(plannedEvents)1" operation="inform"/>
31              </sequence>
32            </parallel>
33          </workunit>
34        </sequence>
35        <exceptionBlock name="cancel">
36          <workunit name="cancel1">
37            <interaction name="BUSINESSMESSAGEcancel(Demand)1" operation="request"/>
38          </workunit>
39        </exceptionBlock>
40      </choreography>
41    </package>
```

Fig. 6.5. Código WS-CDL generado con el método de transformación para PNCs

La Figura **6.5** muestra el código WS-CDL generado por el método de transformación. Esta especificación del PNC es una secuencia de elementos que componen el bloque delimitado por los elementos `<sequence></sequence>` (líneas 7 y 34). El primer mensaje del modelo de PNC *request(ForecastRequestResponse)* es transformado en el elemento *Interaction* (línea 8). El elemento *Xor* del modelo es transformado en el bloque *Choice* (líneas 9 y 21). Los caminos de interacción del *Xor* son transformados en elementos *Sequence* (líneas 10, 17, 18, y 20), que conforman los caminos alternativos del bloque *Choice*. Estos bloques *Sequence* contienen los elementos de tipo *Interaction* que representan a los mensajes *refuse(ForecastRequestResponse)* (línea 11) y *agree(ForecastRequestResponse)* (línea 19). El elemento *Termination* del modelo UP-

ColBPIP se transforma en el atributo *complete* del elemento *Choreography* de WS-CDL (línea 3), y mediante el bloque de elementos *Assign* (líneas 12 y 16).

El siguiente elemento transformado del modelo UP-ColBPIP es el *Cancel*. En la especificación WS-CDL, el bloque *Workunit* (líneas 22 y 33) representa al alcance del *Cancel*, y el bloque *ExceptionBlock* (líneas 35 y 39) representa al manejador de excepciones. Este manejador tiene el elemento *Interaction* (línea 37) que representa al mensaje *cancel(DemandForecast)* del modelo UP-ColBPIP.

El elemento *And* del modelo UP-ColBPIP es transformado en el bloque *Parallel* (líneas 23 y 32). Los caminos de interacción del *And* son transformados en bloques *Sequence* (líneas 24, 28, 29, y 31). El *Loop* de UP-ColBPIP es transformado en el bloque *Workunit* (líneas 25 y 27). El mensaje *inform(POSForecast)* es transformado en el elemento *Interaction* (línea 26). El otro mensaje en paralelo *inform(PlannedEvents)* es transformado en el elemento *Interaction* (línea 30).

6.4. Conclusiones

El método de transformación propuesto permite generar, a partir de modelos de PNCs, especificaciones de PNCs alineadas con dichos modelos. Por medio de la generación de una única representación formal basada en modelos GI-Nets, se logra la alineación de comportamiento entre dos PNCs definidos con distintos lenguajes. El método contribuye a mejorar la calidad del código generado usando metodologías de desarrollo dirigido por modelos de sistemas de información orientados a procesos inter-organizacionales, ya que permite garantizar que el comportamiento de un modelo de PNC, definido en etapas tempranas del desarrollo (solución de negocio), se mantiene en la especificación de dicho PNC (solución tecnológica). Esto conduce a que la solución tecnológica sea consistente con la solución de negocio y refleje el comportamiento esperado por esta última.

El método propuesto consiste primero en generar un modelo formal basado en GI-Nets de un modelo de PNC de entrada, y luego generar la especificación de dicho PNC a partir del modelo formal definido con GI-Nets. El patrón de transformaciones para PNCs definido reutiliza el patrón de transformación para la generación de modelos formales basados en GI-Nets presentado en el Capítulo 3, Sección 3.2.

La condición suficiente para alineación de comportamiento entre modelos y especificaciones de PNCs definida formalmente, permitió demostrar que el método garantiza alineación de comportamiento.

Además de generar PNCs alineados, el proceso de transformación bidireccional provisto por el patrón de transformación propuesto permite generar una especificación de PNC desde un modelo de PNC, y viceversa. Esto es posible debido al uso de una representación formal intermedia con una GI-Net, la cual formaliza el comportamiento de ambos PNCs. Dicha representación formal intermedia, también puede ser utilizada para detectar problemas en el comportamiento de los PNCs. En particular, es posible determinar que si un modelo de PNC cumple las propiedades de comportamiento de acuerdo al método de verificación presentado en el Capítulo 4, entonces su correspondiente especificación también cumplirá las mismas propiedades. De manera análoga, si el modelo de PNC no cumple las propiedades, su correspondiente especificación tampoco lo hará. Esto también contribuye a aumentar la calidad de los artefactos generados en metodologías de desarrollo dirigido por modelos.

Cuando se define un patrón de transformación para dos lenguajes de PNCs, es probable que existan constructores que no puedan ser alineados, por ejemplo, un dado lenguaje puede tener constructores para sincronización avanzada del flujo de control, mientras que el otro no. En estos casos, es necesario acordar una representación formal unificada de los constructores con la misma semántica de comportamiento entre ambos lenguajes. Sin embargo, puede haber constructores para los cuales no es posible acordar dicha representación formal. En otros casos inclusive puede ocurrir que sólo una parte del comportamiento de ambos constructores pueda ser acordada, ya que es usual que existan discrepancias entre la semántica de los constructores de distintos lenguajes. Si bien la aplicabilidad de este método de transformación depende de la compatibilidad de los lenguajes de PNCs, es esperable que exista un subconjunto de constructores de ambos lenguajes que sean compatibles y puedan ser alineados.

El patrón de transformación para PNCs propuesto sólo genera el esqueleto del flujo de control de un PNC, dejando de lado los aspectos de datos que tienen que ver con el intercambio de información entre las organizaciones. Sin embargo, la definición modular del mismo permite la adición de otros patrones de transformación para generar modelos

formales intermedios de los aspectos de datos de los PNCs. Esta separación de funcionalidades permite la definición de métodos para la alineación, verificación y validación de aspectos relacionados a los datos definidos en los PNCs.

El patrón de transformación para PNCs fue aplicado a un caso de estudio con los lenguajes UP-ColBPIP y WS-CDL, y un PNC de gestión de pronóstico de demanda. Para esto se definió una especificación de transformación de PNCs con los constructores compartidos por ambos lenguajes. Solo un subconjunto de los constructores de ambos lenguajes pudieron ser formalizados con el mismo comportamiento, ya que hay constructores complejos de UP-ColBPIP como instancias múltiples, o sincronización avanzada que no son soportados por WS-CDL, y por lo tanto no pueden ser alineados con ningún constructor de dicho lenguaje. Como resultado del caso de estudio se obtuvo la generación de código WS-CDL a partir de un modelo de PNC definido con UP-ColBPIP. El código WS-CDL sólo contiene aspectos referidos al flujo de control.

Finalmente, es necesario destacar que el correcto funcionamiento del patrón de transformación propuesto depende de una correcta implementación del mismo. Esto es, si bien se probó, desde el punto de vista teórico, que es posible definir un patrón de transformación para generar modelos y especificaciones de PNCs alineados, la máquina de transformación que implemente a dicho patrón de transformación puede contener errores de programación o una incorrecta definición de los modelos formales de los constructores de los lenguajes utilizados, con lo cual el funcionamiento del patrón de transformación podría no ser correcto.

7 Implementación y Evaluación

En este capítulo se presenta la implementación y evaluación de los métodos propuestos en la tesis, y se describen las herramientas que dan soporte a la formalización, verificación, y generación de PNCs (Sección 7.1). Se presentan los resultados de la validación y evaluación de los métodos de verificación (Sección 7.2) y transformación (Sección 7.3) con dichas herramientas. Finalmente, se presentan las conclusiones (Sección 7.4).

7.1. Herramienta para la Formalización, Verificación, y Generación de PNCs

Para validar y evaluar los métodos propuestos en esta tesis basados en el formalismo de GI-Nets definido, se desarrolló una herramienta denominada *Global Interaction Nets Tool*[1] que permite:

- el modelado y generación automática de GI-Nets para formalizar constructores de lenguajes de PNCs y modelos de PNCs,
- la verificación de PNCs definidos con UP-ColBPIP usando el método de verificación basado en GI-Nets,
- y la generación automática de especificaciones WS-CDL de PNCs.

La herramienta *Global Interaction Nets Tool* y su código fuente pueden obtenerse en http://code.google.com/p/gi-nets.

La arquitectura de la herramienta (Figura **7.1**) se desarrolló en base a la Plataforma Eclipse (Eclipse 2013), debido a que ésta permite la reutilización de un amplio conjunto de aplicaciones existentes. Los componentes de esta herramienta son "plug-ins" de Eclipse. La herramienta utiliza el "plug-in" EMF (Eclipse Modeling Framework) (Steinberg et al. 2008) para persistir modelos formales en formato XMI y para permitir interoperabilidad entre distintas aplicaciones de la plataforma Eclipse.

[1] http://code.google.com/p/gi-nets

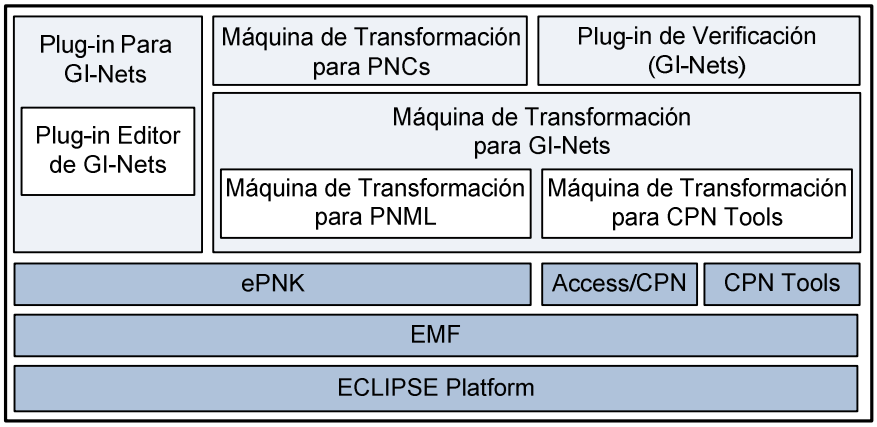


Fig. 7.1. Arquitectura de la herramienta Global Interaction Nets

ePNK[2] es un "framework" y "plug-in" que permite la definición de redes de Petri en el formato estándar PNML (Petri Net Markup Language) (PNML 2004) y provee una infraestructura para el desarrollo de editores gráficos para redes de Petri. La representación de GI-Nets con PNML permite integrar y utilizar diferentes herramientas de verificación de redes de Petri, debido a que es posible pasar modelos de GI-Nets en PNML a modelos GI-Nets en los formatos requeridos por distintas herramientas de verificación. Además, a partir de modelos GI-Nets en PNML es posible generar modelos o especificaciones de distintos lenguajes de PNCs. Esto permite que la máquina de transformación para PNCs sea independiente de cualquier herramienta de verificación de redes de Petri que se utilice.

En la herramienta desarrollada, se integró la herramienta de verificación de redes de Petri Access/CPN[3], la cual provee los mecanismos necesarios para manipular modelos de redes de Petri basados en CPN Tools (Jensen & Kristensen 2009). CPN Tools y Access/CPN dan soporte a la definición de modelos formales de redes de Petri Coloreadas con la modularidad requerida por las GI-Nets, y proveen un conjunto amplio de características que permiten la verificación de redes de Petri, y por lo tanto, de las GI-Nets. Además, CPN Tools facilita el modelado, verificación, y validación de modelos de redes de Petri gracias a las características de estructura jerárquica y modular, instancias de páginas, fusión de módulos y posiciones, expresiones en los arcos, etc, utilizadas en redes de Petri.

[2] ePNK. http://www2.imm.dtu.dk/~eki/projects/ePNK/
[3] Access/CPN. http://cpntools.org/accesscpn/

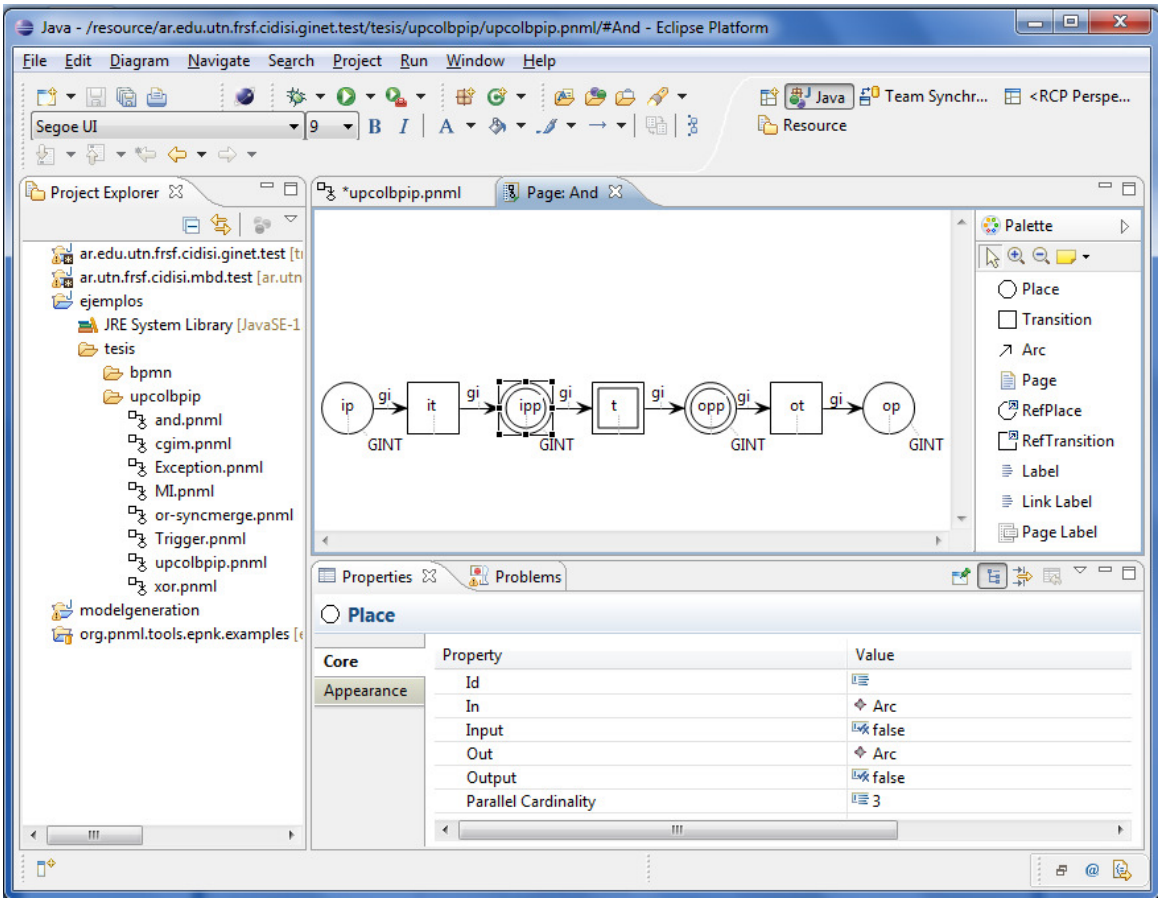


Fig. 7.2. Editor gráfico para GI-Nets

Los componentes de la herramienta que fueron desarrollados e implementados como "plug-ins" de Eclipse son:

- el *Editor gráfico para GI-Nets*
- la *máquina de transformación para GI-Nets*
- la *máquina de transformación para PNCs*
- y el *plug-in para verificación basada en GI-Nets*

El *Editor gráfico para GI-Nets* (Figura **7.2**) que soporta el modelado y persistencia de GI-Nets, está basado en el "framework" ePNK. Este editor permite la formalización de constructores de lenguajes de PNC por medio de la definición de módulos GI abstractos. Soporta la generación automática de módulos GI concretos a partir de módulos abstractos. Esto facilita la validación de los módulos abstractos para determinar si están correctamente definidos y que representan el comportamiento del constructor al que formalizan.

El editor (Figura **7.2**) está compuesto de: una *vista de proyecto*, que permite manipular los archivos PNML que contienen las GI-Nets o módulos GI abstractos; una *vista de edición de GI-Nets*, en el cual se pueden definir y modelar los módulos GI abstractos que formalizan los constructores de un lenguaje de PNC, y desarrollar módulos GI concretos (a partir de los abstractos) y GI-Nets que formalizan PNCs; una *barra de herramientas*, que se utiliza para crear una nueva posición (Place), transición (Transition), o arco (Arc); y una *vista de propiedades*, que permite editar las propiedades de una GI-Net, como el nombre de las posiciones y transiciones. Como ejemplo, la vista de proyecto en la carpeta *upcolbpip* contiene algunos de los módulos GI abstractos definidos que formalizan los constructores del lenguaje UP-ColBPIP. La vista de edición muestra una pestaña en la cual se observa el módulo GI abstracto definido para el constructor *And* de UP-ColBPIP. La vista de propiedades muestra los valores de las propiedades de la transición *ipp* seleccionada en la vista de edición.

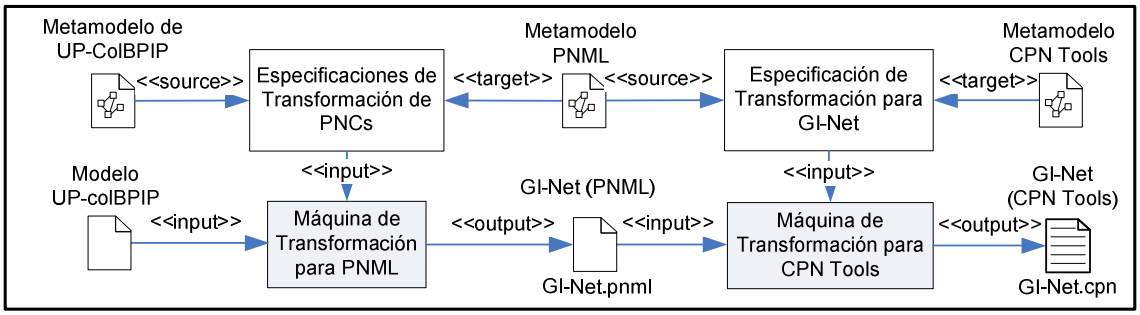


Fig. 7.3. Máquina de transformación para GI-Nets

La *máquina de transformación para GI-Nets* (Figura **7.3**) implementa el patrón de transformaciones para GI-Nets propuesto en el Capítulo 3, y permite la generación de GI-Nets a partir de distintos modelos o especificaciones de PNCs. Esta máquina se divide en una máquina de transformación para PNML y una máquina de transformación para CPN Tools. La entrada de esta máquina de transformación es un modelo de PNC definido con UP-ColBPIP y la especificación de transformación para GI-Nets. La salida puede ser una GI-Net basada en PNML o en CPN Tools, según lo defina el usuario.

El proceso de transformación permite la generación de una GI-Net en formato PNML, y luego a partir de la misma se genera una GI-Net en formato CPN Tools.

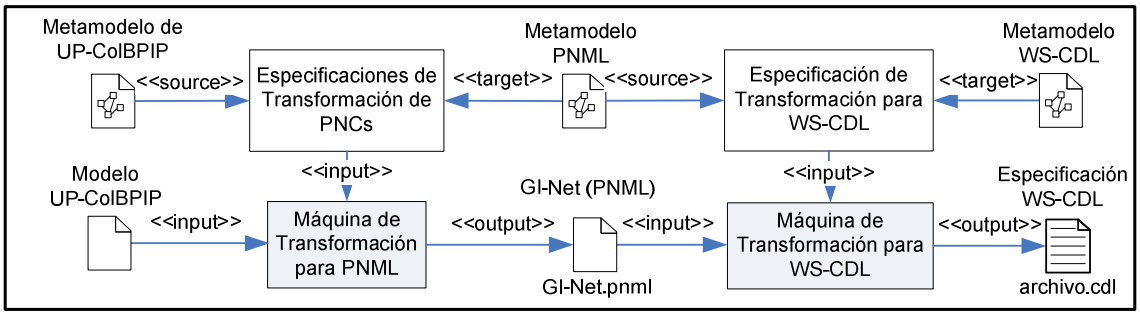


Fig. 7.4. Máquina de transformación para PNCs

La *máquina de transformación para PNCs* (Figura **7.4**) implementa el patrón de transformación para PNCs propuesto en el Capítulo 5, y permite la generación de especificaciones de PNCs a partir de modelos, cuyos comportamientos están alineados. Esta máquina está basada en la máquina de transformación para GI-Nets, ya que reutiliza la generación de GI-Nets en formato PNML como modelos formales intermedios de la transformación. La entrada de esta máquina de transformación es un modelo de PNC definido con UP-ColBPIP. La salida es una especificación de PNC en formato de archivo XML basado en el lenguaje WS-CDL.

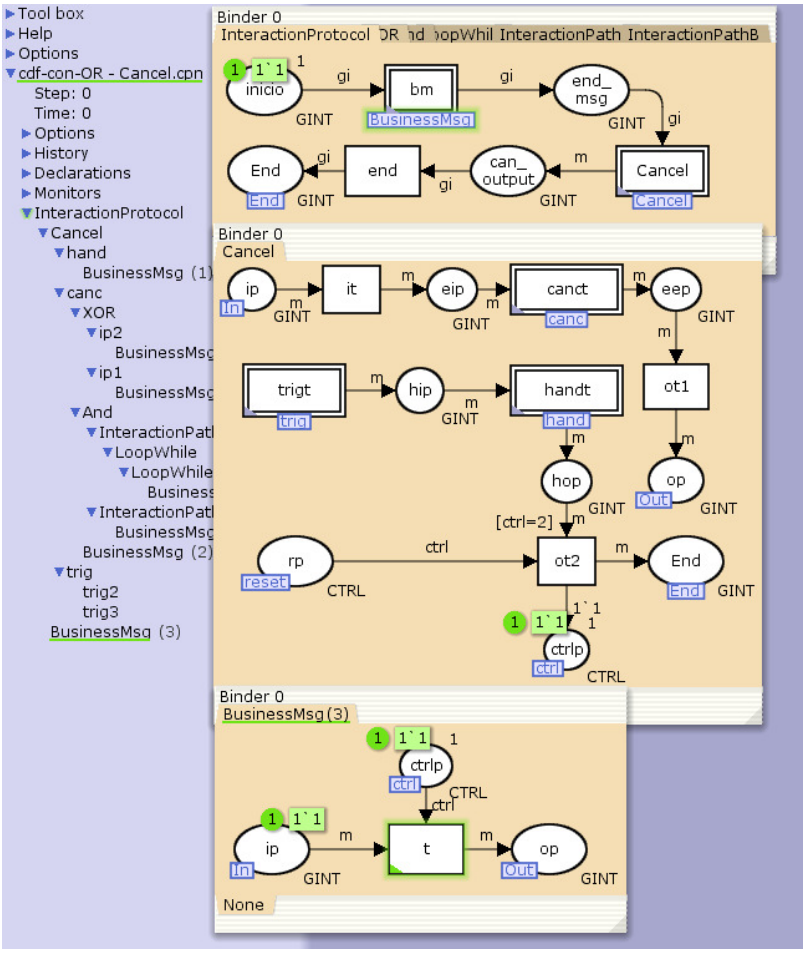


Fig. 7.5. Herramienta CPN Tools

El *"plug-in" para verificación basado en GI-Nets* hace uso del "framework" Access/CPN y de CPN Tools, y de la máquina de transformación para GI-Nets. De esta manera, en base a un modelo de PNC de entrada, genera una GI-Net en formato CPN Tools que formaliza dicho PNC. La Figura **7.5** muestra parte de una GI-Net definida en CPN tools. En la izquierda de la figura se muestra la estructura en forma de árbol de la misma con todos los módulos GI concretos que la conforman. En la derecha se muestran tres módulos concretos: un protocolo de interacción (arriba), un Cancel (en el medio), y un mensaje de negocio (abajo). A través de Access/CPN en conjunto con CPN Tools es posible ejecutar el análisis o verificación de la red de Petri que representa la GI-Net generada, y determinar la propiedad solidez de interacción global de la GI-Net.

Por otra parte, se desarrolló otra herramienta que implementa el método de verificación basado en anti-patrones de comportamiento, la cual es un "plug-in" de Eclipse que extiende y agrega funcionalidades de verificación a la herramienta[4] *editor UP-ColBPIP*, también implementada como "plug-in" de Eclipse. La misma y su código fuente se encuentran disponibles en: https://code.google.com/p/upcolbpip-verification/.

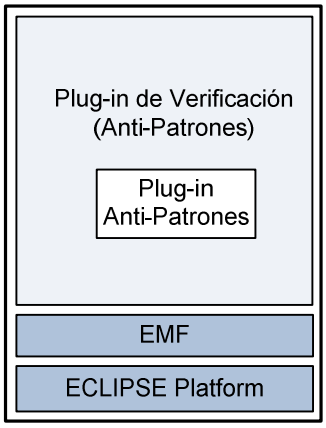


Fig. 7.6. Arquitectura del "plug-in" para verificación de PNCs basada en Anti-Patrones

La Figura **7.6** muestra la arquitectura del *"plug-in" para verificación basada en Anti-Patrones*. Éste utiliza el "framework" EMF para implementar los anti-patrones de comportamiento que son utilizados para verificar PNCs definidos con UP-ColBPIP. La regla que define a cada anti-patrón se implementó con el lenguaje Java, y es básicamente una consulta que se ejecuta sobre el modelo EMF que representa al PNC a verificar. La máquina retorna los elementos que causan problemas en el comportamiento del PNC. La Figura **7.7** muestra la detección de un error en un PNC no sólido, donde el problema se indica en la vista de problemas de Eclipse (parte inferior de la figura).

[4] https://code.google.com/p/upcolbpip-verification

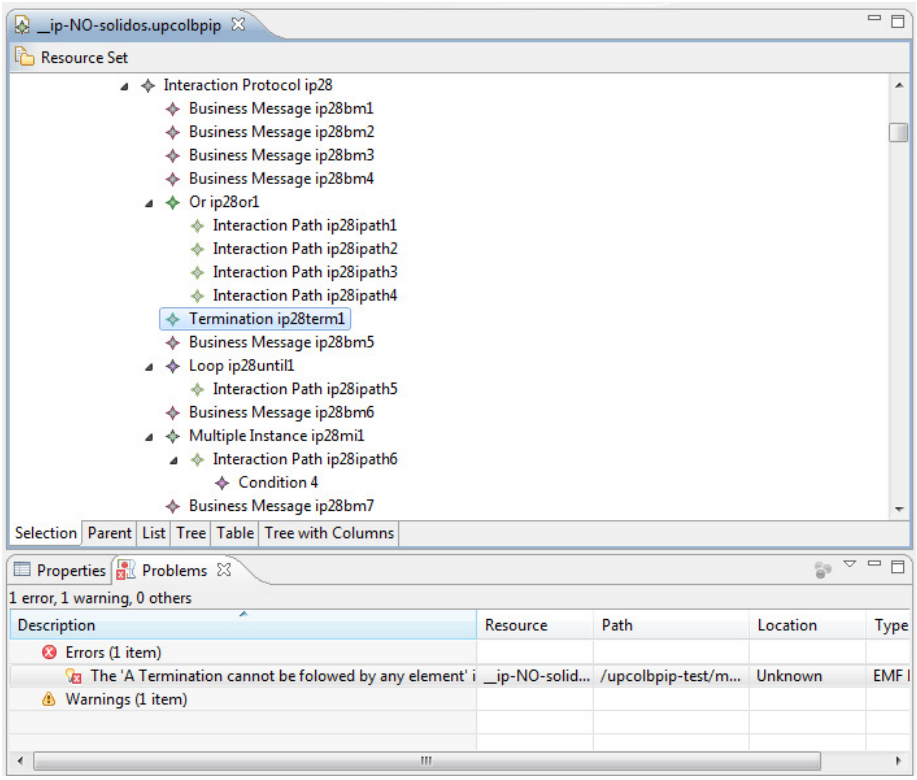


Fig. 7.7. Detección de error en el editor UP-ColBPIP

7.2. Evaluación de los Métodos de Verificación de PNCs

En esta sección se lleva a cabo una evaluación de los métodos de verificación propuestos en esta tesis. Los objetivos de esta evaluación son:

- Validar los métodos de verificación, de manera tal de determinar si los resultados devueltos por los métodos de verificación son los esperados.
- Comparar los resultados obtenidos en ambos métodos de verificación propuestos en esta tesis, de manera tal de determinar qué método es más adecuado en cuanto al rendimiento y completitud de los resultados obtenidos.

Para llevar a cabo la evaluación se definieron experimentos que consistieron en verificar dos librerías de modelos de PNCs definidos en UP-ColBPIP. Una librería consiste de 100 modelos que fueron generados manualmente y la otra de 880 modelos que fueron

generados de manera automática y sistemática a través de un algoritmo implementado a partir de criterios que permiten cubrir un amplio conjunto de casos de prueba. Los experimentos consistieron en verificar los modelos definidos en estas librerías con los métodos de verificación basados en GI-Nets y en anti-patrones de comportamiento. Los experimentos fueron ejecutados en una notebook con procesador Intel Core 2 Duo de 2.8 GHz y 8 GB de RAM. A continuación se detallan los criterios seleccionados para la generación de las librerías de modelos de PNCs, y luego, se muestran los resultados obtenidos para cada método de verificación.

Debido a que existen pocos métodos para verificación de PNCs implementados por otros autores y los propuestos en (Van der Aalst 1998; Diaz et al. 2005; Stuit & Szirbik 2009; Norta & Eshuis 2010) no proveen una herramienta accesible para su utilización, no ha sido posible comparar dichas métodos y herramientas con los propuestos en esta tesis.

7.2.1. Selección de Datos Empíricos

Para llevar a cabo los experimentos se generaron dos librerías de modelos de PNCs definidos con el lenguaje UP-ColBPIP.

Por un lado, en la librería A se generaron de manera manual 100 modelos de PNCs en UP-ColBPIP en base a diversas fuentes obtenidas a partir de casos de estudio y librerías de procesos encontrados en sitios web de organizaciones y repositorios de procesos. Estos modelos fueron definidos por becarios de grado de la carrera Ingeniería en Sistemas de Información de la Facultad Regional Santa Fe de la Universidad Tecnológica Nacional.

En la generación se consideraron solamente aspectos del flujo de control e intercambio de mensajes en PNCs y se dejaron de lado otros aspectos que tienen que ver con la semántica de los mensajes e información intercambiada entre las organizaciones que participan en los PNCs. Los modelos generados en esta librería están conformados por los constructores *And*, *Xor*, *Loop-While, Loop-Until*, Termination, *Or/SyncMerge*, *Cancel*, *Exception* y *Multiple Instances*.

La Figura **7.8** muestra información respecto de la cantidad de elementos y caminos de interacción que hay en los modelos de la librería A. Se puede observar que los modelos tienen entre 1 y 120 elementos, mientras que la cantidad de caminos de interacción por

modelo en los elementos de flujo de control varía entre 0 y 21. No se generaron modelos que superen los 80 caminos de interacción.

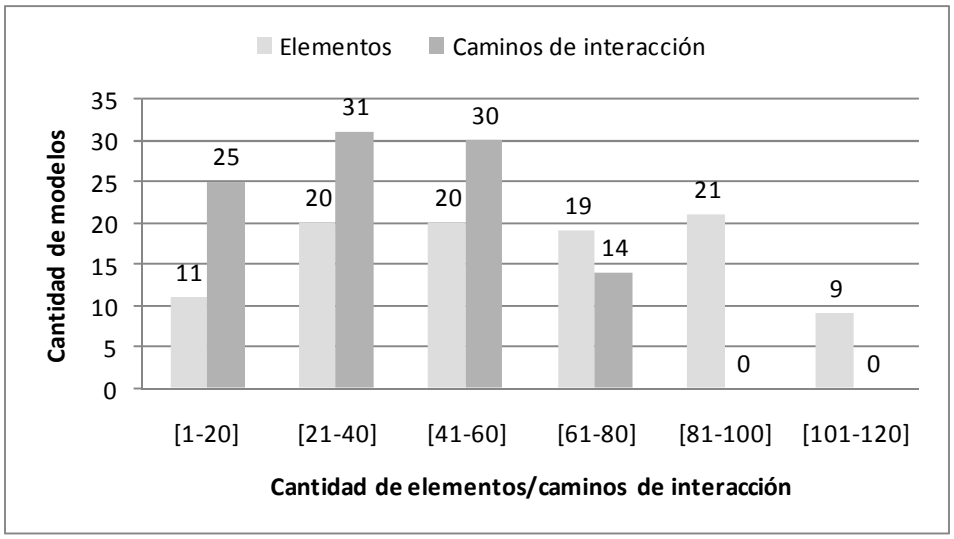


Fig. 7.8. Información de los modelos de la librería A

La Figura **7.9** muestra información respecto de la cantidad de constructores que pertenecen a los modelos de la librería A. Se puede observar que la mayor cantidad de modelos generados contienen entre 1 y 5 constructores de un mismo tipo. Además, hay entre 10 y 20 modelos que no están compuestos de alguno de los constructores en estudio. También se puede observar que hay entre 1 y 10 modelos que tienen entre 6 y 10 constructores utilizados.

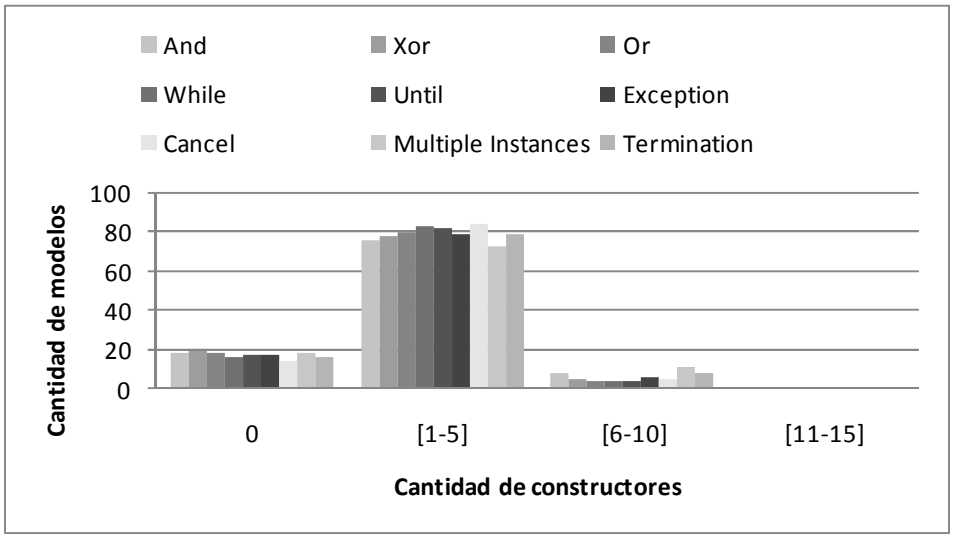


Fig. 7.9. Cantidad constructores por modelo de la librería A

Por otro lado, la librería B se generó de manera sistemática mediante un proceso automatizado implementado en un algoritmo en Java. En dicho proceso se consideraron criterios de generación de modelos de PNCs inferidos a partir del análisis de los modelos de la librería A. Los modelos generados en esta librería están conformados por los constructores *And*, *Xor*, *Loop-While, Loop-Until*, Termination, *Or/SyncMerge*, *Cancel*, *Exception* y *Multiple Instances*.

Para generar los modelos se definieron reglas de construcción, las cuales permiten agregar elementos a los modelos de manera sistemática. Las reglas se definieron considerando todas las posibles combinaciones de constructores que pueden ser definidas a partir del metamodelo de UP-ColBPIP, de manera tal de cubrir todas las combinaciones sólidas y no sólidas de elementos.

La generación de la librería B se dividió en dos partes iguales que conformaron las librerías B1 y B2. En la librería B1, se generaron modelos que contienen combinaciones de elementos que generan PNCs sólidos, mientras que en la librería B2 los modelos generados contienen al menos una combinación no sólida de elementos. De esta manera, el 50% de la librería B contiene modelos sólidos (pertenecientes a la librería B1), y el 50% restante contiene modelos no sólidos (pertenecientes a la librería B2). Esta separación explícita en la generación de modelos es útil para la validación de los métodos propuestos en esta tesis.

En total, se generaron 880 modelos en la librería B: 440 para la sublibrería B1 y 440 para la sublibrería B2. Los criterios para la generación de modelos de PNCs inferidos a partir del análisis de los modelos de la librería A son los siguientes:

- Cantidad de elementos por secuencia: entre 1 y 5
- Cantidad de caminos alternativos/paralelos: entre 2 y 5
- Cantidad de manejadores de excepciones: entre 1 y 3
- Tamaño de los modelos:
    - Cantidad máxima de elementos del PNC: entre 5 y 120
    - Nivel de anidamiento: entre 3 y 8

Una secuencia puede estar compuesta de 1 a 5 elementos. Los elementos con paralelismo o exclusión mutua pueden estar compuestos de 2 a 5 caminos de interacción. Los elementos de excepciones y cancelaciones pueden tener entre 1 y 3 manejadores de excepciones. La cantidad total de elementos de un modelo puede variar entre 5 a 120 elementos. Cada PNC puede tener desde 3 a 8 niveles de anidamiento en la estructura de árbol. Por último, cada elemento tiene una determinada probabilidad de aparición, la cual se define en base a los casos de prueba que se desean realizar. Las librerías generadas están disponibles en https://code.google.com/p/upcolbpip-verification/.

La Figura **7.10** muestra información respecto de la cantidad de elementos y caminos de interacción que hay en los modelos de la librería B. Se puede observar que los modelos tienen entre 1 y 120 elementos, mientras que la cantidad de caminos de interacción por modelo varía entre 0 y 80. No se generaron modelos que superen los 100 caminos de interacción.

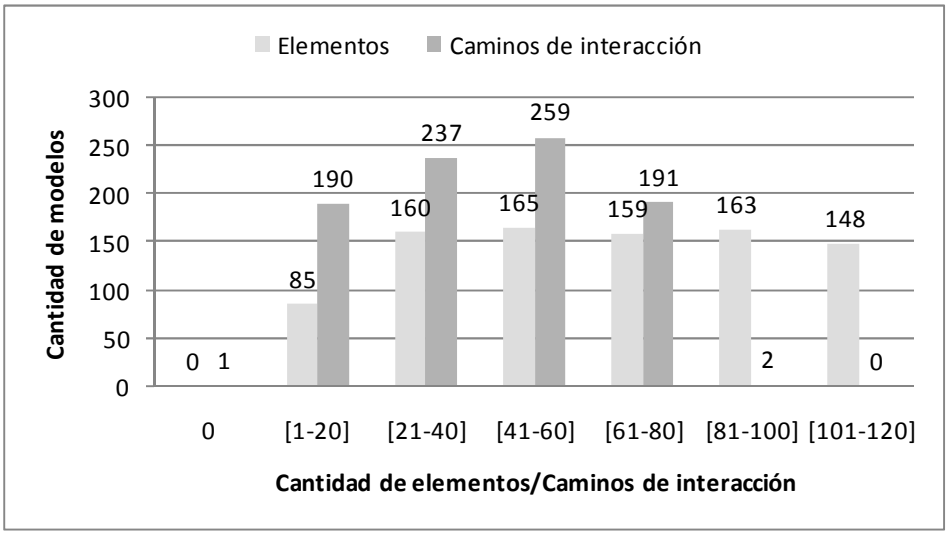


Fig. 7.10. Información de los modelos de la librería B

La Figura **7.11** muestra información respecto de la cantidad de constructores que pertenecen a los modelos de la librería B1. Se puede observar que la mayor cantidad de modelos generados contiene entre 1 y 5 constructores de un mismo tipo. Es importante notar que existe un alto número de modelos que no contiene constructores *Termination*. Esto se debe a que este constructor es el principal causante de bloqueos y fallos de sincronización en los modelos de PNCs, y como la librería B1 contiene solamente modelos sólidos, es normal que disminuya la cantidad de constructores *Termination*. Además del *Termination*, hay entre 55 y 97 modelos que no están compuestos de alguno de los demás constructores.

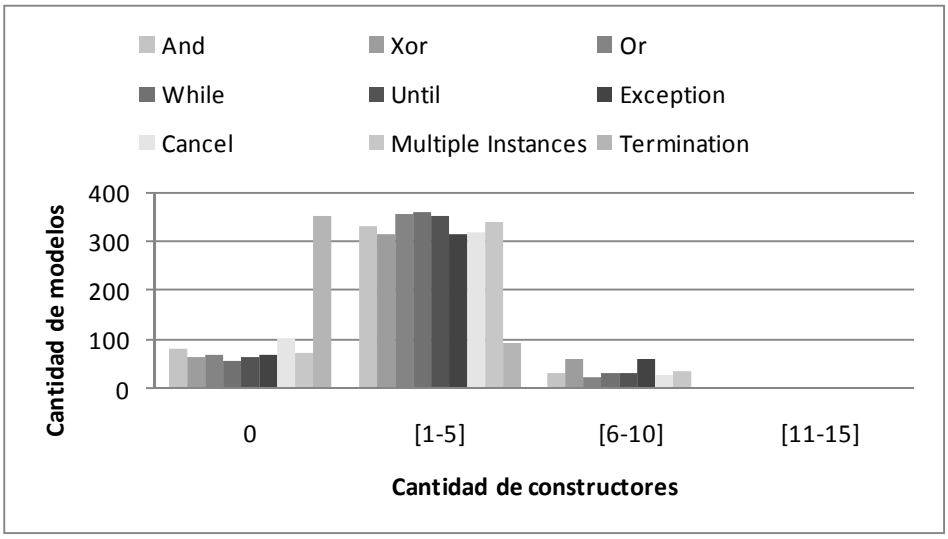


Fig. 7.11. Cantidad constructores por modelo de la librería B1

La Figura **7.12** muestra información respecto de la cantidad de constructores que pertenecen a los modelos de la librería B2. De manera similar a la librería B1, la mayor cantidad de modelos generados contiene entre 1 y 5 constructores de un mismo tipo. En este caso, el número de modelos que no contiene constructores *Termination* es muy bajo, y crece la cantidad de modelos que tienen entre 1 y 5 constructores de este tipo. Esto es esperable ya que la librería B2 contiene solamente modelos no sólidos.

La relación entre cantidad de elementos y caminos de interacción, así como la cantidad de elementos por modelo que aparecen en los modelos generados en la librería B, se ajusta de manera proporcional a las relaciones que existen en los modelos definidos manualmente de la librería A. Por este motivo, se considera que la librería B representa un conjunto de casos de prueba adecuado para llevar a cabo la evaluación de los métodos propuestos en esta tesis.

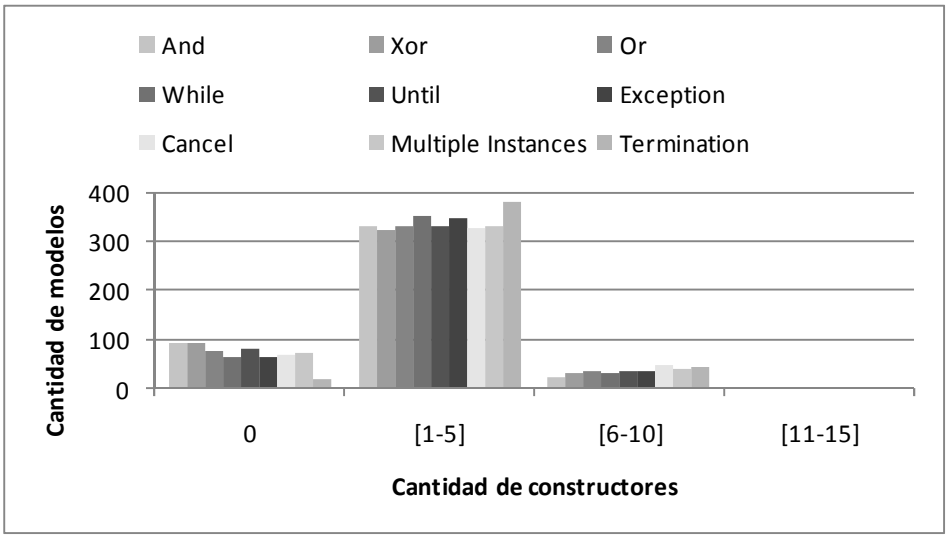


Fig. 7.12. Cantidad constructores por modelo de la librería B2

7.2.2. Verificación de PNCs

Se definieron dos experimentos en los cuales se utilizaron respectivamente los métodos de verificación basados en GI-Nets y en anti-patrones de comportamiento propuestos en los Capítulos 4 y 5. En los experimentos se utilizaron las librerías A y B. Los tiempos de análisis se definieron a partir del promedio de cinco ejecuciones de cada experimento. Se considera que un modelo de PNC no es *tratable* si su tiempo de verificación excede los 5000 milisegundos (ms).

Experimento 1: Verificación basada en GI-Nets

En el primer experimento se utilizó el método de verificación basado en GI-Nets, y se obtuvieron resultados experimentales. Para validar el método se tomaron muestras de ambas librerías y se compararon los resultados experimentales con los resultados esperados para dichas muestras. Se considera que existe una inconsistencia si los resultados obtenidos difieren de los resultados esperados.

Los resultados experimentales con ambas librerías A y B coinciden en un 100% con los resultados esperados, con lo cual no se confirmó la existencia de inconsistencias generadas a partir de un error en la definición/implementación del método de verificación.

Se determinó que el 42% de los modelos de la librería A son no sólidos y el 58% restante son sólidos. Los resultados fueron corroborados manualmente mediante una inspección de cada uno de los modelos. Además, se determinó que el 100% de los modelos de la librería B1 son sólidos, y el 100% de los modelos de B2 son no sólidos. Esto es consistente con el criterio de generación de dichas librerías descripto en la Sección 7.2.1.

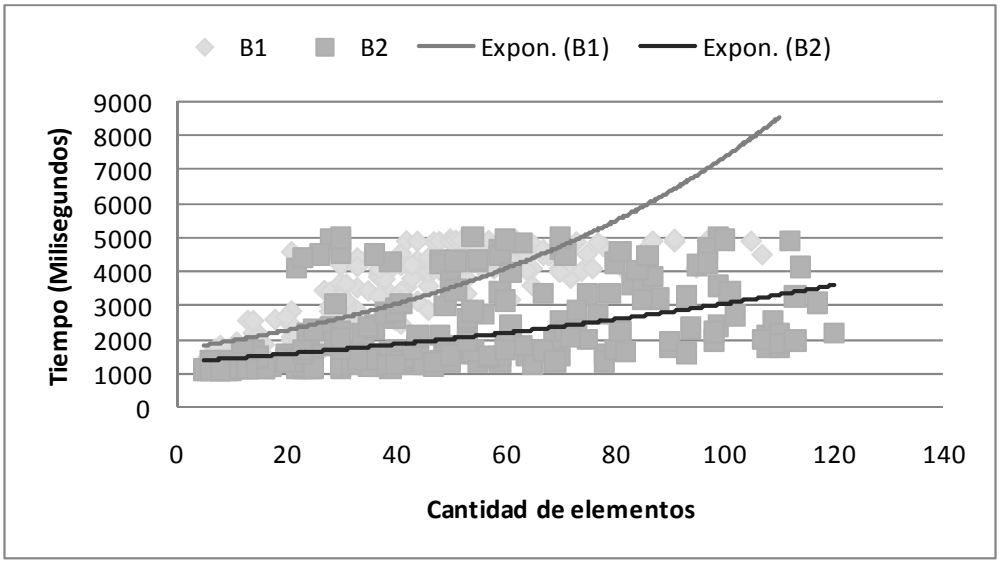


Fig. 7.13. Tiempo de verificación con GI-Nets por cantidad de elementos para la librería B1

La figura **7.13** muestra la relación entre el tiempo de verificación con el método basado en GI-Nets y la cantidad de elementos que tienen los modelos de PNCs de las librerías B1 y B2. Se puede observar que el tiempo de detección de la solidez de los modelos de la librería B1 crece más rápidamente que los de la librería B2 a medida que crece la cantidad de elementos de los modelos. Esto se debe a que para determinar la solidez de un modelo es necesario explorar en forma completa el espacio de estados de la red de Petri que representa una GI-Net. Si durante la exploración se detecta un bloqueo o fallo de sincronización se finaliza el proceso de exploración y se concluye que el modelo no es sólido, sin llegar a la exploración completa. Por lo tanto, la diferencia en los tiempos de respuesta de los modelos de la librería B1 y B2 surge a partir de que en B1 todos los modelos son sólidos y en B2 son no sólidos.

En total se registraron 458 modelos como no tratables, considerando a todos los modelos de la librería B. Es decir que para el 52% de los modelos de B el tiempo de

verificación fue superior a 5000 ms. El tiempo promedio de verificación para los modelos tratables es de 2807 ms. Se observa que para modelos con una cantidad de elementos superior a 50, el tiempo de verificación se incrementa rápidamente.

Experimento 2: Verificación basada en anti-patrones de comportamiento

En el segundo experimento se utilizó el método de verificación basado en anti-patrones. Los anti-patrones utilizados se muestran en la Figura **7.14**. Los algoritmos que implementan estos anti-patrones se encuentran en el Anexo B.

En el experimento se determinó que el 100% de los modelos de la librería B1 son sólidos, y el 100% de los modelos de la librería B2 son no sólidos. Esto es consistente con el criterio de generación de dichas librerías descripto en la Sección 7.2.1.

Para validar este método se compararon los resultados obtenidos con los resultados del método de verificación basado en GI-Nets en el experimento 1. Se consideró que en la comparación de resultados podían surgir inconsistencias por dos motivos: (1) si el conjunto de anti-patrones no es completo, podrían existir combinaciones de constructores que lleven a modelos de PNCs no sólidos y que el algoritmo no los detecte; y (2) si existe algún anti-patrón que no está definido/implementado correctamente. Los resultados experimentales coinciden en un 100% con los resultados del método de verificación basado en GI-Nets obtenidos en el experimento 1, con lo cual no se pudo confirmar la existencia de inconsistencias.

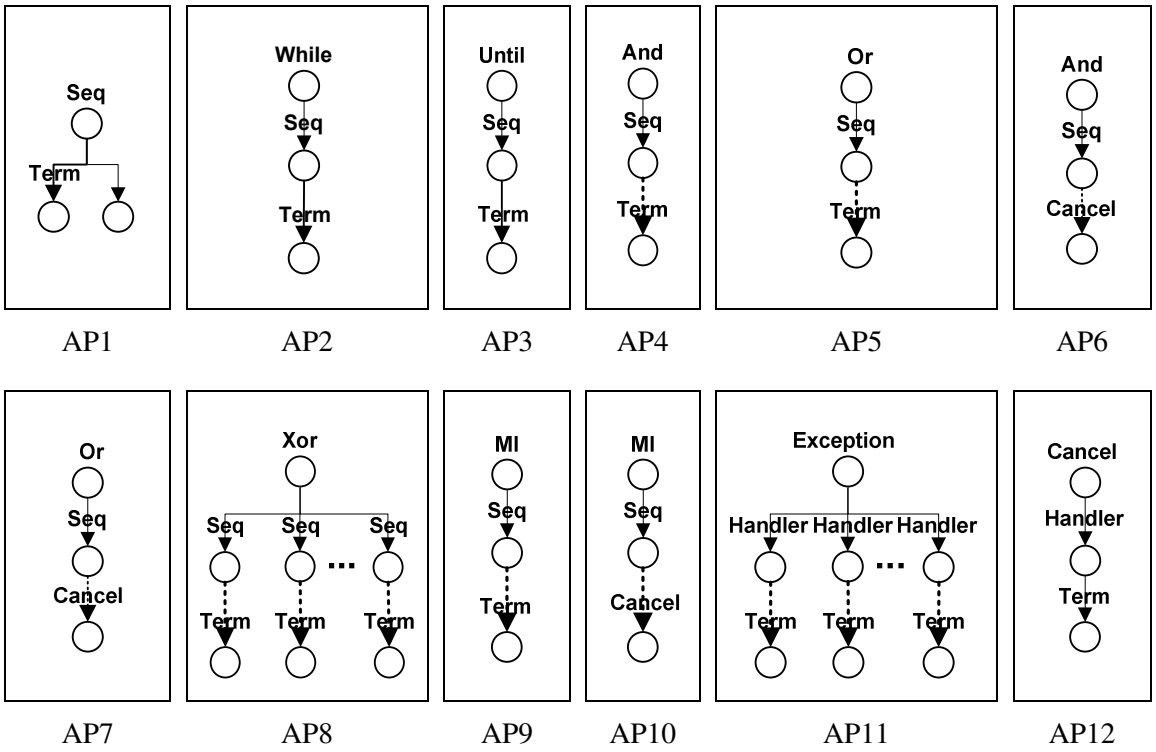


Fig. 7.14. Anti-patrones de comportamiento del lenguaje UP-ColBPIP

Además, para validar los resultados se tomaron muestras de manera aleatoria de ambas librerías y se compararon los resultados obtenidos en el experimento con los resultados esperados para dichas muestras. Se obtuvo un 100% de coincidencia entre los resultados obtenidos y los esperados, con lo cual tampoco se detectaron inconsistencias.

Por lo tanto, se pudo concluir que el método de verificación basado en anti-patrones provee los mismos resultados que el método basado en GI-Nets.

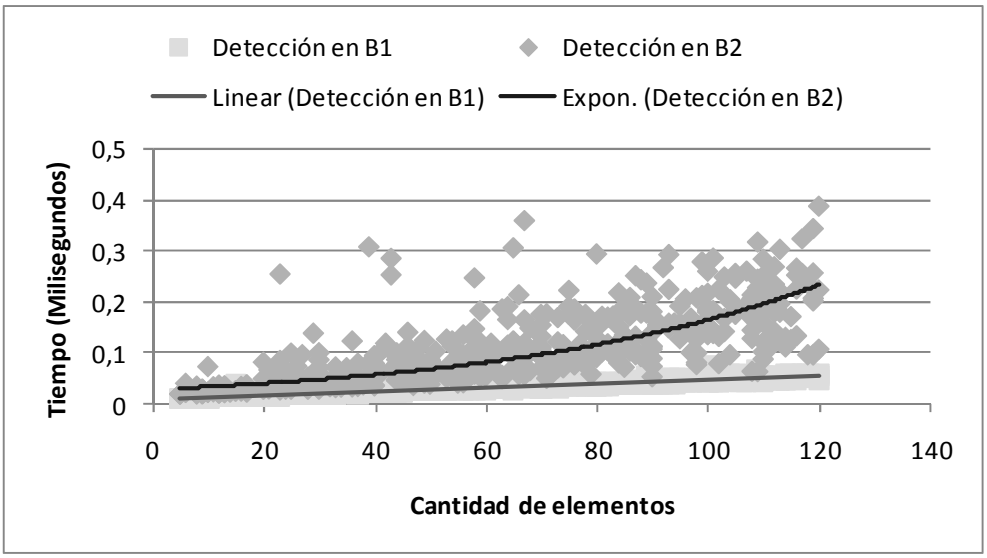


Fig. 7.15. Tiempo de detección de anti-patrones por cantidad de elementos para las librerías B1 y B2

La Figura **7.15** muestra la relación entre el tiempo de detección de los anti-patrones y la cantidad de elementos que tienen los modelos de PNCs de las librerías B1 y B2. Se puede observar que el tiempo de detección de anti-patrones en la librería B1 es lineal a la cantidad de elementos, en cambio, en la librería B2 los tiempos de detección crecen más rápidamente a medida que crece la cantidad de elementos de los modelos. Esto se debe a que los modelos de B2 contienen un alto número de elementos *Termination* (ver Figura **7.12**). Debido a que 9 de los 12 anti-patrones (Figura **7.14**) contienen al menos un elemento *Termination*, cuanto mayor sea la cantidad de elementos *Termination* que tiene un modelo, mayor es el tiempo que lleva la detección de los anti-patrones.

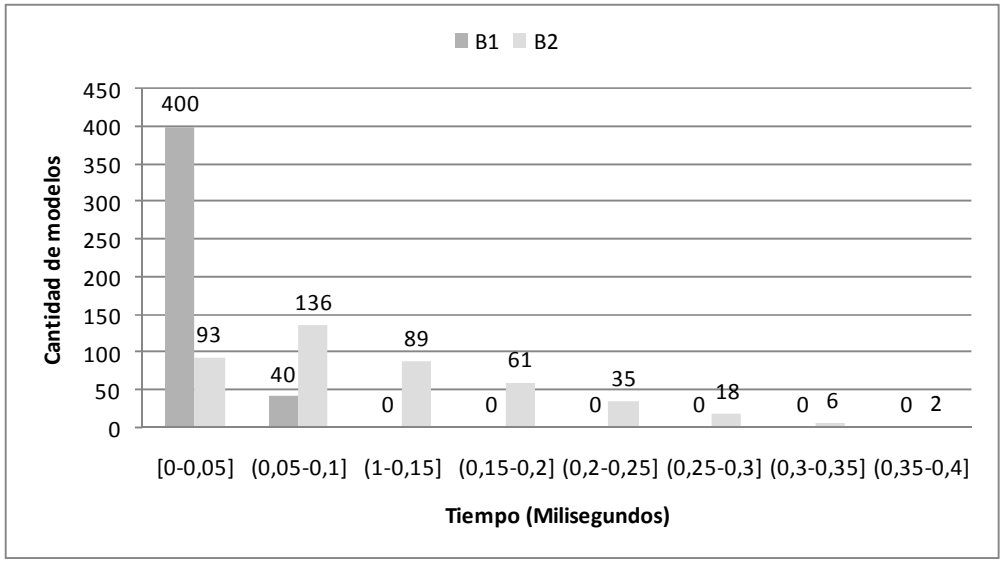


Fig. 7.16. Cantidad de modelos por tiempo de detección de anti-patrones para las librerías B1 y B2

La Figura **7.16** muestra la cantidad de modelos en los que se detectaron anti-patrones en un determinado tiempo de ejecución. Para el 90,9% de los modelos de la librería B1, el tiempo de detección de los anti-patrones resultó inferior a 0,05 ms. Para la librería B2, en el 86,1% de los modelos fue posible detectar los anti-patrones en menos de 0,2 ms.

Es necesario considerar que a diferencia del algoritmo de verificación basado en GI-Nets, el algoritmo de detección de anti-patrones detecta todos los anti-patrones que son parte de un modelo, con lo cual el algoritmo revisa el modelo completo en busca de los anti-patrones. Si el algoritmo finalizara su ejecución al detectar el primer anti-patrón, el rendimiento del mismo se vería mejorado. No se realizaron experimentos considerando esta opción ya que los tiempos de respuesta que brinda el algoritmo al detectar todos los anti-patrones es, en el peor de los casos, inferior a medio milisegundo. Esto se cumple inclusive para modelos cuya cantidad de elementos varía entre 100 y 120, los cuales superan ampliamente el tamaño típico de modelos obtenidos a partir de casos reales (Mendling 2008).

Comparación de resultados

La Tabla **7.1** muestra los resultados obtenidos para los experimentos. Con el método basado en GI-Nets sólo el 48% de los modelos son tratables, es decir, pudieron ser verificados en menos de 5000 ms. En cambio, con el método basado en anti-patrones, el 100% de los modelos resultaron tratables. En cuanto a los tiempos de respuesta en la verificación de cada PNC, para el método basado en GI-Nets, los tiempos varían entre 1091 y 4493 ms, siendo el tiempo de respuesta promedio de 2807 ms. En cambio, con el método basado en anti-patrones los tiempos varían entre 0,009 y 0,386 ms, y el promedio es de 0,072 ms.

Tabla 7.1. Resultados experimentales de los métodos de verificación

		A	**B**
Modelos tratables con GI-Nets	Cantidad	51	422
	%	51%	48
Modelos tratables con Anti-patrones	Cantidad	100	880
	%	100	100
Tiempo de análisis con GI-Nets [ms] (modelos tratables)	Min.	998	1091
	Máx.	4995	4993
	Prom.	2753	2807
Tiempo de análisis con Anti-Patrones [ms]	Min.	0,009	0,009
	Máx.	0,389	0,386
	Prom.	0,096	0,072

El método basado en anti-patrones obtuvo mejores tiempos de respuesta que el método basado en GI-Nets. Esto se debe fundamentalmente a que el método basado en GI-Nets lleva a cabo una exploración del espacio de estados de la red de Petri que formaliza un PNC, mientras que el método basado en anti-patrones se enfoca simplemente en encontrar un conjunto de combinaciones de constructores dentro del PNC. Además, desde el punto de vista de implementación, el método basado en GI-Nets interactúa con una aplicación externa (CPN Tools), lo cual ocasiona demoras en los tiempos de respuesta. En cambio, el método basado en anti-patrones está implementado completamente en Java dentro de una misma aplicación.

7.3. Evaluación del Método de Transformación de PNCs

En esta sección se describe la evaluación del método de transformación para PNCs propuesto en el Capítulo 6, Sección 6.2, con el objetivo de determinar si el comportamiento de las especificaciones de PNCs generadas por dicho método está alineado con el comportamiento de sus respectivos modelos de PNCs.

Para llevar a cabo la validación se definió un experimento que consistió en utilizar un subconjunto de la librería A definida en la Sección 7.2.1 y se validó manualmente si el comportamiento de cada especificación WS-CDL generada es consistente con el comportamiento de su respectivo modelo UP-ColBPIP. El experimento fue realizado en un procesador Intel Core 2 Duo de 2.8 GHz y 8 GB de RAM.

En la selección de datos empíricos se utilizó un subconjunto de las librería A y B. Debido a que los lenguajes UP-ColBPIP y WS-CDL no contienen el mismo conjunto de constructores, los modelos utilizados no contienen los elementos *Multiple Instances*, *Or/SyncMerge*, ni *Exception*, ya que estos constructores no forman parte del lenguaje WS-CDL.

La validación de los resultados se llevó a cabo manualmente. Para esto se generaron manualmente las GI-Nets y las especificaciones WS-CDL correspondientes a los modelos UP-ColBPIP seleccionados para el experimento. Luego, se generaron las GI-Nets y las especificaciones WS-CDL de dichos modelos con la máquina de transformación para PNCs implementada en la herramienta. Para llevar a cabo la validación se inspeccionaron los modelos GI-Nets y WS-CDL generados automáticamente y se los comparó con sus respectivas versiones generadas manualmente. En los resultados experimentales no se detectaron inconsistencias, con lo cual el 100% de las especificaciones WS-CDL generadas tienen el mismo comportamiento que sus respectivos modelos UP-ColBPIP. Los modelos y las especificaciones generadas pueden ser obtenidos en el sitio https://code.google.com/p/upcolbpip-verification/.

7.4. Conclusiones

La herramienta *Global Interaction Nets,* que implementa los métodos de formalización, verificación y generación de PNCs basados en el formalismo de GI-Nets, y la herramienta que implementa el método de verificación basado en anti-patrones, desarrolladas e implementadas sobre la plataforma Eclipse, están compuestas de "plug-ins" que ofrecen las funcionalidades necesarias para llevar a cabo la formalización, verificación, y transformación de PNCs definidos con el lenguaje UP-ColBPIP.

Estas herramientas permitieron validar y evaluar los métodos de verificación y transformación propuestos utilizando dos librerías que contienen en total 980 modelos de PNCs definidos con UP-ColBPIP. Dichas librerías, una generada manualmente a partir de casos de estudio y librerías de procesos y la otra librería generada automáticamente a partir de un conjunto de criterios de generación definidos a partir de los modelos definidos manualmente, permitieron obtener un amplio conjunto de modelos de PNCs que pudieron ser utilizados para validar y evaluar los métodos propuestos en esta tesis.

Con respecto a la completitud del método de verificación basado en GI-Nets, en los experimentos realizados se pudieron verificar todos los modelos definidos en UP-ColBPIP de las librerías en estudio. Sin embargo, el rendimiento de dicho método decrece rápidamente cuando se incrementa la cantidad de elementos en los modelos, y solamente el 48% de los mismos pudo ser verificado en menos de 5000 ms. Estos resultados podrían mejorase si se utilizaran técnicas de reducción de modelos (Fahland et al. 2009) antes de llevar a cabo la verificación. Este tipo de técnicas ha sido utilizada con éxito para la verificación de procesos de negocio intra-organizacionales (Fahland et al. 2009). Sin embargo, estas técnicas tienen el inconveniente de que al reducir los modelos, se pierde precisión en la interpretación de resultados, respecto del lugar donde se encuentra el problema.

Con relación al método de verificación basado en anti-patrones, en los experimentos realizados no se confirmaron inconsistencias entre este método y el método basado en GI-Nets. Esto es un indicador de la completitud del mismo, ya que se pudieron verificar todos los modelos verificados con el método basado en GI-Nets, y se obtuvieron los mismos resultados de verificación. Los resultados experimentales muestran que el rendimiento del

método basado en anti-patrones supera ampliamente al del método basado en GI-Nets. Otra ventaja del uso de anti-patrones es que favorece la integración y reutilización del método de verificación en distintos lenguajes y etapas del modelado de PNCs, debido a que sólo se deben implementar un conjunto de algoritmos muy sencillos y no es necesario interactuar con ninguna aplicación externa.

8 Conclusiones y Trabajos Futuros

8.1. Principales Contribuciones

Las colaboraciones inter-organizacionales permiten a las organizaciones mejorar su competitividad y obtener mayores beneficios. Una colaboración inter-organizacional implica integrar las organizaciones mediante la definición y ejecución conjunta de procesos de negocio colaborativos (PNCs). El comportamiento de la colaboración y las interacciones entre las organizaciones es acordado y definido en el flujo de control de los modelos y especificaciones de PNCs. Las interacciones entre los sistemas de información inter-organizacionales se definen de acuerdo a este flujo de control. Por lo tanto, es importante que el flujo de control sea correctamente definido, tanto en los modelos de PNCs que conforman la solución de negocio como en las especificaciones de PNCs, y que el comportamiento de las especificaciones esté alineado con el de los modelos.

A partir de esta premisa, en esta tesis se abordaron dos problemas principales: el de verificación del comportamiento de los modelos de PNCs; y el de generación de especificaciones de PNCs a partir de modelos, tal que el comportamiento de la especificación generada esté alineado con el comportamiento definido en el modelo. Los enfoques de verificación de PNCs existentes no permiten la detección de bloqueos y estados no alcanzables cuando se utilizan constructores de flujos de control complejos, tales como sincronización avanzada de caminos en paralelo o manejo de excepciones. Por otra parte, si bien existen algunos enfoques para determinar alineación de comportamiento entre procesos, los mismos no se enfocan en generar una especificación de PNC a partir de un modelo, ni consideran la verificación de los mismos.

De acuerdo a los problemas identificados, se definió como hipótesis principal que mediante la aplicación de enfoques de formalización y verificación de modelos estructurados de PNCs, junto con la aplicación de los principios del desarrollo dirigido por modelos, es posible definir un modelo de PNC que satisfaga un conjunto de propiedades de comportamiento y generar, a partir del mismo, una especificación cuyo comportamiento esté alineado con el del modelo.

En respuesta a esta hipótesis, en esta tesis se propusieron: el lenguaje formal *redes de Interacción Global (GI-Nets)*, el *método de verificación de PNCs basado en GI-Nets*, el *método de verificación de PNCs basado en anti-patrones*, el *método de transformación de PNCs con alineación de comportamiento* y *herramientas* que implementan estas propuestas. De esta manera, la tesis ha realizado contribuciones en las siguientes áreas de la ingeniería en sistemas de información y de software: diseño de procesos de negocio, formalización de procesos, verificación de procesos y desarrollo dirigido por modelos.

A continuación se detallan cada una de las principales contribuciones de la tesis.

8.1.1. Formalización de PNCs con el lenguaje GI-Nets

Para dar soporte a la formalización de PNCs se propuso el lenguaje formal *redes de Interacción Global (GI-Nets)*, y el *patrón de transformación de modelos para GI-Nets*.

El lenguaje GI-Nets permite definir modelos formales estructurados de PNCs. Está basado en el formalismo de redes de Petri jerárquicas y coloreadas, y es un nuevo tipo de red de Petri. Los principales beneficios de usar el lenguaje GI-Nets para formalizar modelos de PNCs son:

- permite formalizar de manera separada los constructores de un lenguaje de modelado/especificación de PNCs, de los elementos que conforman un modelo/especificación de PNC con dicho lenguaje. A partir de los módulos GI abstractos que formalizan los constructores del lenguaje, es posible formalizar un modelo definido en dicho lenguaje. Para dicho modelo se define una GI-Net que contiene módulos GI concretos que representan los elementos del modelo. Estos módulos GI concretos se definen a partir de los módulos GI abstractos, según los constructores del lenguaje usados en los elementos del modelo.
- es independiente de la semántica de los lenguajes de modelado/especificación de PNCs, lo cual lo hace flexible y adaptable a ser utilizado para formalizar cualquier lenguaje de PNCs. Puede ser usado para formalizar PNCs definidos con diferentes lenguajes y usados en diferentes etapas del desarrollo dirigido por modelos de sistemas inter-organizacionales. De esta manera, no se requiere usar diferentes enfoques de formalización en las distintas etapas del proceso de desarrollo. El uso de módulos para estructurar una GI-Net junto con el uso de

diferentes tipos de señales para representar interacciones y flujos de control posibilita esta independencia.

- permite la formalización de constructores complejos para sincronización avanzada, manejo de excepciones y cancelaciones, e instancias múltiples. Esto también se logra con el uso de módulos para estructurar la red junto con el uso de diferentes tipos de señales para representar interacciones y flujos de control.

Como limitaciones del lenguaje se puede indicar que el mismo sólo es aplicable a modelos o especificaciones estructurados de PNCs. No obstante, se podrían formalizar modelos no estructurados de PNCs con GI-Nets, aplicando previamente a ellos métodos existentes de estructuración de procesos (Vanhatalo, Völzer & Koehler 2009) que generan un modelo estructurado. Además, como se destacó en esta tesis (Capítulo 2.3), el uso de modelos estructurados provee mayores beneficios que los no estructurados tanto para el diseño como para la verificación de PNCs.

Para dar soporte a la generación de GI-Nets a partir de un modelo de PNCs, se definió un patrón formal de transformación para GI-Nets, siguiendo los principios del desarrollo dirigido por modelos. El patrón está definido de manera tal que pueda ser aplicado para la formalización de modelos o especificaciones de PNCs definidas con cualquier lenguaje de PNCs, haciendo uso de la formalización del lenguaje con GI-Nets.

Se demostró la aplicación de la formalización de modelos de PNCs con GI-Nets a los lenguajes de modelado UP-ColBPIP y al lenguaje estándar de facto BPMN.

8.1.2. Método de Verificación de PNCs basado en GI-Nets

Este método de verificación de comportamiento de modelos de PNCs utiliza el lenguaje GI-Nets y el patrón de transformaciones para GI-Nets. Esto implica que un modelo de PNC primero es transformado en una GI-Net para luego verificar si la misma cumple ciertas propiedades de comportamiento. Para esto, se definió la propiedad *Solidez de Interacción Global*, que permite la detección de bloqueos en el comportamiento de modelos de PNCs, inclusive de aquellos modelos que contienen constructores complejos, tales como sincronización avanzada, excepciones, cancelaciones, e instancias múltiples. La principal característica de esta propiedad es que permite detectar estados inválidos de un modelo de PNC, permitiendo discriminar entre elementos del modelo que representan

aspectos del flujo de control, de aquellos que representan interacciones entre organizaciones. Los principales beneficios de este método son:

- permite verificar modelos de PNCs definidos con cualquier lenguaje. Debido a que la verificación se basa en GI-Nets, el método es independiente de los lenguajes de modelado o especificación de PNCs, lo cual lo hace flexible y adaptable a distintos lenguajes y, por lo tanto, puede ser usado para verificar modelos de PNCs en diferentes etapas de un proceso de desarrollo dirigido por modelos;
- permite verificar modelos conceptuales de PNCs, posibilitando la verificación en etapas tempranas del desarrollo, cuando las principales decisiones del comportamiento de las colaboraciones inter-organizacionales son tomadas;
- permite verificar modelos de PNCs con flujos de control complejos. Esto impacta positivamente en el criterio de completitud, ya que soporta una mayor cantidad de constructores de flujo de control que otros métodos de verificación existentes;
- permite determinar el lugar específico donde ocurre un error en un modelo de PNC, ya que el lugar donde se detecta un error corresponde a un módulo GI concreto el cual está asociado a un elemento del modelo. Esto posibilita mejorar la interpretación de los resultados de verificación;
- permite llevar adelante la verificación automática de modelos de PNCs. El uso de patrones de transformación de modelos posibilita generar automáticamente el modelo formal GI-Net y realizar la verificación automáticamente utilizando la herramienta de redes de Petri CPN Tools.

Se demostró la aplicación del método de verificación con modelos de PNCs definidos con los lenguajes de modelado UP-ColBPIP y BPMN.

Como limitaciones del método de verificación basado en GI-Nets se puede indicar que su rendimiento decrece rápidamente cuando se incrementa la cantidad de elementos en los modelos. Estos resultados podrían mejorarse si se utilizaran técnicas de reducción de modelos (Fahland et al. 2009) antes de llevar a cabo la verificación. Sin embargo, con las mismas se pierde precisión respecto del lugar donde se encuentra un problema. Otra limitación es debido a que el método de verificación se basa en GI-Nets, por lo cual, para

aplicarlo a modelos no estructurados de PNCs, como PNCs definidos en BPMN, los mismos tienen que ser previamente transformados a modelos estructurados. Esto se puede lograr mediante métodos de estructuración de procesos como el propuesto por (Vanhatalo, Völzer & Koehler 2009).

8.1.3. Método de Verificación de PNCs basado en Anti-Patrones

Con el propósito de proveer un método de verificación de PNCs que mejore el criterio de rendimiento sin perjudicar el criterio de completitud, se propuso un método de verificación basado en el uso de anti-patrones de comportamiento. Para obtener los anti-patrones de comportamiento de un lenguaje de PNCs se propuso un nuevo enfoque basado en el análisis de los constructores del lenguaje. Este enfoque se basa en los conceptos de PNCs mínimos y perfil de no solidez de un lenguaje de PNC. Obteniendo los PNCs mínimos de un lenguaje, es posible obtener todas las combinaciones posibles de constructores de flujo de control, verificarlas y determinar aquellas que conducen a la no solidez de un modelo de PNC. La importancia de los PNCs mínimos es que todo modelo de PNC va a estar conformado por alguna de las combinaciones de elementos definidas en los PNCs mínimos. El conjunto completo de todos los PNCs mínimos no sólidos define un *perfil de no solidez* del lenguaje. A partir de los PNCs mínimos no sólidos se demostró cómo es posible especificar los anti-patrones de comportamiento de un lenguaje de PNC.

Los principales beneficios de este método son:

- es independiente de los lenguajes de PNCs, lo cual lo hace flexible y adaptable a ser utilizado con distintos lenguajes y, por lo tanto, puede ser usado para verificar PNCs en diferentes etapas de un proceso de desarrollo;
- cada PNC mínimo del perfil de no solidez puede ser verificado con cualquier método formal, pudiendo utilizarse el lenguaje formal más adecuado a cada PNC mínimo de acuerdo a la semántica de comportamiento de los constructores definidos en él. Esto permite cumplir con el criterio de completitud, pudiendo verificar la mayoría de los tipos posibles de constructores de flujo de control;

- reutiliza los resultados de verificación obtenidos por métodos formales para verificar el comportamiento de modelos de PNCs, lo que posibilita mejorar el rendimiento de la verificación;
- detecta el lugar donde se produce un error y la combinación de los elementos que causan dicho problema en un modelo de PNC, lo que facilita la tarea de corrección de errores e interpretación de resultados por parte del diseñador;

Se demostró la aplicación del método de verificación basado en anti-patrones al lenguaje UP-ColBPIP.

Como limitaciones del método se puede indicar que sólo puede ser aplicado a modelos estructurados de PNCs y que el enfoque propuesto para descubrir y especificar anti-patrones no garantiza que se puedan especificar todos los posibles anti-patrones de un lenguaje. Sin embargo, a partir de los resultados de evaluación de los métodos de verificación propuestos, se determinó que el método de verificación basado en anti-patrones provee los mismos resultados que el método basado en GI-Nets, lo cual indica que ambos tienen el mismo grado de completitud.

8.1.4. Método de Transformación para PNCs que Garantiza Alineación de Comportamiento

El método de transformación de PNCs con alineación de comportamiento propuesto tiene como propósito dar soporte a la generación consistente de especificaciones de PNCs a partir de modelos de PNCs, siguiendo un enfoque de desarrollo dirigido por modelos. Para ello se propuso un patrón de transformación de modelos. El método se basa en que el modelo y su correspondiente especificación compartan un mismo modelo formal que los representa. Para ello se hace uso de las GI-Nets, formalizando un modelo de PNC en un modelo de GI-Net, y usar este último para generar la especificación de PNC. Se definió formalmente la condición suficiente para alineación de comportamiento entre un modelo de PNC y su especificación. Los principales beneficios del método son:

- permite generar especificaciones de PNCs alineadas desde el punto de vista de comportamiento con los modelos a partir de las cuales son generadas. Esto contribuye a alcanzar una alineación entre la solución de negocio y la solución

tecnológica, lo que implica que la solución tecnológica satisface y cumple el comportamiento definido en la solución de negocio;

- posibilita garantizar que si el comportamiento de un modelo de PNC está bien definido y satisface un conjunto de propiedades de comportamiento, entonces la especificación de dicho PNC también lo hará. Esto se logra debido a que el modelo y la especificación comparten un mismo modelo formal del PNC basado en una GI-Net;
- puede ser aplicado a diferentes lenguajes de PNCs, dado que se basa en el lenguaje formal GI-Nets;

Se demostró la aplicación del método a los lenguajes UP-ColBPIP y WS-CDL, para generar especificaciones WS-CDL (basadas en tecnologías de servicios Web) a partir de un modelo de PNC definido con el lenguaje UP-ColBPIP.

Como limitaciones del método se puede indicar que el patrón de transformación propuesto sólo genera el esqueleto del flujo de control de un modelo de PNC, dejando de lado los aspectos de datos que tienen que ver con el intercambio de información entre las organizaciones que participan de la colaboración. Sin embargo, la definición modular del mismo permite la adición de otros patrones de transformación que permitan generar los aspectos de datos de los PNCs. Por otro lado, es necesario tener en consideración que el correcto funcionamiento del patrón de transformación depende de una correcta implementación del mismo. Este es, si bien en esta tesis se probó desde un punto de vista teórico que es posible definir un patrón de transformación para generar modelos y especificaciones de PNCs con alineación de comportamiento, la máquina de transformación que implemente dicho patrón de transformación puede presentar errores de programación o una incorrecta definición de los modelos formales de los constructores de los lenguajes utilizados por el patrón.

8.1.5. Herramientas para la Formalización, Verificación, y Generación de PNCs

Para validar y evaluar los métodos propuestos se desarrollaron e implementaron dos herramientas. La herramienta *Global Interaction Nets Tool* implementa la formalización de

PNCs mediante GI-Nets, el método de verificación basado en GI-Nets y el método de transformación de modelos de PNCs para alineación de comportamiento. La otra herramienta implementa el método de verificación basado en anti-patrones como una extensión del *editor UP-ColBPIP* de modelos de PNCs.

Se realizó un experimento con modelos UP-ColBPIP por cada método de verificación de PNCs. En los experimentos realizados no se confirmaron inconsistencias entre el método de verificación basado en GI-Nets y el basado en anti-patrones. Esto es un indicador de la completitud del método basado en anti-patrones, ya que no sólo se pudieron verificar con el mismo todos los modelos verificados con el método basado en GI-Nets, sino que además, se obtuvieron los mismos resultados de verificación. Además, los resultados experimentales muestran que el rendimiento del método basado en anti-patrones supera ampliamente a la del método basado en GI-Nets, con tiempos inferiores a 0,5 milisegundos para el primero, comparados con tiempos superiores a los 1000 milisegundos para el segundo. Por otro lado, la utilización de anti-patrones favorece la integración y reutilización del método de verificación en distintos lenguajes y etapas del modelado de PNCs, ya que sólo se deben implementar algoritmos simples de búsqueda de coincidencias de elementos para la detección de anti-patrones y no es necesario interactuar con ninguna aplicación externa.

8.2. Trabajos Futuros

8.2.1. Verificación del Flujo de Datos

Los métodos de verificación propuestos en esta tesis contemplan aspectos del comportamiento de modelos de PNCs. Sin embargo, estos métodos no contemplan aspectos relacionados a los datos que se definen en dichos procesos. Por este motivo, es necesaria la definición de métodos y herramientas que permitan llevar a cabo una verificación semántica de los modelos de PNCs.

Un modelo de PNC puede estar correctamente definido desde el punto de vista del flujo de control, y tener sin embargo errores en la semántica del intercambio de información entre las organizaciones. El intercambio de información es también un aspecto

importante a considerar en el modelado de PNCs ya que la definición o uso incorrecto de los datos puede llevar a errores en la lógica de la colaboración. Por lo tanto, es necesario determinar si la semántica de los mensajes a intercambiar entre las organizaciones es correcta. Por este motivo, se proyecta la aplicación de técnicas de análisis semántico de la estructura de modelos de PNCs, a través del uso de ontologías y reglas de inferencia, para llevar adelante la verificación semántica de estos modelos.

8.2.2. Validación de PNCs

Otro problema importante es determinar si la lógica de negocio definida en los modelos de PNCs cumple los compromisos asumidos por las organizaciones. Para esto es necesario llevar a cabo una validación de los modelos de PNCs, como así también realizar un análisis de rendimiento de los mismos. Para alcanzar esto, se proyecta la aplicación de formalismos de simulación de eventos discretos con Redes de Petri que permitan la validación de los requerimientos funcionales (compromisos asumidos por las partes) y no funcionales (detección de cuellos de botella, cálculo del tiempo de finalización promedio, etc) de modelos de PNCs.

8.2.3. Alineación entre PNCs y Procesos Privados

Actualmente, los modelos de PNCs son utilizados para generar automáticamente los modelos de procesos privados de las organizaciones (Lazarte et al. 2010). Sin embargo, los métodos de transformación pueden contener errores que ocasionen inconsistencias entre los modelos de los procesos privados y los modelos de los PNCs a partir de los cuales fueron definidos. Esto implicaría que las organizaciones no cumplan con los acuerdos establecidos en la colaboración. Por este motivo, otro trabajo que se proyecta es llevar a cabo la definición de métodos de transformación que permitan generar automáticamente los modelos de procesos privados de las organizaciones de manera tal que se garantice consistencia con los modelos de los PNCs a partir de los cuales fueron generados. Esta consistencia se debe cumplir tanto para el comportamiento como para los datos definidos en los modelos de PNCs.

Anexo A. Ejemplos de Módulos GI Concretos

En este anexo se presentan ejemplos de módulos GI concretos que formalizan constructores de los lenguajes UP-ColBPIP y BPMN.

A.1 Ejemplos de Módulos GI concretos para elementos de UP-ColBPIP

A continuación se presentan ejemplos de módulos GI concretos para instancias de los siguientes constructores del lenguaje UP-ColBPIP: *Xor, And, Or/SyncMerge, Or/N-de-M, Multiple Instances, y Exception*.

Xor

Un constructor *Xor* representa que sólo uno de un conjunto de caminos de interacción alternativos puede ser ejecutado. La Figura **A.1** muestra el módulo GI abstracto que formaliza el constructor *Xor*.

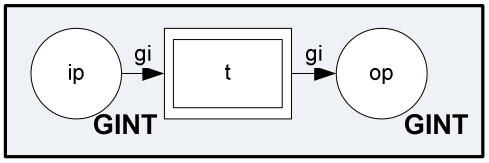
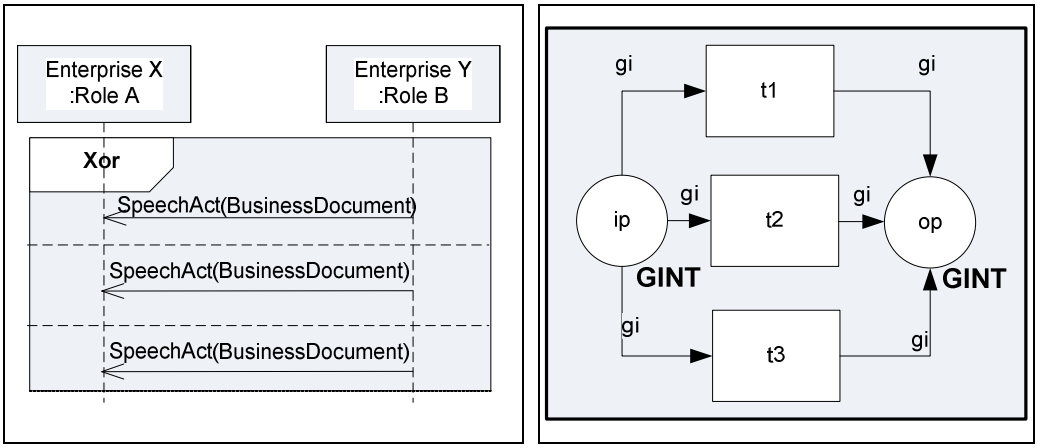


Fig. A.1. Módulo GI abstracto del constructor Xor

La Figura **A.2** muestra un ejemplo de una instancia del constructor *Xor* en UP-ColBPIP con tres caminos de interacción (Figura **A.2** a)) y su respectiva formalización con un módulo GI concreto (Figura **A.2** b)). Este módulo consiste de dos posiciones *ip* y *op* que representan la entrada y salida del módulo respectivamente. Tiene además tres transiciones concretas *t1*, *t2*, y *t3*, que conectan a las posiciones mencionadas, donde cada transición es un posible camino de ejecución del *Xor* alternativo. Si hay una señal en la posición *ip*, sólo una de las transiciones puede ser ejecutada. Una vez ejecutada una transición, las demás

quedan inhabilitadas, representando de esta manera la exclusión mutua entre los caminos de interacción. Cuando finaliza su ejecución se coloca una señal en el place de salida *op*, representando la unión de los caminos alternativos.



a) Xor con tres caminos de interacción b) Módulo GI concreto del Xor

Fig. A.2. Ejemplo de *Xor* en UP-ColBPIP y su módulo GI concreto

And

Un constructor *And* representa la ejecución paralela de caminos de interacción. La sincronización se lleva a cabo luego que finalizaron todos los caminos paralelos. La Figura **A.3** muestra el módulo GI abstracto del constructor *And*.

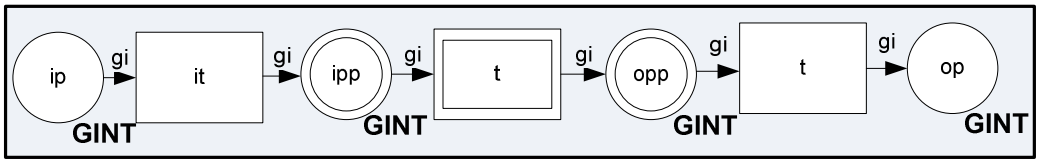
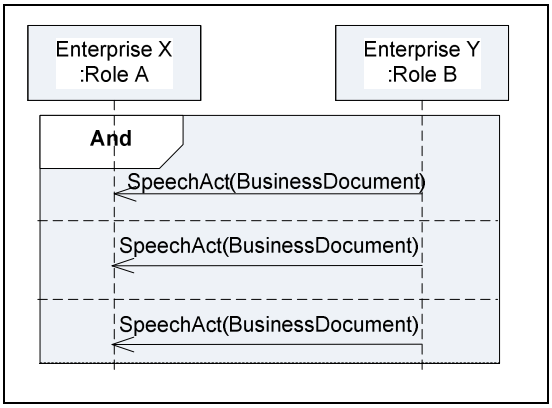


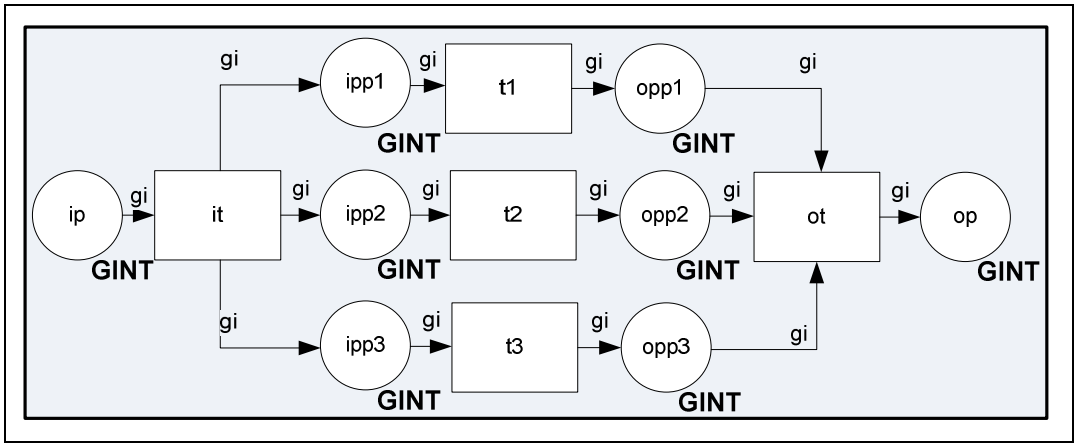
Fig. A.3. Módulo GI abstracto del constructor And

La Figura **A.4** muestra un ejemplo de una instancia del constructor *And* con tres caminos de interacción en paralelo (Figura **A.4** a)) y su respectivo módulo GI concreto que lo formaliza (Figura **A.4** b)). Este módulo contiene dos posiciones *ip* y *op* que representan la entrada y salida del módulo respectivamente, y dos transiciones *it* y *ot* que representan la

bifurcación y sincronización de los caminos paralelos respectivamente. Tiene además tres transiciones concretas *t1*, *t2*, y *t3*, donde cada transición representa un camino de interacción del *And* que se ejecuta en paralelo, donde *ipp1*, *ipp2*, e *ipp3* son las posiciones de entrada de esos caminos de interacción y *opp1*, *opp2*, y *opp3* son las posiciones de salida de dichos caminos.



a) And con tres caminos de interacción



b) Módulo GI concreto del And

Fig. A.4. Ejemplo de And en UP-ColBPIP y su módulo GI concreto

Si hay una señal en la posición *ip*, se puede ejecutar la transición *it*. La ejecución de dicha transición coloca una señal en las posiciones *ipp1*, *ipp2*, e *ipp3*, representando de esta manera el paralelismo entre los caminos de interacción. La finalización de la ejecución de todos los caminos se logra cuando existe una señal en cada uno de los places *opp1*, *opp2*, y

opp3. Una vez que finaliza la ejecución de todos los caminos, se habilita la transición *ot* que permite llevar a cabo la sincronización y depositar una señal en la posición final *op*.

Or/Synchronizing Merge

El constructor *Or* con sincronización *Synchronizing Merge* permite la ejecución de dos o más caminos de interacción alternativos, donde todos los caminos que se ejecuten deben ser sincronizados. La Figura **A.5** muestra el módulo GI abstracto de este constructor.

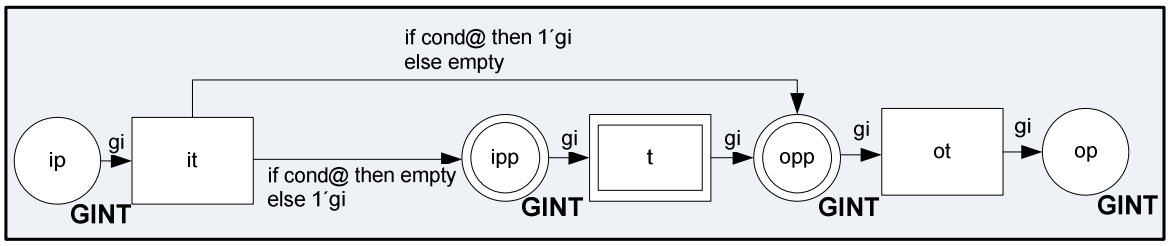
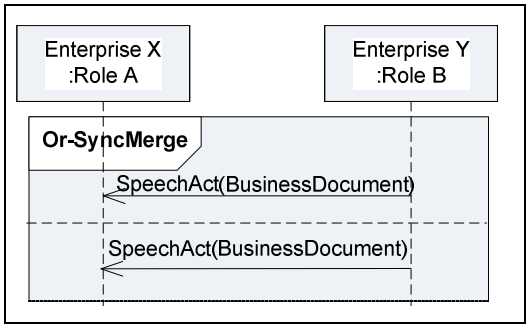
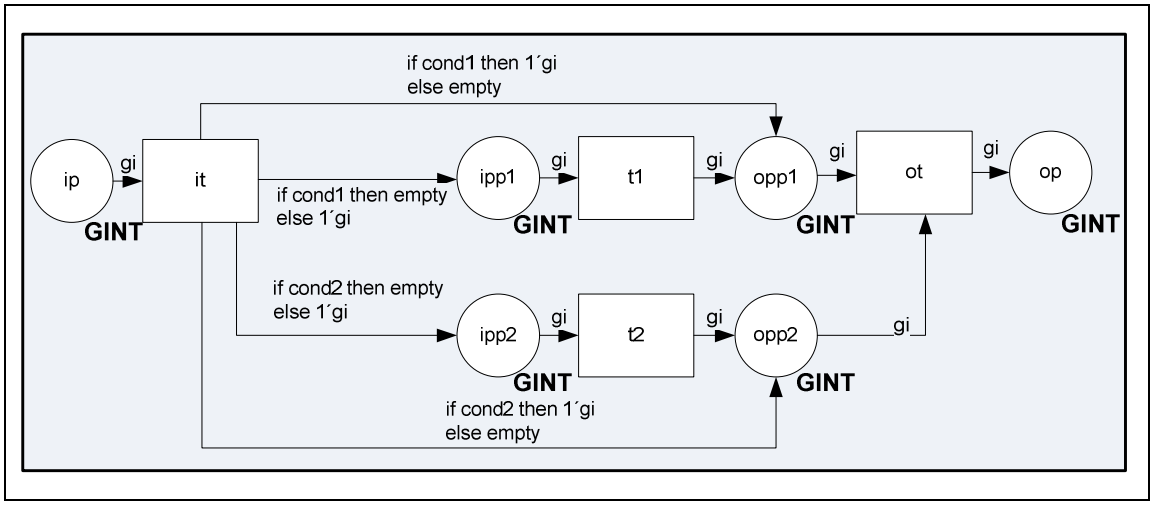


Fig. A.5. Módulo GI abstracto del constructor Or con Synchronizing Merge

La Figura **A.6** muestra un ejemplo de una instancia del constructor *Or* con dos caminos de interacción (Figura **A.6** a)) y su respectivo módulo GI concreto (Figura **A.6** b)). Este módulo contiene la posición inicial *ip* y final *op*, y dos transiciones *it* y *ot* que representan la bifurcación y sincronización del *Or* respectivamente. Tiene además dos transiciones *t1* y *t2*, donde cada transición representa un camino de interacción del *Or*, en los cuales *ipp1* e *ipp2* son las posiciones de entrada de esos caminos de interacción y *opp1* y *opp2* son las posiciones de salida.



a) Or con Synchronizing Merge con dos caminos



b) Módulo GI concreto del Or con Synchronizing Merge

Fig. A.6. Ejemplo de Or con Synchronizing Merge en UP-ColBPIP y su módulo GI concreto

Si hay una señal en la posición *ip*, se puede ejecutar la transición *it*. Si las variables booleanas *cond1* y *cond2* retornan verdadero, se deposita una señal en las posiciones *ipp1* e *ipp2*, y las posiciones *opp1* y *opp2* quedan vacías. Esto representa la ejecución de todos los caminos de interacción que forman parte del *Or*. En cambio, si *cond1* y *cond2* retornan falso, se deposita una señal en las posiciones *opp1* e *opp2*, y los places *ipp1* y *ipp2* quedan vacíos. Esto representa que no se ejecutan los caminos de interacción que forman parte del *Or*. Las variables *cond1* y *cond2* son independientes entre sí, lo que permite que se ejecuten uno, ambos o ninguno de los caminos del *Or*. Una vez que finaliza la ejecución de todos los caminos, se habilita la transición *ot* que permite llevar a cabo la sincronización y depositar una señal en la posición final *op*.

Or/N-out-of-M

El constructor *Or* con sincronización *N-out-of-M* permite la ejecución de dos o más caminos de interacción alternativos, donde *N* de un total de *M* caminos deben ser sincronizados, y el resto deben ser descartados. La Figura **A.7** muestra el módulo GI abstracto de este constructor.

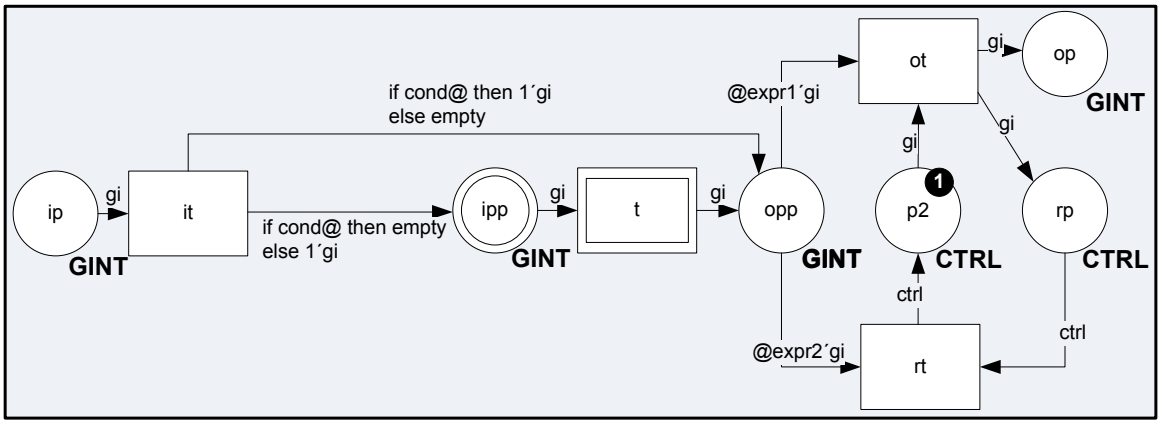
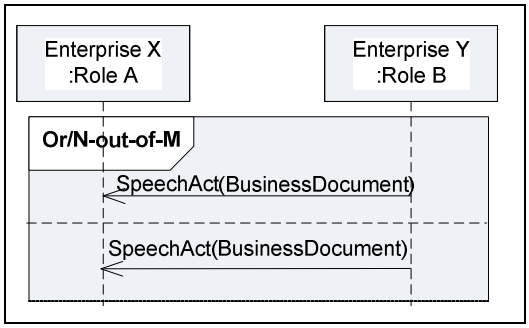
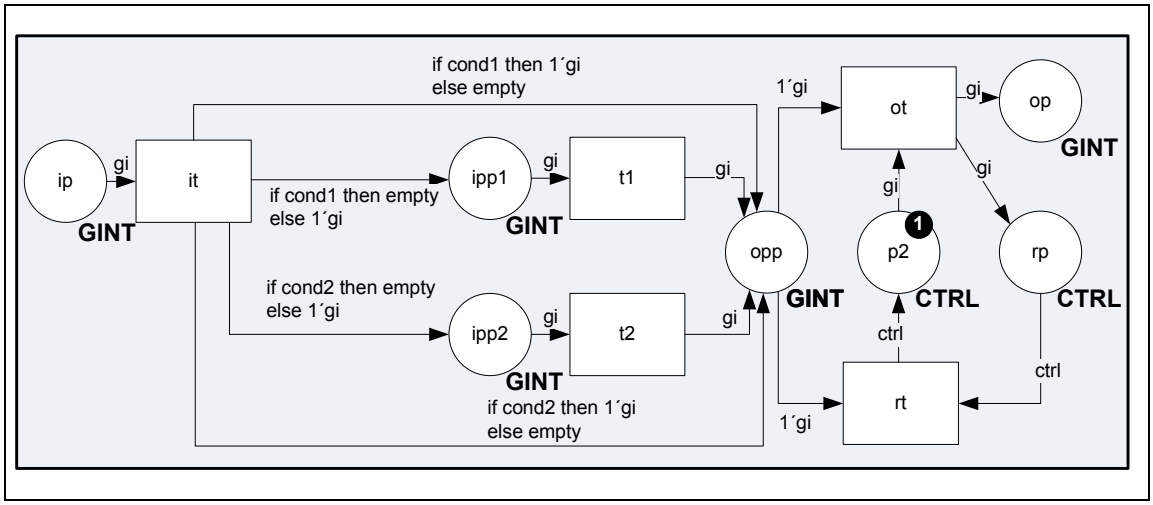


Fig. A.7. Módulo GI abstracto del constructor Or con N-out-of-M

La Figura **A.8** muestra un ejemplo de una instancia del constructor *Or* en la cual se deben sincronizar uno de dos *(1-out-of-2)* posibles caminos de interacción (Figura **A.8** a)) y su respectivo módulo GI concreto (Figura **A.8** b)). Este módulo contiene la posición inicial *ip* y final *op*, y dos transiciones *it* y *ot* que representan la bifurcación y sincronización del *Or* respectivamente. Tiene además dos transiciones *t1* y *t2*, donde cada transición representa un camino de interacción del *Or*, en los cuales *ipp1* e *ipp2* son las posiciones de entrada de esos caminos de interacción y *opp1* y *opp2* son las posiciones de salida.



a) Or con sincronización 1-out-of-2



b) Módulo GI concreto del Or con N-out-of-M

Fig. A.8. Ejemplo de Or con N-out-of-M en UP-ColBPIP y su módulo GI concreto

El comportamiento de la parte del módulo que representa a la bifurcación de tipo *Or* es similar a la presentada para el ejemplo del constructor Or/SyncMerge (Figura **A.6** b)). El arco que va de la posición *opp* a la transición *ot* indica que debe haber al menos una señal en la posición *opp* para que se habilite la transición *ot*. Este arco representa a la cantidad de caminos *n* que deben ser sincronizados. En este ejemplo sólo un camino debe ser sincronizado y el otro descartado. Además, para que *ot* esté habilitada es necesario que haya una señal en la posición *p2*. Está posición tiene una señal en el estado inicial (indicada con un punto negro), con lo cual partiendo del estado inicial, la transición *ot* sólo necesita que se deposite una señal en la posición *opp* para estar habilitada.

El arco que va de la posición *opp* a la transición *rt* indica que debe haber al menos una señal en la posición *opp* para que se habilite la transición *rt*. Este arco representa a los

m-n caminos que deben ser descartados, en este caso se debe descartar sólo un camino. Además, para que *rt* esté habilitada es necesario que haya una señal en la posición *rp*.

De esta manera, si se ejecuta la transición *ot*, se consume una señal de las posiciones *opp* y *p2* y se deposita una señal en las posiciones *op* y *rp*. La transición *ot* ya no puede ser habilitada hasta que las posiciones *opp* y *p2* tengan una señal nuevamente. Esto impide que la transición *ot* se vuelva a ejecutar antes que se descarte el otro camino.

Ahora, al haber una señal en la posición *rp* la transición *rt* sólo necesita que haya una señal en la posición *opp*. La ejecución de esta transición permite que se descarte el camino restante y que el constructor quede en su estado inicial.

Multiple Instances

Un constructor *Multiple Instances* representa la ejecución en paralelo de instancias múltiples de un camino de interacción y su sincronización. El número de instancias a ejecutarse y sincronizarse puede ser conocido en tiempo de diseño, en tiempo de ejecución o bien no conocido. En UP-ColBPIP existe un constructor *Multiple Instances* para cada una de estas variantes. La Figura **A.9** muestra el módulo GI abstracto del constructor *Multiple Instances*, el cual es aplicable cuando se conocen el número de instancias en tiempo de diseño.

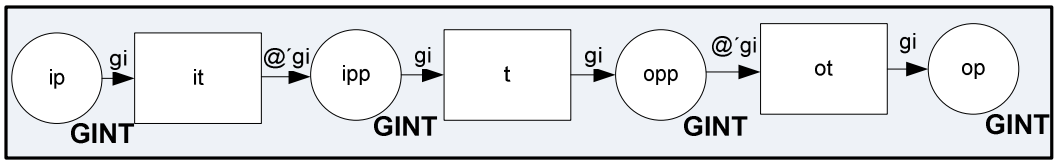
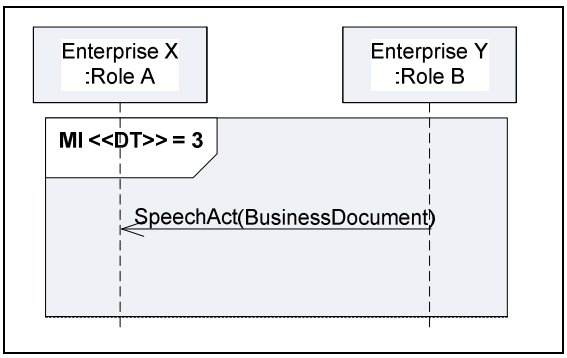


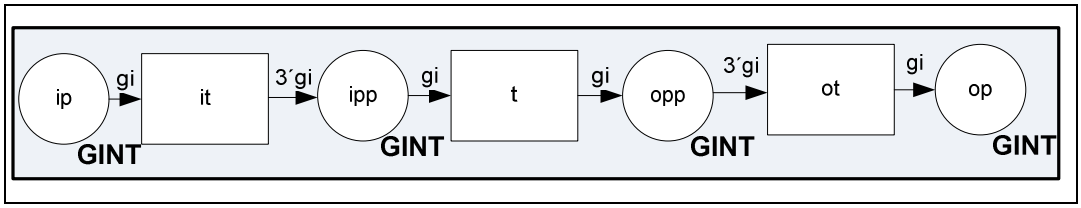
Fig. A.9. Módulo GI abstracto del constructor Multiple Instances

La Figura **A.10** muestra un ejemplo de un elemento *Multiple Instances* con conocimiento en tiempo de diseño de la cantidad de instancias, las cuales son tres (Figura **A.10** a)), y su respectivo módulo GI concreto (Figura **A.10** b)). El arco que va desde la transición *it* a la posición *ipp* permite depositar tres señales en la posición *ipp* representando la generación de tres instancias, y el que va desde la posición *opp* a la transición *ot* permite que la transición *ot* se ejecute sólo si la posición *opp* contiene tres señales, representando de

esta manera la sincronización de las instancias. La transición *t* representa al camino de interacción que se va a ejecutar tres veces, una por cada instancia.



a) Multiple Instances con tres instancias

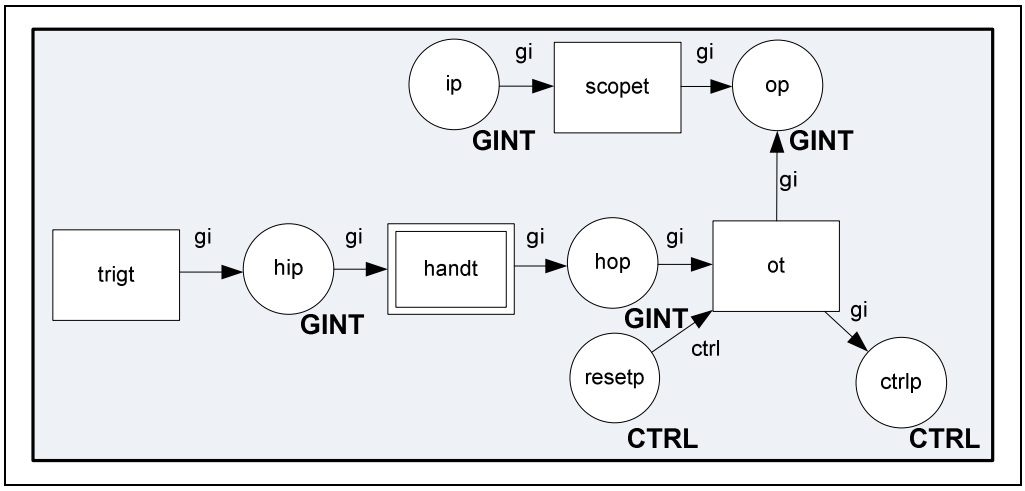


b) Módulo GI concreto del Multiple Instances

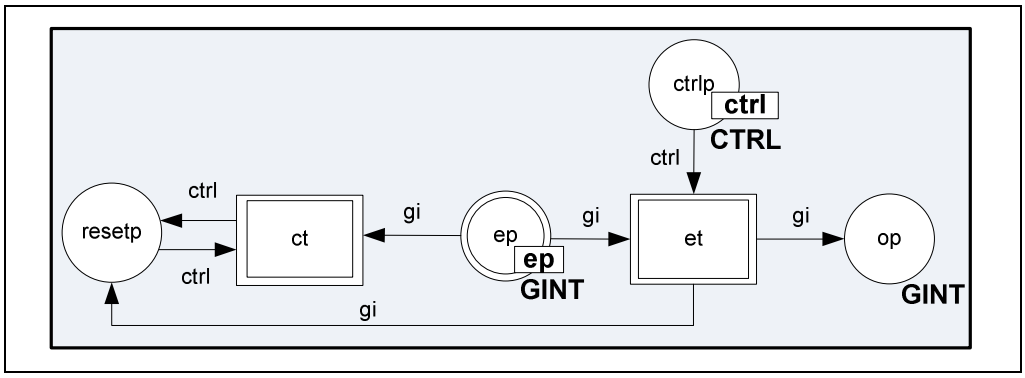
Fig. A.10. Ejemplo de Multiple Instances en UP-ColBPIP y su módulo GI concreto

Exception

Un constructor *Exception* define el camino a seguir luego que ocurre una determinada excepción. Consiste de un camino de interacción que representa al alcance de la excepción, el cual va a contener a todos los elementos que formen parte de dicho camino de interacción, y de uno o más caminos de interacción que representan los manejadores de excepciones que capturan y gestionan excepciones que ocurren dentro el alcance de la excepción. La Figura **A.11** muestra el módulo GI abstracto del constructor *Exception*. El módulo está divido en dos partes, el constructor *Exception* (Figura **A.11** a)) y el disparador (Trigger) de las excepciones (Figura **A.11** b)).



a) Exception



b) Trigger

Fig. A.11. Módulo GI abstracto del constructor Exception

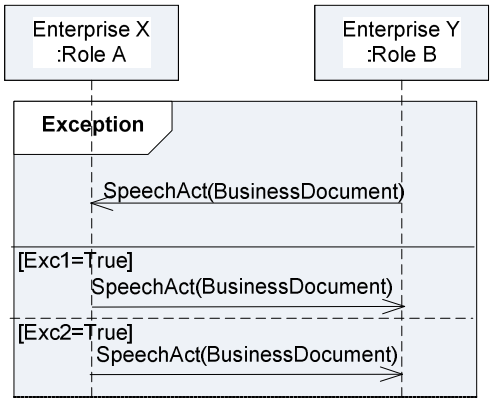
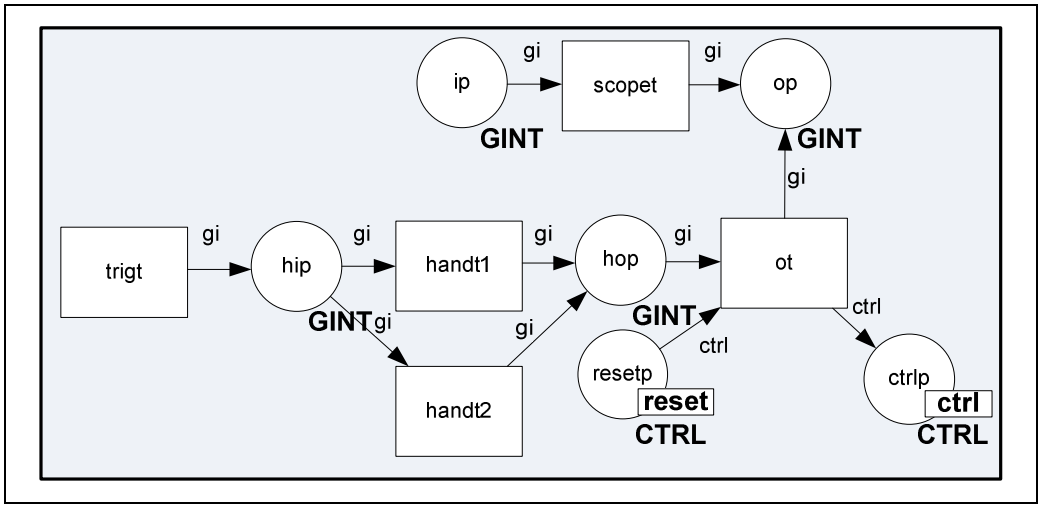
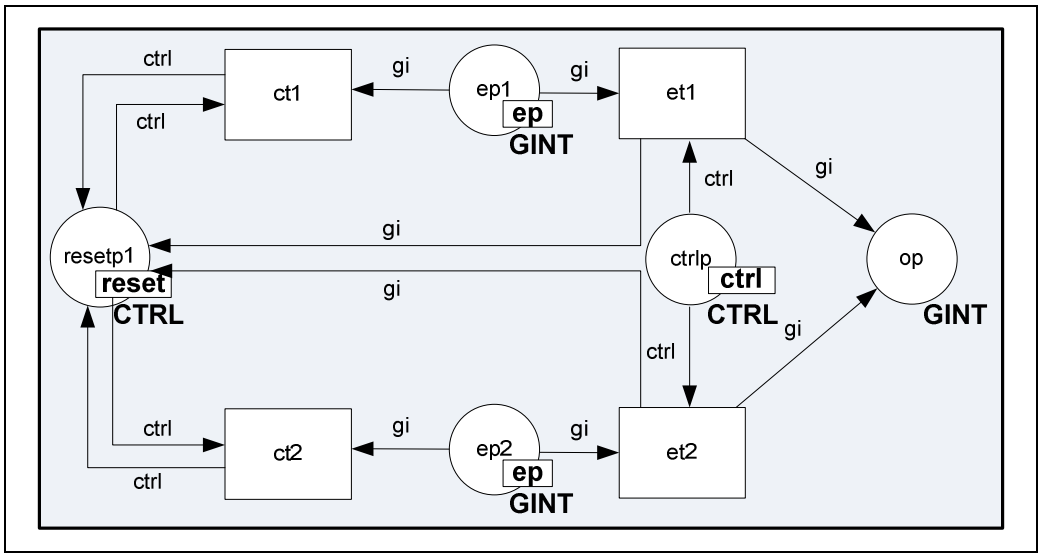


Fig. A.12. Ejemplo de Exception en UP-ColBPIP

La Figura **A.12** muestra un ejemplo de una instancia del constructor *Exception* con dos manejadores de excepciones, los cuales indican las condiciones a evaluar para el disparo de las excepciones. Su respectivo módulo GI concreto se muestra en la Figura **A.13**. En la Figura **A.13** a) se muestran dos manejadores de excepciones, donde la transición *trigt* representa al disparador de la excepción, mientras que las transiciones *handt1* y *handt2* representan a los manejadores de la excepción. En la Figura **A.13** b) se muestra el módulo Trigger, que contiene dos puntos de excepción representados por las posiciones *ep1* y *ep2*.



a) Módulo GI concreto del Exception



b) Módulo GI concreto del Trigger

Fig. A.13. Ejemplo de Exception en UP-ColBPIP y su módulo GI concreto

En este ejemplo se puede ejecutar una excepción si hay una señal en alguna de las posiciones *ep1* o ep2, las cuales representan a los puntos de excepción. La transiciones *et1* y *et2* permiten ejecutar la excepción depositando una señal en la posición *op*. Esta posición está asociada a la posición *hip* del módulo Exception, con lo cual si hay una señal en la posición *op*, también habrá una señal en la posición *hip*. Esta posición habilita a que se ejecute el manejador de excepciones representado por la transición *handt* del modulo Exception. La transición *et* deposita además una señal en la posición *resetp*, habilitando de

esta manera a la transición *ct* que se encarga de limpiar todas las señales que quedaron en las posiciones dentro del alcance de la excepción.

Una vez que finaliza la ejecución del manejador de excepciones se deposita una señal en la posición *hop*, se ejecuta la transición de salida *ot*, la cual deposita una señal en la posición final *op*, que representa la finalización del módulo Exception.

A.2 Ejemplos de Módulos GI concretos para elementos de BPMN

A continuación se presentan ejemplos de módulos GI concretos para instancias de los siguientes constructores estructurados de BPMN: *Parallel/Exclusive*, *Parallel/Complex*, *Exclusive/Parallel*, *Exclusive/Complex*, *Inclusive/Parallel*, e *Inclusive/Exclusive*.

Parallel/Exclusive

La Figura **A.14** muestra el módulo GI abstracto del constructor estructurado de BPMN que combina los gateways *Parallel* y *Exclusive*. Este módulo contiene dos posiciones *ip* y *op* que representan inicial y final del módulo respectivamente, y la transición *it* que representan la bifurcación del *Parallel*. La posición abstracta *ipp* junto con la transición abstracta *t* permiten representar a todos los posibles flujos de secuencia paralelos del *Parallel*. La unión del *Exclusive* gateway se representa por la transición abstracta *t*, la cual está conectada a la posición final *op*.

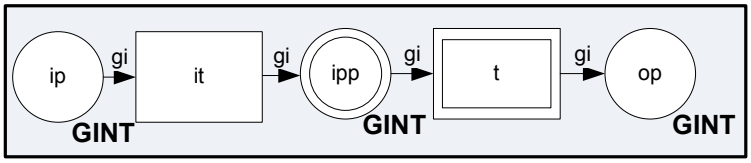
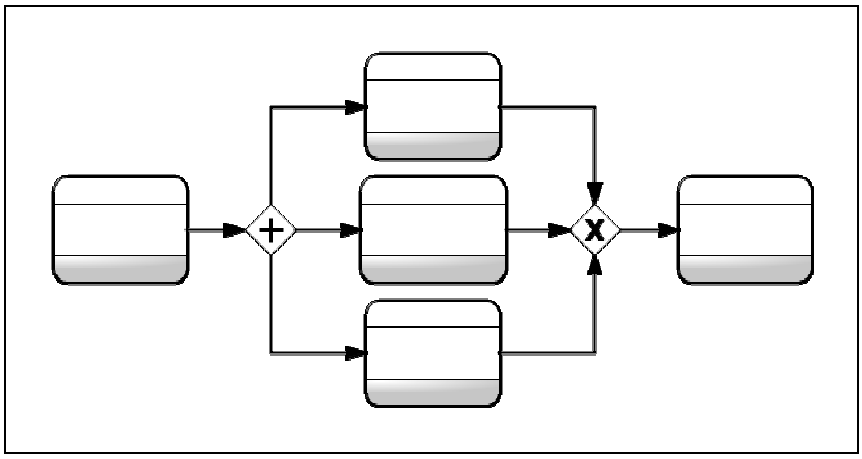


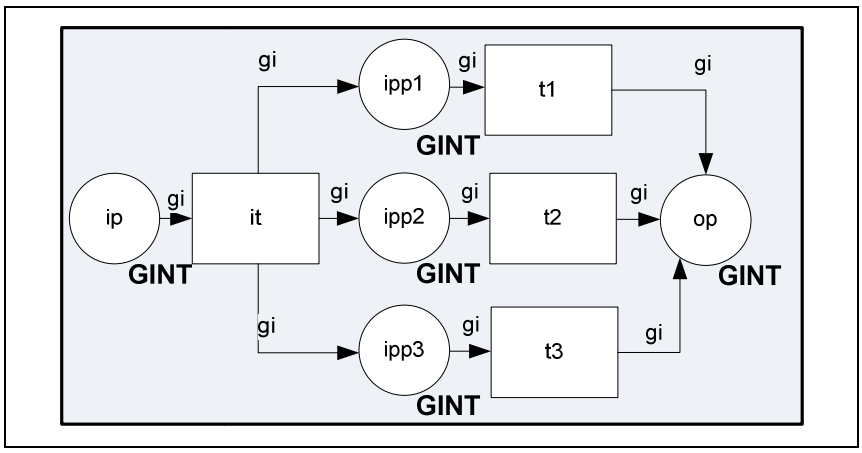
Fig. A.14. Módulo GI abstracto de Parallel/Exclusive

La Figura **A.15** muestra un ejemplo de un constructor estructurado de BPMN compuesto de un *parallel gateway* y un *exclusive gateway* con tres flujos de secuencia

paralelos (Figura **A.15** a)) y el respectivo módulo GI concreto que lo formaliza (Figura **A.15** b)). Este módulo contiene dos posiciones *ip* y *op* que representan la entrada y salida del módulo respectivamente, y una transición *it* que representa la bifurcación del *Parallel*. Tiene además tres transiciones concretas *t1*, *t2*, y *t3*, donde cada transición representa un flujo de secuencia del *Parallel*, donde *ipp1*, *ipp2*, e *ipp3* son las posiciones de entrada de esos flujos de secuencia y *op* es la posición de salida de dichos flujos.



a) Constructor Parallel/Exclusive con tres caminos



b) Módulo GI concreto del Parallel/Exclusive

Fig. A.15. Ejemplo de Parallel/Exclusive en BPMN y su módulo GI concreto

Si hay una señal en la posición *ip*, se puede ejecutar la transición *it*. La ejecución de dicha transición coloca una señal en las posiciones *ipp1*, *ipp2*, e *ipp3*, representando de esta manera el paralelismo entre los flujos de secuencia. Por cada flujo de secuencia que finaliza

su ejecución se deposita una señal en la posición *op*, representando de esta manera al *Exclusive gateway*.

Parallel/ Complex

La Figura **A.16** muestra el módulo GI abstracto del constructor estructurado de BPMN que se forma a partir de la combinación de un *Parallel* gateway y un *Complex* gateway. Este módulo contiene dos posiciones *ip* y *op* que representan la entrada y salida del módulo respectivamente, y la transición *it* que representa la bifurcación del *Parallel*. La posición abstracta *ipp* junto con la transición abstracta *t* permiten representar a todos los posibles flujos de secuencia paralelos del *Parallel*.

La sincronización del *Complex* gateway se representa por el conjunto de posiciones y transiciones que siguen a continuación de la transición abstracta *t*. La descripción de esta parte del constructor es idéntica a la realizada para el constructor *Or/N-out-of-M* de UP-ColBPIP (Figura **3.11**).

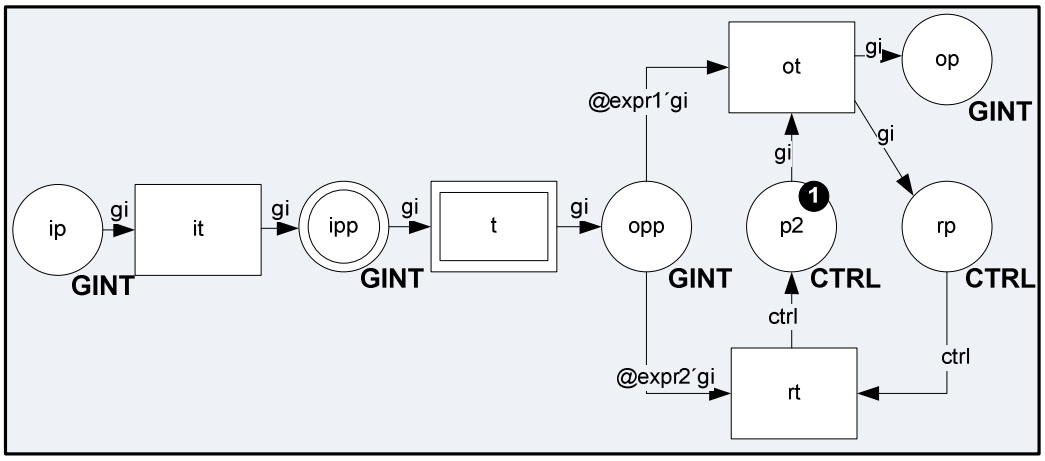
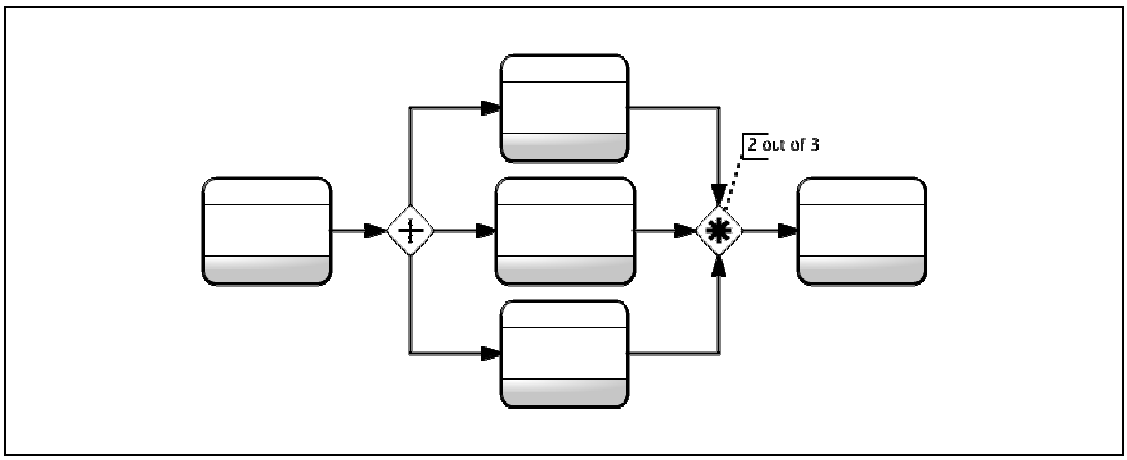


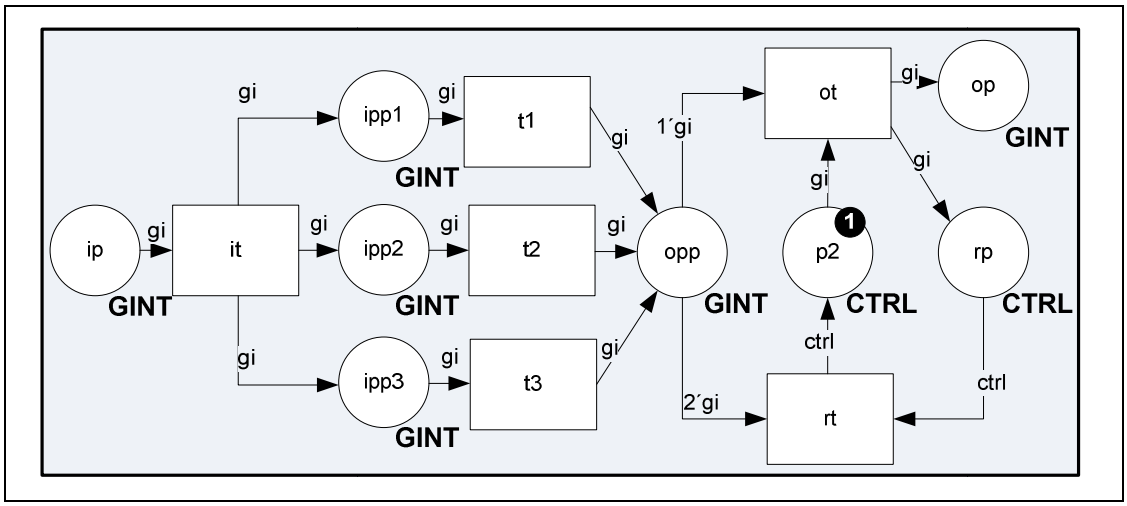
Fig. A.16. Módulo GI abstracto del constructor Parallel/Complex

La Figura **A.17** muestra un ejemplo de un constructor estructurado de BPMN compuesto de un *parallel gateway* con un *complex gateway* con tres flujos de secuencia paralelos (Figura **A.17** a)) y su respectivo módulo GI concreto que lo formaliza (Figura **A.17** b)). Este módulo contiene dos posiciones *ip* y *op* que representan las posiciones inicial y final del módulo respectivamente, y una transición *it* que representa la bifurcación

del *Parallel*. Tiene además tres transiciones concretas *t1*, *t2*, y *t3*, donde cada transición representa un flujo de secuencia del *Parallel*, donde *ipp1*, *ipp2*, e *ipp3* son las posiciones de entrada de esos flujos de secuencia y *opp* es la posición de salida de los mismos.



a) Constructor Parallel/Complex con tres caminos



b) Módulo GI concreto del constructor Parallel/Complex

Fig. A.17. Ejemplo de constructor Parallel/Complex en BPMN y su módulo GI concreto

Si hay una señal en la posición *ip*, se habilita la transición *it*. La ejecución de dicha transición coloca una señal en las posiciones *ipp1*, *ipp2*, e *ipp3*, representando de esta manera el paralelismo entre los flujos de secuencia. Por cada flujo de secuencia que finaliza su ejecución se deposita una señal en la posición *opp*.

El arco que va de la posición *opp* a la transición *ot* indica que debe haber al menos dos señales en la posición *opp* para que se habilite la transición *ot*. Este arco representa a la cantidad de caminos *n* que deben ser sincronizados. En este ejemplo dos caminos deben ser sincronizados y el otro descartado. Además, para que *ot* esté habilitada es necesario que haya una señal en la posición *p2*. Está posición tiene una señal en el estado inicial (indicada con un círculo negro), con lo cual partiendo del estado inicial, la transición *ot* sólo necesita que se depositen dos señales en la posición *opp* para estar habilitada.

El arco que va de la posición *opp* a la transición *rt* indica que debe haber al menos una señal en la posición *opp* para que se habilite la transición *rt*. Este arco representa a los *m-n* caminos que deben ser descartados, en este caso se debe descartar sólo un camino. Además, para que *rt* esté habilitada es necesario que haya una señal en la posición *rp*.

De esta manera, si se ejecuta la transición *ot*, se consumen dos señales de las posiciones *opp* y una señal de *p2* y se deposita una señal en las posiciones *op* y *rp*. La transición *ot* ya no puede ser habilitada hasta que las posiciones *opp* y *p2* tengan una señal nuevamente. Esto impide que la transición *ot* se vuelva a ejecutar antes que se descarte el otro camino.

Ahora, al haber una señal en la posición *rp* la transición *rt* sólo necesita que haya una señal en la posición *opp*. La ejecución de esta transición permite que se descarte el camino restante y que el constructor quede en su estado inicial.

Exclusive/Parallel

La Figura **A.18** muestra el módulo GI abstracto de los constructores estructurados de BPMN que se forman a partir de la combinación de un *Exclusive gateway* y un *Parallel gateway*. Este módulo contiene dos posiciones *ip* y *op* que representan a la posición de inicio y final del módulo respectivamente, y la transición *ot* que representa la sincronización *Parallel*. Tiene además una transición abstracta *t* que conecta a las posiciones mencionadas. La posición abstracta *opp* junto con la transición abstracta *t* permiten representar a todos los posibles flujos de secuencia alternativos del constructor.

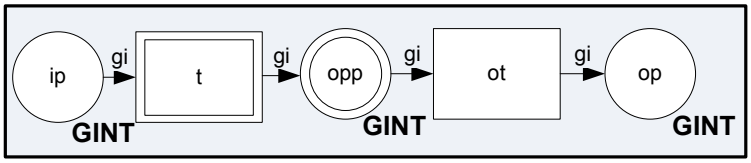
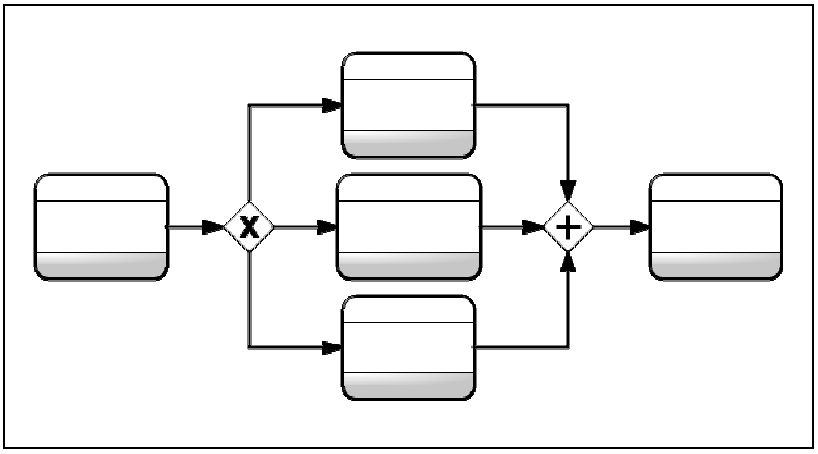
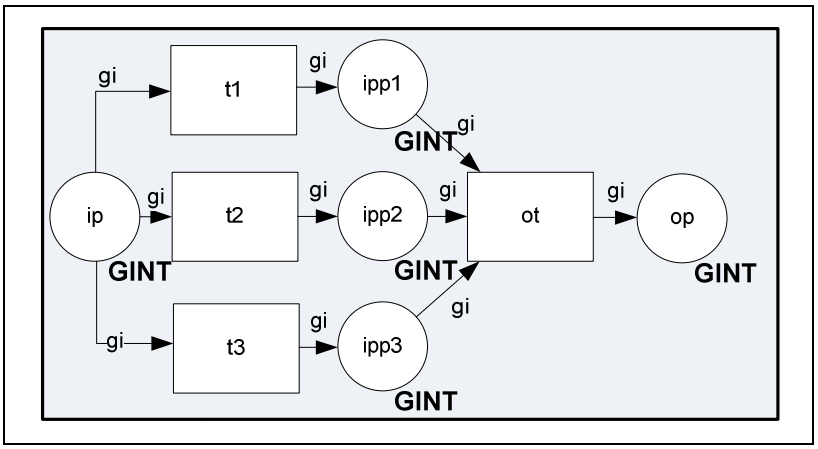


Fig. A.18. Módulo GI abstracto del constructor Exclusive/Parallel

La Figura **A.19** muestra un ejemplo de un constructor estructurado de BPMN compuesto de un *Exclusive gateway* y un *Parallel gateway* con tres flujos de secuencia alternativos (Figura **A.19** a)) y su respectiva formalización con un módulo GI concreto (Figura **A.19** b)). Este módulo consiste de dos posiciones *ip* y *op* que representan las posiciones inicial y final del módulo respectivamente. Tiene además tres transiciones concretas *t1*, *t2*, y *t3*, y tres posiciones *opp1*, *opp2*, y *opp3* que representan los tres flujos de secuencia alternativos. Si hay una señal en la posición *ip*, sólo una de las transiciones *t1*, *t2*, o *t3* puede ser ejecutada. Una vez ejecutada una transición, las demás quedan inhabilitadas, representando de esta manera la exclusión mutua entre los caminos de interacción. Si hay una señal en alguna de las posiciones *opp1*, *opp2*, y *opp3* se habilita la transición *ot* que representa la sincronización *Parallel*, la cual al ejecutarse coloca una señal en el place de salida *op*.



a) Constructor estructurado Exclusive/Parallel con tres caminos



b) Módulo GI concreto del constructor Exclusive/Parallel

Fig. A.19. Ejemplo de constructor Exclusive/Parallel en BPMN y su módulo GI concreto

Exclusive/Complex

La Figura **A.20** muestra el módulo GI abstracto de los constructores estructurados de BPMN que se forman a partir de la combinación de un *Exclusive gateway* y un *Complex gateway*. Este módulo contiene dos posiciones *ip* y *op* que representan las posiciones inicial y final del módulo respectivamente. La transición abstracta *t* permite representar a flujos de secuencia alternativos del *Exclusive gateway*. La sincronización del *Complex gateway* se representa por el conjunto de posiciones y transiciones que siguen a continuación de la transición abstracta *t*. La descripción de la semántica de comportamiento de estos

elementos que representan al *Complex gateway* es igual a la realizada para el constructor *Parallel/Complex* (Figura **3.20**).

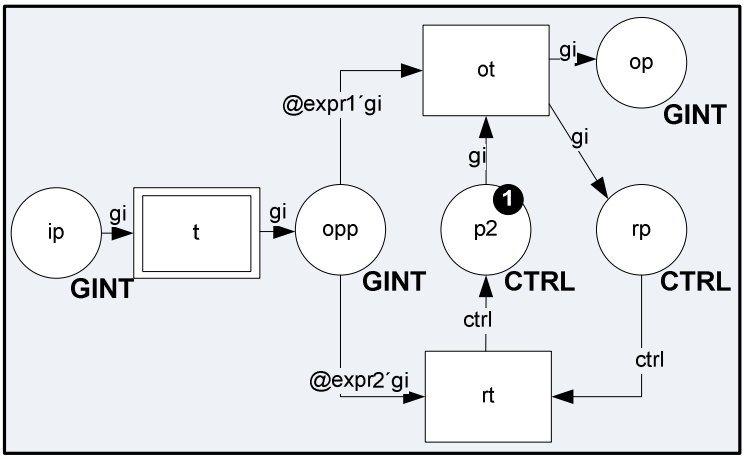
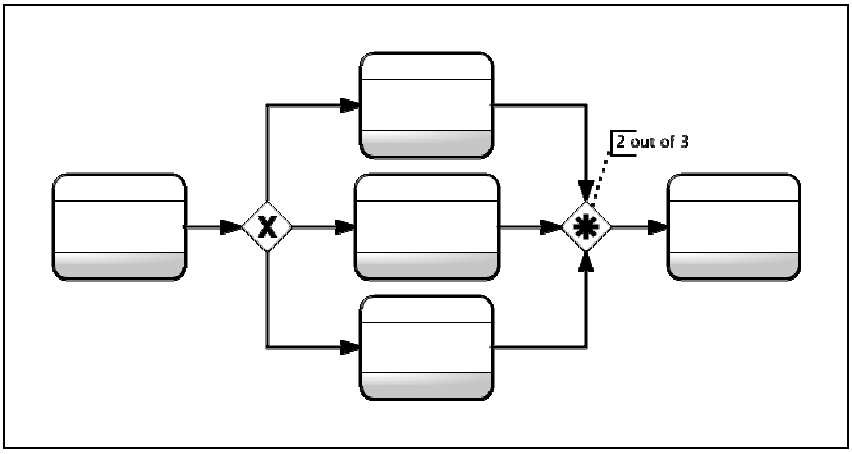


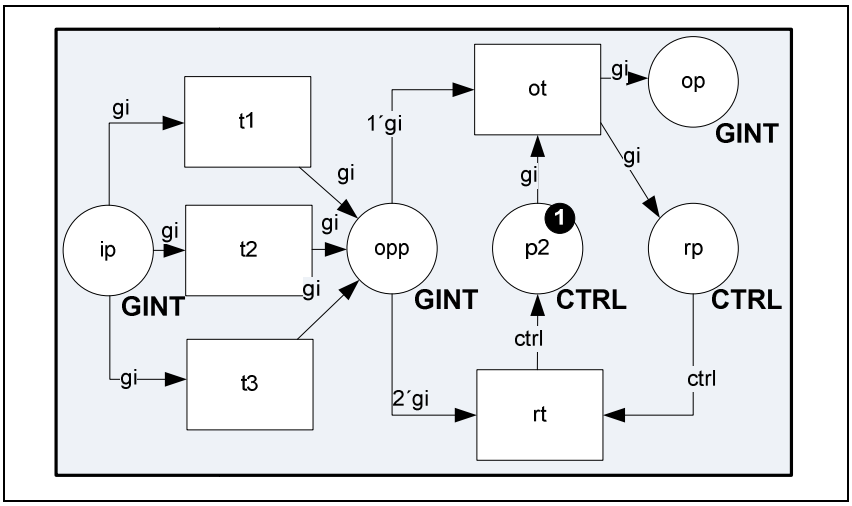
Fig. A.20. Módulo GI abstracto del constructor Exclusive/Complex

La Figura **A.21** muestra un ejemplo de un constructor estructurado de BPMN compuesto de un *Exclusive gateway* con un *Complex gateway* con tres flujos de secuencia paralelos (Figura **A.21** a)) y su respectivo módulo GI concreto que lo formaliza (Figura **A.21** b)). Este módulo consiste de dos posiciones *ip* y *op* que representan las posiciones inicial y final del módulo respectivamente. Tiene además tres transiciones concretas *t1*, *t2*, y *t3*, donde cada transición representa un flujo de secuencia alternativo del *Exclusive gateway*, donde *ip* es la posición de entrada de esos flujos de secuencia y *opp* es la posición de salida de los mismos.

Si hay una señal en la posición *ip*, se puede ejecutar la cualquiera de las transiciones *t1, t2, o t3*. La ejecución cualquiera de estas transiciones coloca una señal en la posición *opp* representando de esta manera la excusión mutua entre los flujos de secuencia. La descripción de la parte de la sincronización por medio del *Complex gateway* es análoga a la realizada para el ejemplo de la Figura **A.17** en el cual se utiliza al Complex gateway para llevar a cabo la sincronización.



a) Constructor Exclusive/Complex con tres caminos



b) Módulo GI concreto del constructor Exclusive/Complex

Fig. A.21. Ejemplo de constructor Exclusive/Complex en BPMN y su módulo GI concreto

Inclusive/Parallel

La Figura **A.22** muestra el módulo GI abstracto del constructor estructurado de BPMN que se forma a partir de la combinación de un *Inclusive gateway* y un *Parallel gateway*. Este módulo contiene dos posiciones *ip* y *op* que representan las posiciones inicial y final del módulo respectivamente, y dos transiciones *it* y *ot* que permiten realizar la bifurcación y sincronización del constructor respectivamente. Las posiciones abstractas *ipp*

y *opp* junto con la transición abstracta *t* permiten representar a todos los posibles flujos de secuencia del *Inclusive gateway*.

La transición *it* deposita una señal en la posición *ipp* si la evaluación de la expresión @*cond* del arco que conecta a dichos elementos retorna verdadero. Caso contrario no se deposita ninguna señal en la posición *ipp*. La expresión @*cond* en estos arcos indica a una variable booleana que debe ser evaluada para determinar si se deposita una señal en la posición *ipp*. La transición *t* deposita una señal en la posición *ipp* si la evaluación de la expresión @*cond* del arco que conecta a *it* con *ipp* retorna verdadero. Esto representa la ejecución de uno de los caminos del *Or*.

Cada módulo GI concreto que formaliza a un elemento que es instancia del constructor *Inlusive/Parallel* debe tener definida una variable booleana distinta por cada arco que conecte a la transición *it* en base a la cantidad de flujos de secuencia que tiene dicho elemento.

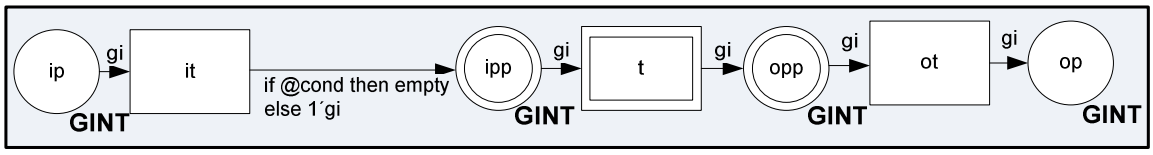
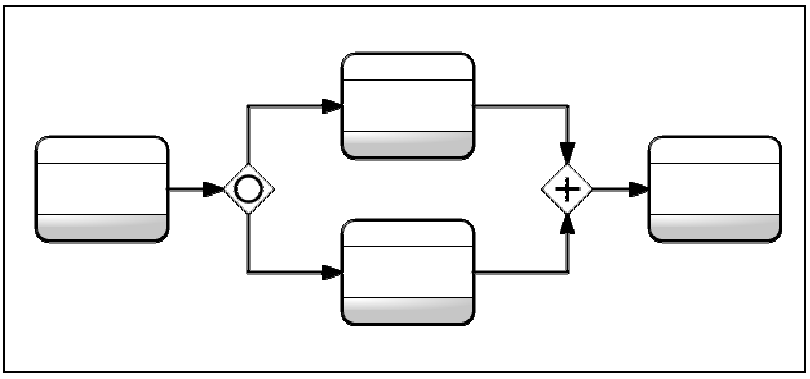


Fig. A.22. Módulo GI abstracto de Inclusive/Parallel

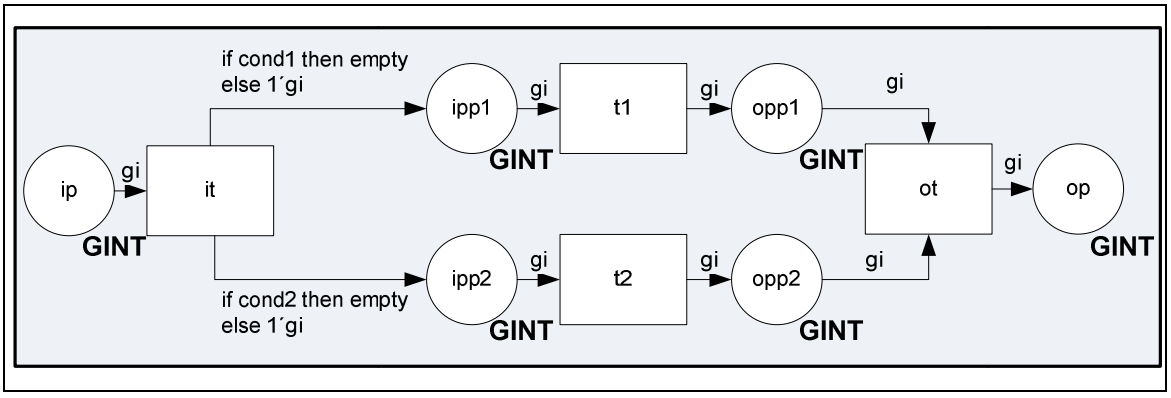
La Figura **A.23** muestra un ejemplo de un constructor estructurado de BPMN compuesto de un *Inclusive gateway* con un *Parallel gateway* con dos flujos de secuencia paralelos (Figura **A.23** a)) y su respectivo módulo GI concreto que lo formaliza (Figura **A.23** b)). Este módulo consiste de dos posiciones *ip* y *op* que representan las posiciones inicial y final del módulo respectivamente, dos transiciones *it* y *ot* que representan la bifurcación y sincronización del *Inclusive gateway* y la sincronización del *Parallel gateway* respectivamente. Tiene además dos transiciones *t1* y *t2*, donde cada transición representa un flujo de secuencia, en los cuales *ipp1* e *ipp2* son las posiciones de entrada de esos caminos de interacción y *opp1* y *opp2* son las posiciones de salida.

Si hay una señal en la posición *ip*, se puede ejecutar la transición *it*. Si las variables booleanas *cond1* y *cond2* retornan verdadero, se deposita una señal en las posiciones *ipp1* e *ipp2*. Esto representa la ejecución de todos los caminos de interacción que forman parte del

constructor *Inclusive/Parallel*. En cambio, si *cond1* y *cond2* no son verdaderas, no se deposita ninguna señal en las posiciones *ipp1* y *ipp2* y por lo tanto quedan vacías. Esto representa que no se ejecutan los caminos de interacción que forman parte del *Inclusive/Parallel*. Las variables *cond1* y *cond2* son independientes entre sí, lo que permite que se ejecuten uno, ambos o ninguno de los caminos del *Inclusive/Parallel*. Una vez que finaliza la ejecución de todos los caminos, se habilita la transición *ot* que permite llevar a cabo la sincronización y depositar una señal en la posición *op*.



a) Constructor Inclusive/Parallel con dos caminos



b) Módulo GI concreto del Inclusive/Parallel

Fig. A.23. Ejemplo de Inclusive/Parallel en BPMN y su módulo GI concreto

Inclusive/Exclusive

La Figura **A.24** muestra el módulo GI abstracto del constructor estructurado de BPMN que se forma a partir de la combinación de un *Inclusive gateway* y un *Exclusive*

gateway. Este módulo GI abstracto difiere del constructor *Inclusive/Parallel* en que la transición *t* se conecta directamente a la posición *op*. La descripción de los demás elementos es análoga a la de dicho constructor.

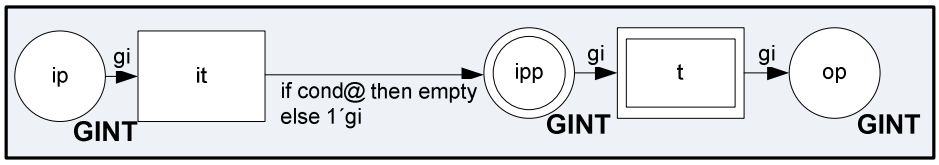
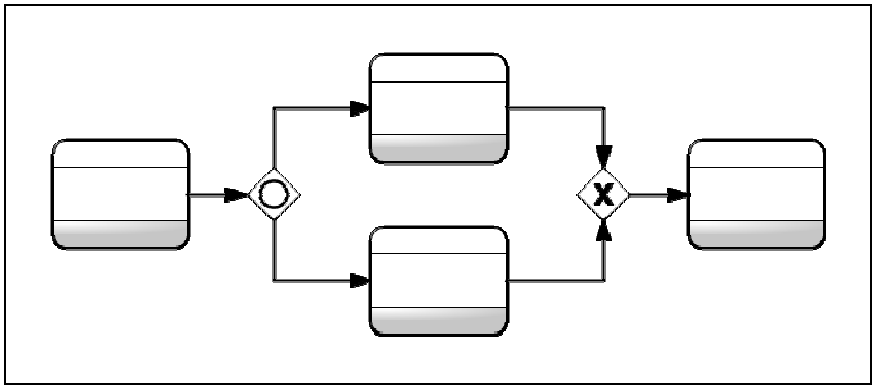


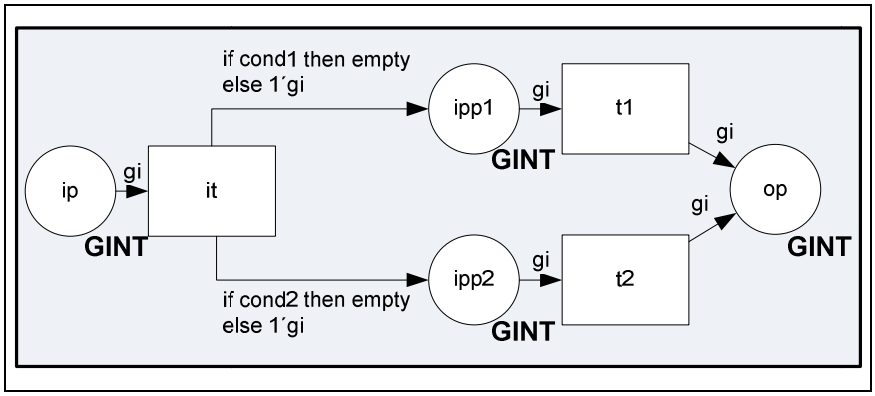
Fig. A.24. Módulo GI abstracto de Inclusive/Exclusive

La Figura **A.25** muestra un ejemplo de un constructor estructurado de BPMN compuesto de un *Inclusive gateway* con un *Exclusive gateway* con dos flujos de secuencia paralelos (Figura **A.25** a)) y su respectivo módulo GI concreto que lo formaliza (Figura **A.25** b)). La estructura de este módulo GI concreto es similar a la del módulo presentado en la Figura **A.23**. La diferencia reside en que las transiciones *t1* y *t2* se conectan directamente a la posición final *op*.

Si hay una señal en la posición *ip*, se puede ejecutar la transición *it*. Si las variables booleanas *cond1* y *cond2* retornan verdadero, se deposita una señal en las posiciones *ipp1* e *ipp2*. Esto representa la ejecución de todos los caminos de interacción que forman parte del constructor *Inclusive/Exclusive*. Si *cond1* y *cond2* no son verdaderas, no se deposita ninguna señal en las posiciones *ipp1* y *ipp2* y por lo tanto quedan vacías. Esto representa que no se ejecutan los caminos de interacción que forman parte del *Inclusive/Exclusive*. Las variables *cond1* y *cond2* son independientes entre sí, lo que permite que se ejecuten uno, ambos o ninguno de los caminos del *Inclusive/Exclusive*. Una vez que finaliza la ejecución de todos los caminos, se deposita una señal en la posición *op*.



a) Constructor Inclusive/Exclusive con dos caminos



b) Módulo GI concreto del Inclusive/Exclusive

Fig. A.25. Ejemplo de Inclusive/Exclusive en BPMN y su módulo GI concreto

Anexo B. Algoritmos para Detectar Anti-Patrones de Comportamiento en UP-ColBPIP

En este anexo se presentan algoritmos para la detección de anti-patrones de comportamiento en el lenguaje UP-ColBPIP.

Detección del Anti-Patrón AP1

La función *isBlockedSequence* que se muestra en el Algoritmo **B.1** permite detectar el anti-patrón *AP1* a partir de un elemento de PNC que recibe como entrada. Si el anti-patrón se cumple la función devuelve *true*, caso contrario devuelve *false*.

Algoritmo B.1. Pseudocódigo para detectar el anti-patrón AP1.

```
Function isBlockedSequence (element)
if (element=Termination) and Parent(element)=Sequence
        and Successor(element)≠ ∅ then
  return true
else
  return false
end if
```

Detección de los Anti-Patrones AP2 y AP3

El Algoritmo **B.2** define la función *isBlockedCycle* que permite detectar los anti-patrones *AP2* y *AP3* a partir de un elemento de PNC que recibe como entrada. Si uno de estos anti-patrones se cumple la función devuelve *true*, caso contrario devuelve *false*.

Algoritmo B.2. Pseudocódigo para detectar los anti-patrones AP2 y AP3.

```
Function isBlockedCycle (element)
if (element=Termination)
  s = Parent(element)
  if (Parent(s)=Loop-While or Parent(s)=Loop-Until)) then
    retornar true
sino
    retornar false
```

Detección de los Anti-Patrones AP4, AP5, AP6, y AP7

La función *isBlockedParallelism* que se muestra en el Algoritmo **B.3** permite detectar los anti-patrones AP4, AP5, AP6, y AP7 a partir de un elemento de PNC que recibe como entrada. Si alguno de los anti-patrones se cumple la función devuelve *true*, caso contrario devuelve *false*.

Algoritmo B.3. Pseudocódigo para detectar los anti-patrones AP4, AP5, AP6, y AP7.

```
Function isBlockedParallelism (element)
if (element=Termination or element=Cancel) then
  for all n such that n ∈ Ancestors(element) do
    if (n=Sequence) then
      if (Parent(n)=And or Parent(n)=Or/SyncMerge) then
        return true
    end if
  end
end if
return false
```

Detección del Anti-Patrón AP8

La función *isBlockedMutualExclusion* que se muestra en el Algoritmo **B.4** permite detectar el anti-patrón *AP8* a partir de un elemento de PNC que recibe como entrada. Si el anti-patrón se cumple la función devuelve *true*, caso contrario devuelve *false*. La función *getTreePaths(element)* devuelve todos los caminos del árbol que se inician en el elemento *element* y finalizan en un nodo hoja.

Algoritmo B.4. Pseudocódigo para detectar el anti-patrón AP8.

```
Function isBlockedMutualExclusion (element)
if (element=Xor) then
  for all path such that path ∈ getTreePaths(element) do
    term ⟵ false
    for all n such that n ∈ path do
      if (n=Termination) then
        term ⟵ true
        exit for
      end if
    end
    if (term=false) then
      return false
    end if
  end

  return term

end if
```

Detección de los Anti-Patrones AP9 y AP10

La función *isBlockedMultipleInstances* que se muestra en el Algoritmo **B.5** permite detectar los anti-patrones *AP9* y *AP10* a partir de un elemento de PNC que recibe como entrada. Si el PNC contiene alguno de los anti-patrones la función devuelve *true*, caso contrario devuelve *false*.

Algoritmo B.5. Pseudocódigo para detectar los anti-patrones AP9 y AP10.

```
Function isBlockedMultipleInstances (element)
if (element=Termination or element=Cancel) then
  for all n such that n ∈ Ancestors(element) do
    if (n=Sequence) then
      if (Parent(n)=MI then
        return true
    end if
  end
end if
return false
```

Detección de los Anti-Patrones AP11 y AP12

La función *isBlockedExceptionHandler* que se muestra en el Algoritmo **B.6** permite detectar un bloqueo en los manejadores de excepciones de un constructor *Exception* a partir de un elemento de PNC que recibe como entrada. Si el PNC contiene al anti-patrón AP11 de la Figura **5.22** a) la función devuelve *true*, caso contrario devuelve *false*. La función *isBlockedCancelHandler* que se muestra en el Algoritmo **B.7** permite detectar un bloqueo en los manejadores de excepciones de un constructor *Cancel* a partir de un elemento de PNC que recibe como entrada. Si el PNC contiene al anti-patrón AP12 de la Figura **5.22** b) la función devuelve *true*, caso contrario devuelve *false*.

Algoritmo B.6. Pseudocódigo para detectar el anti-patrón AP11.

```
Function isBlockedExceptionHandler (element)
if (element=Exception) then
  for all path such that path ∈ getTreePaths(element) do
    term ⟵ false
    for all n such that n ∈ path do
      if (n=Termination) then
        term ⟵ true
        exit for
      end if
    end
    if (term=false) then
      return false
    end if
  end
  return term
end if
```

Algoritmo B.7. Pseudocódigo para detectar el anti-patrón AP12.

```
Function isBlockedCancelHandler (element)
if (element=Termination and parent(element)=Handler) then
  s=parent(element)
  if (parent(s)=Cancel) then
    return true
  else
    return false
  end if
end if
```

Bibliografía

Awad, A & Puhlmann, F 2008, 'Structural Detection of Deadlocks in Business Process Models', *11th International Conference, BIS 2008*, Springer Berlin Heidelberg, Innsbruck, Austria.

Basten, T & van der Aalst, W 2001, 'Inheritance of behavior', *Journal of Logic and Algebraic Programming*, vol 47, no. 2, pp. 47-145.

Bauer, B, Roser, S & Müller, J 2005, 'Adaptive Design of Cross-Organizational Business Processes using a Model-Driven Architecture', in OK Ferstl, EJ Sinz, S Eckert, T Isselhorst (eds.), *Wirtschaftsinformatik 2005: eEconomy, eGovernment, eSociety*, Physica-Verlag HD.

Bergstra, JA & Klop, JW 1984, 'Process algebra for synchronous communication', *Information and Control*, vol 60, no. 1-3, pp. 109-137.

Billington, J, Christensen, S, van Hee, KE, Kindler, E, Kummer, O, Petrucci, L, Post, R & Stehno, C 2003, 'The Petri Net Markup Language: Concepts, Technology, and Tools', *24th International Conference on Applications and Theory of Petri Nets*, Springer Berlin Heidelberg, Eindhoven, The Netherlands.

Breugel, F & Koshkina, M 2006, *Models and Verification of BPEL*, <http://www.cse.yorku.ca/~franck/research/drafts/tutorial.pdf>.

Brown, W, Malveau, R & Mowbray, T 1998, *AntiPatterns: refactoring software, architectures, and projects in crisis*, John Wiley & Sons, Hoboken, NJ.

Camarinha-Matos, LM, Nathalie Galeano, HA & Molina, A 2009, 'Collaborative networked organizations - Concepts and practice in manufacturing enterprises', *Computers & Industrial Engineering*, vol 57, no. 1, pp. 46-60.

Cambronero, ME, Díaz, G, Valero, V & Martínez, E 2011, 'Validation and verification of web services choreographies by using timed automata', *Journal of Logic and Algebraic Programming*, vol 80, no. 1, pp. 25-49.

Carbone, M, Honda, K & Yoshida, N 2008, 'Theoretical aspects of communication-centred programming', *Electronic Notes in Theoretical Computer Science*, vol 209, pp. 125-133.

Clarke, EM & Kurshan, R 1996, 'Computer-aided verification', *IEEE Spectrum*, vol 33, no. 6, pp. 61-67.

Corrales, J, Grigori, D & Bouzeghoub, M 2006, 'BPEL processes matchmaking for service discovery', *Proceedings of the OTM International Conferences*, Springer, Montpellier, France.

CPN Tools, viewed November 2013, <http://cpntools.org/>.

CPN Tools 2013, viewed November 2013, <http://cpntools.org/>.

Danylevych, O, Karastoyanova, D & Leymann, F 2010, 'Service Networks Modelling: An SOA & BPM Standpoint', *Journal of Universal Computer Science*, vol 16, no. 13, pp. 1668-1693.

Decker, G 2009, *Design and analysis of process choreographies*, University of Potsdam, Ph.D. thesis.

Decker, G, Kopp, O & Barros, A 2008, 'An Introduction to Service Choreographies', *Information Technology*, vol 50, no. 2, pp. 122-127.

Decker, G & Weske, M 2007, 'Behavioral consistency for B2B process integration', *Proceedings of the 19th international conference on Advanced information systems engineering*, Springer-Verlag, Trondheim, Norway.

Diaz, G, Pardo, J-J, Cambronero, M-E, Valero, V & Cuartero, F 2005, 'Automatic translation of WS-CDL choreographies to timed automata', *Formal Techniques for Computer Systems and Business Processes*, Springer Berlin Heidelberg, Versailles, France.

Dijkman, R 2007, 'A Classification of Differences between Similar Business Processes', *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC'07)*, IEEE Computer Society.

Dijkman, R 2008, 'Diagnosing differences between business process models', *Proceedings of the 6th International Conference on Business Process Management*, Springer-Verlag, Milan, Italy.

Dijkman, R, Dumas, M, Garcia-Banuelos, L & Kaarik, R 2009, 'Aligning Business Process Models', *IEEE International Enterprise Distributed Object Computing Conference (EDOC '09)*, IEEE Computer Society.

Dijkman, R, Dumas, M & Ouyang, C 2008, 'Semantics and analysis of business process models in BPMN', *Information & Software Technology*, vol 50, no. 12, pp. 1281-1294.

Dijkman, RM, Quartel, DAC, Pires, LF & van Sinderen, MJ 2004, 'A rigorous approach to relate enterprise and computational viewpoints', *Proceedings of Eighth IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004)*, IEEE Computer Society.

Dijkman, RM, Quartel, DAC, Pires, LF & van Sinderen, MJ 2004, 'A rigorous approach to relate enterprise and computational viewpoints', *Proceedings of Eighth IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004)*, IEEE Computer Society.

Dongen, BFV, Mendling, J & Aalst, WMPVD 2006, 'Structural Patterns for Soundness of Business Process Models', *Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference*, IEEE Computer Society, Washington, DC, USA.

Dumas, M, van der Aalst, W & ter Hofstede, AHM 2005, *Process-Aware Information Systems: Bridging People and Software through Process Technology*, John Wiley & Sons, Inc., New Jersey, USA.

ebXML 2002, <https://www.oasis-open.org/committees/download.php/272/ebMS_v2_0.pdf>.

Eclipse 2013, *Eclipse*, viewed November 2013, <http://www.eclipse.org>.

Eclipse , *Eclipse*, viewed November 2013, <http://www.eclipse.org>.

Ehrig, M, Koschmider, A & Oberweis, A 2007, 'Measuring similarity between semantic business process models', *Proceedings of the Fourth AsiaPacific Conference on Conceptual Modelling (APCCM 2007)*, Australian Computer Science Communications, Ballarat, Victoria, Australia.

Fahland, D, Favre, C, Jobstmann, B, Koehler, J, Lohmann, N, Völzer, H & Wolf, K 2009, 'Instantaneous Soundness Checking of Industrial Business Process Models', *Business Process Management*, Springer Berlin / Heidelberg.

Ferrante, J, Ottenstein, KJ & Warren, JD 1987, 'The program dependency graph and its uses in optimization', *ACM Transactions on Programming Languages and Systems*, vol 9, no. 3, p. 319–349.

Gamma, E, Helm, R, Johnson, R & Vlissides, J 1995, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley.

Girault, C & Valk, R 2001, *Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications*, Springer-Verlag New York.

Goldkuhl, GALM 2004, 'Developing e-interactions - a framework for business capabilities and exchanges', *Proceedings of 12th European Conference on information systems (ECIS2004)*, ECIS, Turku.

Graham, SL & Wegman, M 1976, 'A Fast and Usually Linear Algorithm for Global Flow Analysis', *J. ACM*, vol 23, no. 1, pp. 172-202.

Gruhn, V & Laue, R 2009, 'A Heuristic Method for Business Process Model Evaluation', *Advances in Enterprise Engineering III*, Springer Berlin Heidelberg, Amsterdam, The Netherlands.

Han, Z, Gong, P, Zhang, L, Ling, J & Huang, W 2013, 'Definition and Detection of Control-Flow Anti-patterns in Process Models', *IEEE 37th Annual Computer Software and Applications Conference Workshops (COMPSACW), 2013* , IEEE, Kyoto.

Harel, D & Rumpe, B 2004, 'Meaningful modeling: what's the semantics of "semantics"?', *IEEE Computer*, vol 37, no. 10, pp. 64-72.

Hauser, RF, Friess, M, Kuster, JM & Vanhatalo, J 2008, 'An Incremental Approach to the Analysis and Transformation of Workflows Using Region Trees', *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol 38, no. 3, pp. 347-359.

Hoffmann, CM & O'Donnell, MJ 1982, 'Pattern Matching in Trees', *Journal of the ACM*, vol 29, no. 1, pp. 68-95.

Holweg, M & Pil, FK 2001, 'Successful build-to-order strategies start with the customer', *MIT Sloan Management Review*, vol 43, no. 1, pp. 74-83.

Holzmann, G 2003, *The SPIN Model Checker: Primer and Reference Manual*, Addison-Wesley, Boston, Massachusetts.

Huemer, C, Liegl, P, Motal, T, Schuster, R & Zapletal, M 2008, 'The development process of the UN/CEFACT modeling methodology', *Proceedings of the 10th international conference on Electronic commerce*, ACM, Innsbruck, Austria.

Iacob, ME, Steen, MWA & Heerink, L 2008, 'Reusable model transformation patterns', *Proc. of the 12th Enterprise Distributed Object Computing*, IEEE Computer Society, Washington, DC, USA.

IBM 2013, *IBM WebSphere Business Modeler*, viewed November 2013, <http://www-03.ibm.com/software/products/en/modeler-advanced>.

Issarny, V, Georgantas, N, Hachem, S, Zarras, A, Vassiliadist, P, Autili, M, Gerosa, M & Hamida, A 2011, 'Service-oriented middleware for the Future Internet: state of the art and research directions', *Journal of Internet Services and Applications*, vol 2, no. 1, pp. 23-45.

Jensen, K & Kristensen, L 2009, *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*, Springer Publishing Company, Incorporated.

Johnson, R, Pearson, D & Pingali, K 1994, 'The Program Structure Tree: Computing Control Regions in Linear Time', *Proceedings of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation*, ACM, Orlando, Florida, USA.

Koehler, J & Vanhatalo, J 2007, 'Process anti-patterns: How to avoid the common traps of business process modeling', *IBM WebSphere Developer Technical Journal*, vol 10, no. 2.

Kopp, O, Martin, D, Wutke, D & Leymann, F 2009, 'The difference between Graph-Based and Block-Structured business process modelling languages', *Enterprise Modeling and Information systems*, vol 4, no. 1, pp. 3-13.

Kühne, S, Kern, H, Gruhn, V & Laue, R 2010, 'Business process modeling with continuous validation', *Journal of Software Maintenance and Evolution: Research and Practice*, vol 22, no. 6-7, pp. 547-566.

Kurtev, I 2005, *Adaptability of model transformations*, University of Twente, Enschede, Ph.D. Thesis.

Küster, J, Gerth, C, Förster, A & Engels, G 2008, 'Detecting and Resolving Process Model Differences in the Absence of a Change Log', *6th International Conference on Business Process Management*, Springer Berlin Heidelberg, Milan, Italy.

Larsen, KG, Pettersson, P & Yi, W 1997, 'Uppaal in a nutshell', *International Journal on Software Tools for Technology Transfer (STTT)*, pp. 134-152.

Laue, R & Awad, A 2010, 'Visualization of Business Process Modeling Anti Patterns', *Electronic Communications of the EASST*, vol 25.

Lazarte, IM, Tello-Leal, E, Roa, J, Chiotti, O & Villarreal, PD 2010, 'Model-Driven Development Methodology for B2B Collaborations', *Proceedings of 14th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW)*, IEEE Computer Society, Vitoria, Brazil.

Li, J, He, J, Zhu, H & Pu, G 2007, 'Modeling and verifying web services choreography using process algebra', *Proceedings of the 31st IEEE Software Engineering Workshop*, IEEE Computer Society, Columbia.

Lippe, S, Greiner, U & Barros, A 2005, 'A Survey on State of the Art to Facilitate Modelling of Cross-Organisational Business Processes', *Proceedings of the 2nd GI Workshop XML4BPM - XML Interchange Formats for Business Process Management at 11th GI Conference BTW 2005*, Karlsruhe, Germany.

Li, C, Reichert, MU & Wombacher, A 2007, 'On measuring process model similarity based on high-level change operations', Centre for Telematics and Information Technology, University of Twente, Technical Report TR-CTIT-07-89, Enschede.

Liu, R & Kumar, A 2005, 'An analysis and taxonomy of unstructured workflows', *Proceedings of the 3rd international conference on Business Process Management*, Springer-Verlag, Nancy, France.

Liu, C, Li, Q & Zhao, X 2009, 'Challenges and opportunities in collaborative business process management: Overview of recent advances and introduction to the special issue', *Information Systems Frontiers*, vol 11, no. 3, pp. 201-209.

Li, Q, Zhou, J, Peng, Q, Li, C, Wang, C, Wu, J & Shao, B 2010, 'Business processes oriented heterogeneous systems integration platform for networked enterprises', *Computers in Industry*, vol 61, no. 2, p. 127–144.

Lohmann, N, Verbeek, E & Dijkman, R 2009, 'Petri Net Transformations for Business Processes --- A Survey', in K Jensen, W van der Aalst (eds.), *Transactions on Petri Nets and Other Models of Concurrency II*, Springer Berlin Heidelberg.

Lohmann, N, Verbeek, E, Ouyang, C & Stahl, C 2009, 'Comparing and evaluating Petri net semantics for BPEL', *International Journal of Business Process Integration and Management*, vol 4, no. 1, pp. 60-73.

Lu, R & Sadiq, S 2007, 'On the discovery of preferred work practice through business process variants', *Conceptual Modeling (ER 2007)*, Springer.

Madhusudan, T, Zhao, L & Marshall, B 2004, 'A case-based reasoning framework for workflow model management', *Data Knowledge Engineering*, vol 50, no. 1, p. 87–115.

Medjahed, B, Benatallah, B, Bouguettaya, A, Ngu, AHH & Elmagarmid, AK 2003, 'Business-to-business interactions: issues and enabling technologies', *The VLDB Journal*, vol 12, no. 1, pp. 59-85.

Mendling, J 2008, *Metrics for process models*, Lecture Notes in Business Information Processing, Springer.

Mendling, J, Neumann, G & Aalst, W 2007, 'Understanding the occurrence of errors in process models based on metrics', *Proceedings of the OTM Conference*, Springer.

Mendling, J, Reijers, H & Cardoso, J 2007, 'What Makes Process Models Understandable?', *5th International Conference, BPM 2007*, Springer Berlin Heidelberg, Brisbane, Australia.

Mili, H, Tremblay, G, Jaoude, GB, Lefebvre, É, Elabed, L & Boussaidi, GE 2010, 'Business process modeling languages: Sorting through the alphabet soup', *ACM Computing Surveys*, vol 43, no. 1, pp. 4:1-4:56.

Minor, M, Tartakovski, A & Bergmann, R 2007, 'Representation and structure-based similarity assessment for agile workflows', *7th International Conference on Case-Based Reasoning*, Springer.

Muehlen, M & Recker, J 2008, 'How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation', *20th International Conference, CAiSE 2008* , Springer Berlin Heidelberg, Montpellier, France.

Murata, T 1989, 'Petri nets: Properties, analysis and applications', *Proceedings of the IEEE*, vol 77, no. 4, pp. 541-580.

Nejati, S, Sabetzadeh, M, Chechik, M, Easterbrook, S & Zave, P 2007, 'Matching and merging of statecharts specifications', *29th International Conference on Software Engineering*.

Norta, A & Eshuis, R 2010, 'Specification and verification of harmonized business-process collaborations', *Information Systems Frontiers*, vol 12, no. 4, pp. 457-479.

OASIS 2007, *Web Services Business Process Execution Language, version 2.0 (WSBPEL)* , <http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel>.

OMG 2011, *Unified Modeling Language*, viewed November 2013, <http://www.omg.org/spec/UML/>.

OMG-BPMN 2011, *Business Process Model and Notation (BPMN) version 2.0*, Object Management Group, Inc. (OMG), <www.omg.org/spec/BPMN/2.0/>.

Onoda, S, Ikkai, Y, Kobayashi, T & Komoda, N 1999, 'Definition of deadlock patterns for business processes workflow models', *Proceedings of the Thirty-second Annual Hawaii International Conference on System Sciences*, IEEE Computer Society, Washington, DC, USA.

Peterson, JL 1981, *Petri net theory and the modeling of systems*, Prentice-Hall, Englewood Cliffs.

Petri, CA 1962, *Kommunikation mit Automaten*, Institut für instrumentelle Mathematik, Bonn, PhD thesis.

PNML 2004, *PNML Standard, ISO/IEC 15909-1:2004*, viewed November 2013, <http://www.pnml.org/>.

PNML, *PNML Standard, ISO/IEC 15909-1:2004*, viewed November 2013, <http://www.pnml.org/>.

Polyvyanyy, A & Bussler, C 2013, 'The Structured Phase of Concurrency', *Seminal Contributions to Information Systems*, Springer-Verlag.

Polyvyanyy, A, Weidlich, M & Weske, M 2011, 'Connectivity of workflow nets: the foundations of stepwise verification', *Acta Informatica*, vol 48, no. 4, pp. 213-242.

Pomello, L, Rozenberg, G & Simone, C 1992, 'A survey of equivalence notions for net based systems', *Advances in Petri Nets 1992*, Springer-Verlag.

Roa, J, Chiotti, O & Villarreal, P 2012a, 'A Verification Method for Collaborative Business Processes', *Business Process Management Workshops*, Springer Berlin Heidelberg.

Roa, J, Chiotti, O & Villarreal, P 2012b, 'Behavior Alignment and Control Flow Verification of Process and Service Choreographies', *Journal of Universal Computer Science*, vol 18, no. 17, pp. 2383-2406.

Roa, J, Chiotti, O & Villarreal, P 2013, 'Verification of Structured Processes: A Method Based on an Unsoundness Profile', *14° Simposio Argentino de Ingeniería de Software (42° JAIIO)*, Córdoba, Argentina.

Rosenberg, F, Enzi, C, Michlmayr, A, Platzer, C & Dustdar, S 2007, 'Integrating quality of service aspects in top-down business process development using WS-CDL and WS-BPEL', *Eleventh IEEE International EDOC Enterprise Computing Conference, EDOC 2007*, IEEE Computer Society, Annapolis, Maryland, USA.

Roser, S & Bauer, B 2005, 'A Categorization of Collaborative Business Process Modelling Techniques', *7th IEEE International Conference on E-Commerce Technology Workshops*.

Russell, N, ter Hofstede, A, van der Aalst, W & Mulyar, N 2006, *Workflow control flow patterns: a revised review. BPM Center Report BPM-06-22*, <http://bpmcenter.org/reports>.

Sanders, N 2007, 'An empirical study of the impact of e-business technologies on organizational collaboration and performance', *Journal of Operations Management*, vol 25, no. 6, pp. 1332 - 1347.

Searle, J 1976, 'A taxonomy of illocutionary acts', *Language in Society*, vol 5, no. 1, pp. 1-23.

Selic, B 2003, 'The pragmatics of model-driven development', *IEEE Software*, vol 20, no. 5, pp. 19-25.

Steinberg, D, Budinsky, F, Paternostro, M & Merks, E 2008, *EMF: Eclipse Modeling Framework*, 2nd edn, Addison-Wesley Professional.

Stuit, M & Szirbik, N 2009, 'Towards agent-based modeling and verification of collaborative business processes: an approach centered on interactions and behaviors', *International journal of cooperative information systems*, vol 18, no. 03n04, pp. 423-479.

Su, J, Bultan, T, Fu, X & Zhao, X 2008, 'Towards a theory of web service choreographies', *Proceedings of the 4th international conference on Web services and formal methods*, Springer-Verlag, Brisbane, Australia.

Tello-Leal, E 2012, *Agentes de Software para la Gestion de Colaboraciones Inter-Organizacionales Dinamicas*, Universidad Tecnológica Nacional, Santa Fe, Argentina, Tesis doctoral.

Tello-Leal, E, Chiotti, O & Villarreal, PD 2012, 'Process-Oriented Integration and Coordination of Healthcare Services across Organizational Boundaries', *Journal of Medical Systems*, vol 36, no. 6, pp. 3713-3724.

Vallecillo, A 2008, 'A Journey through the Secret Life of Models', *Perspectives Workshop: Model Engineering of Complex Systems (MECS)*, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, Dagstuhl, Germany, http://drops.dagstuhl.de/opus/volltexte/2008/1601.

Valmari, A 1998, 'The state explosion problem', in W Reisig, G Rozenberg (eds.), *Lectures on Petri Nets I: Basic Models*, Springer Berlin Heidelberg, Dagstuhl, Germany.

van der Aalst, W 1997, 'Verification of workflow nets', *18th International Conference, ICATPN'97* , Springer Berlin Heidelberg, Toulouse, France.

Van der Aalst, WMP 1998, 'Modeling and analyzing interorganizational workflows', *Proceedings of the 1998 International Conference on Application of Concurrency to System Design*, IEEE Computer Society, Fukushima.

van der Aalst, W 1998, 'The Application of Petri Nets to Workflow Management', *Journal of Circuits, Systems and Computers*, vol 08, no. 01, pp. 21-66.

Van Der Aalst, WMP 2003, 'Inheritance of Interorganizational Workflows: How to Agree to Disagree Without Loosing Control?', *Inf. Technol. and Management*, vol 4, no. 4, pp. 345--389.

van der Aalst, WMALN, Massuthe, P, Stahl, C & Wolf, K 2010, 'Multiparty contracts: Agreeing and implementing interorganizational processes', *The Computer Journal*, vol 53, no. 1, pp. 90-106.

van der Aalst, W, van Hee, K, ter Hofstede, A, Sidorova, N, Verbeek, H, Voorhoeve, M & Wynn, M 2010, 'Soundness of workflow nets: classification, decidability, and analysis', *Formal Aspects of Computing*, vol 23, no. 3, pp. 333-363.

van Dongen, B, Dijkman, R & Mendling, J 2008, 'Measuring similarity between business process models', *Proceedings of the 20th International Conference on Advanced Information Systems Engineering (CAiSE)*, Springer.

van Glabbeek, RJ & Weijland, WP 1996, 'Branching time and abstraction in bisimulation semantics', *Journal of ACM*, vol 43, no. 3, pp. 555-600.

Vanhatalo, J, Völzer, H & Koehler, J 2009, 'The refined process structure tree', *Data & Knowledge Engineering*, vol 68, no. 9, pp. 793-818.

Vanhatalo, J, Völzer, H & Leymann, F 2007, 'Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition', *Proceedings of the 5th international conference on Service-Oriented Computing (ICSOC '07)*, Springer-Verlag, Vienna, Austria.

Varró, D & Pataricza, A 2003, 'Automated formal verification of model transformations', *CSDUML*.

Varró, D & Pataricza, A 2003, 'Automated Formal Verification of Model Transformations', *Proceedings of the UML'03 Workshop (CSDUML 2003): Critical Systems Development in UML*, Technische Universität München.

VICS, *An Overview of Collaborative Planning, Forecasting and Replenishment (CPFR)*, viewed 2013, <http://www.vics.org/docs/committees/cpfr/>.

Villarreal, P 2005, *Método para el Modelado y Especificación de Procesos Colaborativos*, Universidad Tecnológica Nacional, Santa Fe, Argentina.

Villarreal, P 2005, *Método para el modelado y especificación de procesos colaborativos*, Universidad Tecnológica Nacional, Santa Fe, Argentina, PhD Thesis.

Villarreal, PD, Lazarte, IM, Roa, J & Chiotti, O 2010, 'A Modeling Approach for Collaborative Business Processes Based on the UP-ColBPIP Language', in S Rinderle-Ma, S Sadiq, F Leymann, W van der Aalst, J Mylopoulos, M Rosemann, M Shaw, C Szypers-Ki (eds.), *Business Processes Management Workshops*, Springer Berlin Heidelberg.

Villarreal, P, Lazarte, I, Roa, J & Chiotti, O 2010, 'A Modeling Approach for Collaborative Business Processes Based on the UP-ColBPIP Language', *Business Process Management Workshops*, Springer Berlin Heidelberg.

Villarreal, P, Salomone, E & Chiotti, O 2005, 'Applying Model-Driven Development to Collaborative Business Processes', *Proceedings 8º Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software (IDEAS´05)*, Valparaiso, Chile.

Villarreal, P, Salomone, E & Chiotti, O 2006, 'A MDA-based Development Process for Collaborative Business Processes', *Proceedings of the European Workshop on Milestone, Models and Mappings for Model-Driven Architecture (3M4MDA).*

Villarreal, P, Salomone, E & Chiotti, O 2006, 'MDA Approach for Collaborative Business Processes: Generating Technological Solutions based on Web Services Composition', *Proceedings 9º Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software (IDEAS´06)*, La Plata, Argentina.

Villarreal, PD, Salomone, E & Chiotti, O 2006, 'Transforming Collaborative Business Proccess Models into Web Services Choreography Specifications', *Proceedings of the Second International Workshop, DEECS 2006, San Francisco, CA, USA*, Springer Berlin Heidelberg.

Villarreal, PD, Salomone, EH & Chiotti, O 2007, 'Modeling and Specifications of Collaborative Business Processes using a MDA Approach and a UML Profile', in *Enterprise Modeling and Computing with UML*, Idea Group Inc., USA.

W3C-WSCDL 2005, *Web Services Choreography Description Language Version 1.0*, <http://www.w3.org/TR/ws-cdl-10/>.

Weber, B, Reichert, M & Rinderle-Ma, S 2008, 'Change patterns and change support features - enhancing flexibility in processaware information systems', *Data & Knowledge Engeneering*, vol 66, no. 3, p. 438–466.

Weidlich, M, Barros, A, Mendling, J & Weske, M 2009, 'Vertical alignment of process models - how can we get there?', *Enterprise, Business Process and Information Systems*, Springer Berlin Heidelberg.

Weidlich, M, Dijkman, R & Weske, M 2010, 'Deciding Behaviour Compatibility of Complex Correspondences between Process Models', *Business Process Management*, Springer Berlin Heidelberg, Hoboken, NJ, USA.

Weidlich, MADG, Großkopf, A & Weske, M 2008, 'BPEL to BPMN: The Myth of a Straight-Forward Mapping', in R Meersman, Z Tari (eds.), *On the Move to Meaningful Internet Systems: OTM 2008*, Springer Berlin Heidelberg.

Weske, M 2007, *Business Process Management. Concepts, Languages, Architectures*, Springer.

Wombacher, A 2006, 'Evaluation of technical measures for workflow similarity based on a pilot study', *International Conference on Cooperative Information Systems (CoopIS'06)*, Springer.

Woodcock, J, Larsen, PG, Bicarregui, J & Fitzgerald, J 2009, 'Formal methods: Practice and experience', *ACM Computing Surveys*, vol 41, no. 4, pp. 19:1--19:36.

Wynn, MT, Verbeek, HMW, van der Aalst, WM, ter Hofstede, AH & Edmond, D 2009, 'Business process verification–finally a reality!', *Business Process Management Journal*, vol 15, no. 1, pp. 74-92.

Yang, H, Zhao, X, Qiu, Z, Pu, G & Wang, S 2006, 'A formal model for web service choreography description language (WS-CDL)', *International Conference on Web Services (ICWS '06)*, Chicago.

Yang, H, Zhao, X, Qiu, Z, Pu, G & Wang, S 2006, 'A Formal Model forWeb Service Choreography Description Language (WS-CDL)', *International Conference on Web Services, 2006 (ICWS '06)*, IEEE Computer Society, Chicago.

Zaha, J, Dumas, M, ter Hofstede, A, Barros, A & Decker, G 2006, 'Service Interaction Modeling: Bridging Global and Local Views', *10th IEEE International Enterprise Distributed Object Computing Conference (EDOC '06)*, IEEE, Hong Kong.

Zeller, A 1999, 'Yesterday, my program worked. Today, it does not. Why?', *SIGSOFT Softw. Eng. Notes* , vol 24, no. 6, pp. 253-267.

Zeller, A & Hildebrandt, R 2002, 'Simplifying and Isolating Failure-Inducing Input', *IEEE Trans. Softw. Eng.*, vol 28, no. 2, pp. 183-200.

Zhao, W, Hauser, R, Bhattacharya, K, Bryant, BR & Cao, F 2006, 'Compiling Business Processes: Untangling Unstructured Loops in Irreducible Flow Graphs', *Int. J. Web Grid Serv.*, vol 2, no. 1, pp. 68-91.